HAL
open science

Data centers energy optimization

Léa Bayati

To cite this version:

Léa Bayati. Data centers energy optimization. Operations Research [math.OC]. Université Paris-Est, 2019. English. NNT : 2019PESC0063 . tel-03120640

HAL Id: tel-03120640

https://theses.hal.science/tel-03120640v1

Submitted on 25 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université Paris-EstÉcole doctorale MSTIC

Thèse de Doctorat

Pour obtenir le titre de Docteur d'Université Paris-Est
Spécialité : Informatique

Data Centers Energy Optimization

Optimisation de l'énergie dans les centres de données

Défendue par

LÉA MARZIYEH BAYATI

Soutenue le 20 septembre 2019 devant le jury composé de :

Rapporteur	ALAIN JEAN-MARIE	Directeur de Recherche	INRIA Sophia Antipolis
Rapporteur	KAMEL BARKAOUI	Professeur	CNAM
Examinateur	PIERRE VALARCHER	Professeur	UPEC
Examinateur	HIND CASTEL-TALEB	Professeur	TSP-Telecom SudParis
Examinateur	JEAN-MICHEL FOURNEAU	Professeur	UVSQ
Examinateur	BENOÎT BARBOT	Maître de conférences	UPEC
Directrice	NIHAL PEKERGIN	Professeur	UPEC

Contents

1 Introduction **1**
1.1 Context of work: Background and Generalities 1
1.1.1 Background (Cloud in real life) 1
1.1.2 Data center 2
1.2 Generalities and problems 5
1.2.1 Queuing model theory 5
1.2.2 Policy and strategy 6
1.2.3 Optimality 7
1.2.4 Metrics 7
1.3 Queue model 8
1.3.1 Evolution equations 9
1.3.2 Histogram operators 9
1.3.3 Description of Arrivals 13
1.3.4 Service rate Description 16
1.3.5 Energy and QoS metrics 17
1.4 Outline and solution 18
1.4.1 Proposed solutions 18
1.4.2 Outline 20

2 State of the art **21**
2.1 Analytical solutions based on CTMC 22
2.1.1 Thresholds policy without latency of servers 22
2.1.2 Thresholds policy with latency of servers 24
2.1.3 Allocation policy with rejected jobs 27
2.1.4 Impact of data center size on the effectiveness of DPM . . 29
2.1.5 Multiple arrival jobs process 30
2.2 Continuous time MDP solutions 32
2.3 Discrete time MDP solutions 34

2.3.1 Constrained optimization 34
2.3.2 Probabilistic model checking for DPM 36
2.4 Papers consolidation 38

3 Thresholds policy **42**
3.1 Thresholds policy based on Markov chain 42
3.1.1 Optimization mechanism 43
3.1.2 Application example 46
3.2 Thresholds policy based on histogram operators 51
3.2.1 Problem specification 51
3.2.2 Optimization specification 54
3.2.2.1 Time interval analysis by coupling 55
3.2.2.2 Numerical analysis 57
3.2.3 Case study 60
3.3 Conclusion 65

4 Markov Decision Process **66**
4.1 MDP for homogeneous data center 66
4.1.1 Problem specification 67
4.1.1.1 Energy and Performance metric 68
4.1.1.2 Objective function 68
4.1.2 Optimization approach 69
4.1.2.1 Modelization 69
4.1.2.2 Space Complexity 72
4.1.2.3 Solver algorithm 72
4.1.3 Example 74
4.1.4 Optimal Strategy Structure 75
4.1.4.1 Monotone policy example 77
4.1.4.2 Optimal policy and Monotony 78
4.1.5 Experimental results 82
4.1.5.1 MDP under PRISM 82
4.1.5.2 Small Model 84
4.1.5.3 Medium Model 90
4.1.5.4 Big Model 96
4.1.5.5 Comparison with the threshold strategy 99
4.2 MDP for heterogeneous data center 101
4.2.1 Generalization of the problem 102
4.2.2 Optimization approach 104
4.2.2.1 Modelization 104
4.2.2.2 Memory complexity 105
4.2.2.3 Solver algorithm 107

4.2.3 Example . . . 107
4.2.4 Optimal strategy structure . . . 108
4.2.5 Test and results . . . 108
4.3 MDP for data center with latency . . . 114
4.3.1 Problem extension . . . 114
4.3.2 Optimization approach . . . 116
4.3.2.1 Modelization . . . 116
4.3.2.2 Space state complexity . . . 119
4.3.3 Example . . . 120
4.3.4 Experimental application . . . 123
4.4 MDP for variable arrival distribution . . . 124
4.5 Lessons learn from this work . . . 129

5 Greedy-window optimization algorithm 131
5.1 Problem specification . . . 131
5.1.1 Energy and Performance metric . . . 133
5.1.2 Objective function . . . 133
5.2 Optimization approach . . . 134
5.2.1 Truncated value iteration . . . 134
5.2.2 Greedy optimization . . . 138
5.2.3 Greedy-window optimization . . . 141
5.3 Experimental results . . . 146
5.3.1 Experiments for Dirac arrivals job . . . 146
5.3.1.1 Optimization and load factor . . . 147
5.3.1.2 Order of magnitude of costs . . . 148
5.3.2 Experiments for i.i.d. arrival distribution . . . 150
5.3.2.1 Load factor variation . . . 151
5.3.2.2 Cost variation . . . 153
5.3.3 Experiments for variable arrival distribution . . . 158
5.3.3.1 Hourly arrival variation . . . 158
5.3.3.2 Daily arrival variation . . . 160

6 Comparison of methods and Conclusion 167
6.1 Final result . . . 168
6.1.1 First Experimentation . . . 171
6.1.2 Second Experimentation . . . 174
6.1.3 Third Experimentation . . . 175
6.1.4 Fourth Experimentation . . . 178
6.1.5 Fifth Experimentation . . . 181
6.1.6 Sixth Experimentation . . . 187
6.2 Contributions . . . 192

6.3 Future work 193

APPENDICES **201**

A Schwartz et al. paper detail [SPTG12] **202**

B Mitrani's paper detail [Mit13] **205**

C Including lost into the state of the system **209**

D MDP for variable arrival distribution **211**
D.1 Problem specification 211
D.1.1 Energy and Performance metric 212
D.1.2 Objective function 212
D.2 Optimization 213
D.3 First approach 214
D.3.1 Modelization 214
D.3.2 Space and time complexities 215
D.3.3 Example 216
D.4 Second approach 217
D.4.1 Modelization 217
D.4.2 Space and time complexities 219
D.4.3 Example 220

E Additional material **222**
E.1 Energy 222

List of Tables

1.1 Costs considered over one slot. 17

2.1 Experimental simulation settings in [SPTG12]. 23
2.2 Experimental simulation settings in [Mit13]. 26
2.3 Experimental simulation settings in [DM10]. 28
2.4 Experimental numerical settings in [GHB11]. 30
2.5 Experimental simulation settings in [YCNH11]. 33
2.6 Latency in [NPK+05] 36
2.7 Average of energy consumption in [NPK+05] 36
2.8 Paper consolidation 40
2.9 Paper collection 41

3.1 Model and DTMC Parameters. 45
3.2 Thresholds policy settings test based on Markov chain 46
3.3 Thresholds policy comparison based on Markov chain 46
3.4 Settings of the study case 61

4.1 Model and MDP Parameters. 71
4.2 Settings of the first numerical analysis for small data center. 84
4.3 Result of the numerical analysis for small data center. 86
4.4 Settings of the numerical analysis for a medium data center. 91
4.5 Result of the numerical analysis for a medium data center. 91
4.6 Settings of the numerical analysis for big data center. 97
4.7 Result of the numerical analysis for big data center. 97
4.8 Settings of different data centers parameters. 99
4.9 Result of the numerical analysis 100
4.10 Energy and Performance metric for heterogeneous model 103
4.11 Model and MDP Parameters. 106
4.12 Parameters of the numerical analysis. 110

4.13 Result of the numerical analysis. 111
4.14 Model and MDP Parameters. 119
4.15 Parameters of the numerical analysis. 123
4.16 Approaches comparison 124

5.1 Example of optimization greedy-window algorithm 139
5.2 Workload of system according to ρ. 146
5.3 Impact of costs on the number of operational servers. 148
5.4 Settings of the first numerical analysis. 151
5.5 Settings of the second numerical analysis. 153
5.6 Settings of the third numerical analysis. 159
5.7 Example of arrivals daily variation (Google trace) 160
5.8 Settings of the last numerical analysis. 165

6.1 Time complexity comparison 168
6.2 Space complexity comparison 169
6.3 Time complexity comparison. 169
6.4 Settings of the numerical analysis. 171
6.5 Settings of the numerical analysis. 174
6.6 Settings of the numerical analysis. 175
6.7 Settings of the numerical analysis. 178
6.8 Settings of the numerical analysis. 181
6.9 Settings of the numerical analysis. 187

A.1 Experimental simulation settings in [SPTG12]. 204

B.1 Experimental simulation settings in [Mit13]. 207

D.1 Approaches comparison 213
D.2 Model and MDP Parameters. 219

E.1 Units of energy 223
E.2 List of some of the most important symbols and notations 224

List of Figures

1.1 Cooling servers 3
1.2 Overview of a data center 4
1.3 Data center access 4
1.4 Controlling by action mechanism 6
1.5 Illustration of the generic queuing model 10
1.6 Convolution FFT-based method 14
1.7 Google trace arrivals analysis 15
1.8 Google trace arrivals and i.i.d.-ness test 16

3.1 β/σ mechanism 44
3.2 DTMC for $\sigma = 3$ and $\beta = 3$ 47
3.3 DTMC for $\sigma = 0$ and $\beta = 1$ 48
3.4 DTMC for $\sigma = 1$ et $\beta = 2$ 49
3.5 Cost comparison for a finite horizon 50
3.6 Up/Down mechanism 54
3.7 Coupling and stationarity detection 56
3.8 Google trace arrivals and i.i.d.-ness test for nighttime and daytime 61
3.9 Arrival traffic distribution 62
3.10 Threshold results 63
3.11 Operational servers number evolution 64
3.12 Operational servers number evolution 64

4.1 Illustration of the queuing model. 67
4.2 MDP example for a data center with two servers. 75
4.3 Monotony/non-monotony of the optimal strategy 81
4.4 First MDP experiment for small data center 86
4.5 Second MDP experiment for small data center 87
4.6 Third MDP experiment for small data center 88
4.7 Fourth MDP experiment for small data center 89

4.8 Arrival traffic distribution 90
4.9 First MDP experiment for a medium data center 92
4.10 Second MDP experiment for a medium data center 93
4.11 Third MDP experiment for a medium data center 94
4.12 Fourth MDP experiment for a medium data center 95
4.13 Arrival traffic distribution 96
4.14 PRISM out of memory 98
4.15 Arrival traffic distribution 99
4.16 Cost comparison 100
4.17 Google trace servers repartition 101
4.18 Illustration of the queuing model. 102
4.19 Depiction of an MDP action 105
4.20 MDP example. 109
4.21 Arrival traffic distribution 110
4.22 Unitary costs variation 112
4.23 Heterogeneous and homogeneous servers comparison 113
4.24 Server latency 115
4.25 Illustration of the queuing model. 115
4.26 MDP example for latency 122
4.27 Result of numerical results on latency model 123
4.28 Arrival traffic for different hours 125
4.29 Arrival traffic for different minutes 125
4.30 Illustration of the generic i.i.d. arrival jobs 126
4.31 Illustration of a generic variable arrival jobs 126

5.1 Illustration of the queuing model. 132
5.2 Paths visited by MDP 136
5.3 Paths visited by Truncated MDP 137
5.4 Paths visited by greedy optimization 140
5.5 Paths visited by Greedy-window optimization 142
5.6 Relationship between window and optimization efficiency 144
5.7 Operational servers number and load factor 147
5.8 Arrival traffic distribution 150
5.9 Operational servers number depending on the load factor 151
5.10 Evolution of the number operational server depending on the load factor 152
5.11 Arrival traffic distribution 153
5.12 Operational servers number evolution depending on waiting cost . 154
5.13 Operational servers number evolution depending on loss cost . . . 155
5.14 Inertia phenomenon for $b = 1000$ 156
5.15 Inertia phenomenon for $b = 3000$ 156

5.16 Inertia phenomenon for $b = 5000$ 157
5.17 Inertia phenomenon for $b = 7000$ 157
5.18 Arrival traffic distribution 158
5.19 Operational servers number evolution during one day 159
5.20 Google trace arrivals and i.i.d.-ness test for the Sunday days. . . . 160
5.21 Google trace arrivals and i.i.d.-ness test for the Saturday days. . . 161
5.22 Google trace arrivals and i.i.d.-ness test for the Tuesday days. . . . 161
5.23 Google trace arrivals and i.i.d.-ness test for the Wednesday days. . 162
5.24 Google trace arrivals and i.i.d.-ness test for the Thursday days. . . 162
5.25 Google trace arrivals and i.i.d.-ness test for Friday days. 163
5.26 Google trace arrivals and i.i.d.-ness test for the Monday days. . . . 163
5.27 Cumulative distribution for week days 164
5.28 Evolution over a week of the number of operational servers 165

6.1 Arrival traffic distribution 170
6.2 Cost comparison 171
6.3 Space and computation-time comparison 172
6.4 Threshold results 173
6.5 Cost comparison 174
6.8 Cost comparison 175
6.6 Space and computation-time comparison 176
6.7 Threshold results 177
6.11 Cost comparison 178
6.9 Space and computation-time comparison 179
6.10 Threshold results 180
6.14 Arrival traffic distribution 181
6.12 Space and computation-time comparison 182
6.13 Threshold results 183
6.15 Cost comparison 184
6.16 Space and computation-time comparison 185
6.17 Threshold results 186
6.18 Arrival traffic distribution 187
6.19 Cost comparison 188
6.20 Space and computation-time comparison 189
6.21 Threshold results 190
6.22 Average cost per slot for MDP and window-greedy algorithm . . . 191
6.23 Deployment schema 193

C.1 Comparison of the number of states 210
C.2 Cost comparison 210

D.1 MDP example for arrival distribution that change over time 217
D.2 MDP example for arrival distribution that change over time 217
D.3 MDP example for arrival distribution that change over time 217
D.4 MDP example for arrival distribution that change over time. . . . 221

List of Algorithms

1 Example of optimal strategy given in [NPK+05]. 38

2 Thresholds policy management algorithm 44
3 Thresholds policy management algorithm 54
4 Computation with coupling detection 57
5 Main function 58

6 Value iteration algorithm for MDP. 73
7 Value iteration algorithm for MDP. 128

8 Truncated value iteration algorithm. 135
9 Greedy optimization algorithm for one slot. 138
10 Greedy-window optimization algorithm. 141
11 The online version of the greedy-window algorithm. 145

12 Value iteration algorithm for MDP. 215

Listings

2.1 Service Requester 37
2.2 Service Request Queue 37
4.1 Expected reachability properties in PRISM 82
4.2 Instantaneous properties 82
4.3 Cumulative properties 82
4.4 Reachability properties 83
4.5 Formula to check for Small Data Center 84
4.6 Example of a simple PRISM specification. 85
4.7 Formula to check for a medium Data Center 91
4.8 Formula to check for Big Data Center 97
4.9 Example of PRISM specification for a model with latency. 120

Abstract

To ensure both good data center service performance and reasonable power consumption, a detailed analysis of the behavior of these systems is essential for the design of efficient optimization algorithms to reduce energy consumption. This thesis fits into this context, and our main work is to design dynamic energy management systems based on stochastic models of controlled queues. The goal is to search for optimal control policies for data center management, which should meet the growing demands of reducing energy consumption and digital pollution while maintaining quality of service. We first focused on the modeling of dynamic energy management by a stochastic model for a homogeneous data center, mainly to study some structural properties of the optimal strategy, such as monotony. Afterwards, since data centers have a significant level of server heterogeneity in terms of energy consumption and service rates, we have generalized the homogeneous model to a heterogeneous model. In addition, since the data center server's wake-up and shutdown are not instantaneous and a server requires a little more time to go from sleep mode to ready-to-work mode, we have extended the model to the purpose of including this server time latency. Throughout this exact optimization, arrivals and service rates are specified with histograms that can be obtained from actual traces, empirical data, or traffic measurements. We have shown that the size of the MDP model is large and leads to the problem of the explosion of state space and a large computation time. Thus, we have shown that optimal optimization requiring a MDP is often difficult or almost impossible to apply for large data centers. Especially if we take into account real aspects such as server heterogeneity or latency. So, we have suggested what we call the greedy-window algorithm that allows to find a sub-optimal strategy better than that produced when considering a special mechanism like threshold approaches. And more importantly, unlike the MDP approach, this algorithm does not require the complete construction of the structure that encodes all possible strategies. Thus, this algorithm gives a strategy very close to the optimal strategy with low space-time complexities. This makes this solution practical, scalable, dynamic and can be put online.

Résumé

Pour garantir à la fois une bonne performance des services offerts par des centres de données, et une consommation énergétique raisonnable, une analyse détaillée du comportement de ces systèmes est essentielle pour la conception d'algorithmes d'optimisation efficaces permettant de réduire la consommation énergétique. Cette thèse s'inscrit dans ce contexte, et notre travail principal consiste à concevoir des systèmes de gestion dynamique de l'énergie basés sur des modèles stochastiques de files d'attente contrôlées. Le but est de rechercher les politiques de contrôle optimales afin de les appliquer sur des centres de données, ce qui devrait répondre aux demandes croissantes de réduction de la consommation énergétique et de la pollution numérique tout en préservant la qualité de service. Nous nous sommes intéressés d'abord à la modélisation de la gestion dynamique de l'énergie par un modèle stochastique pour un centre de données homogène, principalement pour étudier certaines propriétés structurelles de la stratégie optimale, telle que la monotonie. Après, comme des centres de données présentent un niveau non négligeable d'hétérogénéité de serveurs en termes de consommation d'énergie et de taux de service, nous avons généralisé le modèle homogène à un modèle hétérogène. De plus, comme le réveil (resp. l'arrêt) d'un serveur de centre de données n'est pas instantané et nécessite un peu plus de temps pour passer du mode veille au mode prêt à fonctionner, nous avons étendu le modèle dans le but d'inclure cette latence temporelle des serveurs. Tout au long de cette optimisation exacte, les arrivées et les taux de service sont spécifiés avec des histogrammes pouvant être obtenus à partir de traces réelles, de données empiriques ou de mesures de trafic entrant. Nous avons montré que la taille du modèle MDP est très grande et conduit au problème de l'explosion d'espace d'états et à un temps de calcul important. Ainsi, nous avons montré que l'optimisation optimale nécessitant le passage par un MDP est souvent difficile, voire pratiquement impossible pour les grands centres de données. Surtout si nous prenons en compte des aspects réels tels que l'hétérogénéité ou la latence des serveurs. Alors, nous avons suggéré ce que nous appelons l'algorithme greedy-window qui permet de trouver une stratégie sous-optimale meilleure que celle produite lorsqu'on envisage un mécanisme spécial comme les approches à seuil. Et plus important encore, contrairement à l'approche MDP, cet algorithme n'exige pas la construction complète de la structure qui encode toutes les stratégies possibles. Ainsi, cet algorithme donne une stratégie très proche de la stratégie optimale avec des complexités spatio-temporelles faibles. Cela rend cette solution pratique et permet son implémentation dans un contexte à temps réel.

Résumé long (français)

Le développement croissant des centres de données et l'expansion récente des *clouds* posent des problèmes énergétiques et des problèmes de pollution numérique.

Plusieurs études montrent une augmentation significative de la consommation d'énergie et de la pollution numérique générée par des ordinateurs. Plus de 1.3% de la consommation mondiale d'énergie est due à l'électricité utilisée par des infrastructures informatiques. En outre, un serveur de centre de données peut produire plus de 10 kg de CO_2 par jour.

Plusieurs enquêtes scientifiques ont révélé que ces taux sont en augmentation. Des clouds et des centres de données sont conçus pour prendre en charge le trafic maximal attendu. Toutefois, la charge globale moyenne représente environ 60% de la charge maximale.

De ce fait, un nombre important de serveurs ne sont pas en charge et continuent de consommer environ 65% de l'énergie maximale. Ainsi, des besoins d'économie d'énergie apparaissent pour envisager des stratégies de gestion de l'énergie. Plusieurs études montrent qu'une grande partie de l'énergie consommée dans le centre de données est principalement due à l'électricité utilisée pour faire fonctionner les serveurs et les refroidir (70% du coût total du centre de données).

Ainsi, le principal facteur de cette consommation d'énergie est lié au nombre de serveurs opérationnels. De nombreux efforts ont été consacrés à l'amélioration des serveurs et à leur refroidissement. Des travaux ont été effectués pour construire de meilleurs composants et des processeurs à faible consommation d'énergie, des réseaux plus efficaces en terme énergétique, des systèmes de refroidissement plus efficaces, et des noyaux système optimisés.

Par conséquent, pour garantir à la fois une bonne performance des services offerts par ces centres de données et une consommation d'énergie raisonnable, une analyse détaillée du comportement de ces systèmes est essentielle pour la conception d'algorithmes d'optimisation efficaces permettant de réduire la consommation d'énergie. Dans ce contexte, une approche complémentaire en matière d'économie d'énergie est à envisager. Une politique de gestion permettant de gérer le démarrage/l'arrêt des serveurs dans un centre de données afin d'assurer à la fois une meilleure qualité des services offerts par ces centres de données et une consommation d'énergie raisonnable. Deux exigences sont en conflit:

1. Maintenir une qualité de service (QoS) élevée.

2. Consommer moins d'énergie.

Pour répondre à la première exigence, nous devons allumer un grand nombre de serveurs, qui consomment plus d'énergie, réduisent la durée d'attente et réduisent

le taux de perte de clients, mais nécessitent une consommation énergétique élevée. Pour répondre à la deuxième exigence, nous devons activer un petit nombre de serveurs, ce qui entraîne une consommation d'énergie moindre, mais entraîne davantage d'attente et augmente le taux de pertes de clients. Ainsi, l'objectif est de concevoir de meilleurs algorithmes de gestion de l'énergie tenant compte ces deux contraintes afin de minimiser la durée d'attente, le taux de perte et la consommation d'énergie.

Dans cette thèse, notre travail principal consiste à concevoir des systèmes de gestion dynamique de l'énergie (DPM) basés sur des modèles stochastiques de files d'attente contrôlées. Le but est de rechercher les politiques de contrôle optimales afin de les appliquer sur des centres de données, ce qui devrait répondre aux demandes croissantes de réduction de la consommation d'énergie et de la pollution numérique tout en préservant la qualité de service (QoS).

Premièrement, on note qu'un travail important avait été fait pour analyser le problème de l'optimisation de l'énergie dans les data centers dans un contexte à temps continu (voir section 2.4 et les tables 2.8, 2.9).

Notre travail est entièrement consacré au contexte à temps discret. Notons que les contextes temporels continus offrent un cadre mathématique permettant de trouver les politiques optimales et étudier leurs propriétés structurelles en utilisant des méthodes analytiques.

Dans tout notre travail, le processus des arrivées et la capacité de traitement des requêtes par les serveurs sont modélisés par des distributions obtenues à partir de traces réelles, de données empiriques ou de mesures du trafic entrant. Ainsi, nous avons principalement utilisé la trace ouverte de Google [Wil11, RWH11], pour laquelle nous avions suggéré, dans un premier temps, un algorithme basé sur l'utilisation du test statistique des points de réflexion pour trouver la période d'échantillonnage adéquate afin de garantir la propriété i.i.d. (une distribution repartie de manière indépendante et identique) des données échantillonnées. Autrement dit, les choix des valeurs des paramètres ont été choisis comme suit:

- l'unité de temps: comme on est dans un cadre discret, on doit fixer la durée de l'unité de temps, pour la trouver on a proposé une technique qui se base du le test statistique *turning points* pour trouver la meilleur durée de l'unité de temps à considérer dans l'échantillonnage, l'analyse et l'optimisation (voir la fin de la Section 1.3.3).
- taille de la queue: dans la majorité des analyses que nous avons effectuées, on a fait varier la taille de la queue b de 1 jusqu'à une valeur assez grande.
- nombre maximal de machines: de même dans la majorité des analyses on fait varier $\mathcal{M}$, le nombre maximal de machines, de 1 jusqu'à une valeur assez grande.

- l'horizon h: ceci représente la durée sur laquelle on observe le système, là aussi on a pris soit un jour soit une semaine soit un mois d'analyse, pourquoi? Car on ne dispose que d'un seul mois de données de la trace de Google [Wil11, RWH11], donc soit on fait une analyse sur un mois, soit on accumule les quatre semaines pour faire une analyse sur la semaine, soit on accumule les jours du mois pour faire une analyse sur un jour. On a aussi considéré une analyse sur la nuit et sur la journée.
- les coûts unitaires: dans la majorité des analyses on ne fixe pas les coûts unitaires mais plutôt, on fait varier les coûts unitaires dans des intervalles selon plusieurs contraintes, par exemple, on fait varier c_n, c_m, et c_l avec la contrainte $c_n > c_m > c_l$ ou $c_n < c_m < c_l$ ou d'autre contraintes du même style.
- arrivées et services: on déduit des distributions à partir de traces réelles.

Et pour chaque expérimentation on analyse et on interprète le comportement du système. Il faut noter qu'on a considéré deux types d'expérimentations:

- expérimentation pour analyser le comportement du système (évolution),
- expérimentation pour analyser la complexité algorithmique.

Pour le dernier cas on est obligé de fixer tous les paramètres du système pour pouvoir comparer plusieurs techniques, dans le but d'analyser la taille et la rapidité de l'algorithme et non pas pour étudier le comportement de système.

Ensuite, et avant de montrer comment résoudre le problème d'optimisation énergétique de manière optimale, nous avons expliqué comment modéliser un mécanisme spécifique pour résoudre le problème d'optimisation énergie/QoS par une solution sous-optimale. Nous introduisons donc l'utilisation de DTMC (chaîne de Markov à temps discret) pour modéliser et résoudre le problème de la réduction de la consommation d'énergie dans un centre de données tout en maintenant une bonne QoS basée sur une stratégie monotone classique prédéfinie, à savoir la stratégie des seuils. Ce type de politique appartient en général à la classe des politiques sous-optimales. Après cela, et dans la continuité de l'approche basée sur DTMC, nous présentons un système d'optimisation stochastique plus compliqué, alors que nous le résolvons encore par un mécanisme spécial de stratégie de seuils.

Ainsi, cette seconde approche montre comment les opérateurs d'histogramme peuvent être utilisés pour éviter de générer explicitement un nombre important de DTMC comme dans la première approche. Plus précisément, nous introduisons l'utilisation des opérateurs d'histogrammes pour modéliser et résoudre le problème d'économie d'énergie consommée dans un centre de données tout en préservant les performances de son service, sur la base du mécanisme de stratégie de seuils.

De plus, nous montrons comment utiliser l'algorithme de détection de stationnarité pour accélérer le calcul de la meilleure politique de seuils.

Il est bien connu que les politiques de seuils sont pratiques et faciles à mettre en œuvre mais elles ne sont pas nécessairement les meilleures. Ainsi, dans un deuxième temps, nous nous sommes concentrés sur la minimisation *optimale* de la consommation d'énergie dans les centres de données. Donc pour trouver la stratégie *optimale* nous avons formaliser le modèle dans un cadre temporel discret basé sur l'utilisation du concept de *processus de décision de Markov* afin de garantir une consommation d'énergie raisonnable avec une bonne performance du service offerte par le centre de données.

Pour trouver la politique optimale, nous formulons le problème d'optimisation à l'aide d'un MDP à temps discret. Nous nous intéressons d'abord à la modélisation de la gestion dynamique de l'énergie par un modèle stochastique pour un centre de données homogène, principalement pour étudier certaines propriétés structurelles de la stratégie optimale, telle que la monotonie. Deuxièmement, comme les centres de données présentent un niveau non négligeable d'hétérogénéité de serveurs en termes de consommation d'énergie et de taux de service, nous généralisons le modèle homogène à un modèle hétérogène. Enfin, comme le réveil (resp. l'arrêt) d'un serveur de centre de données n'est pas instantané et nécessite un peu plus de temps pour passer du mode veille au mode prêt à fonctionner, nous étendons le modèle dans le but d'inclure cette latence temporelle des serveurs. Comme précédemment, tout au long de cette optimisation exacte, les arrivées et les taux de service sont spécifiés avec des histogrammes pouvant être obtenus à partir de traces réelles, de données empiriques ou de mesures de trafic entrant.

Pour un modèle homogène, les résultats théoriques et expérimentaux montrent qu'en général[1], la politique optimale n'est pas monotone, ce qui nous mène aux conclusions suivantes:

- la politique optimale ne peut pas être exprimée selon une structure de seuils,
- la politique basée sur une structure à double seuil est en général sous-optimale.

L'optimisation par MDP conduit à la politique optimale qui économise une quantité importante d'énergie. Cependant, en comparaison avec les méthodes basées sur les seuils, MDP sous PRISM nécessite plus de temps de calcul et plus d'espace mémoire pour calculer cette stratégie optimale. Nous observons que l'approche basée sur les seuils calcule la stratégie d'optimisation plus rapidement et utilise moins d'espace mémoire, mais elle ne renvoie qu'une stratégie sous-optimale.

[1] Pour des valeurs particulières des paramètres du système la politique peut être monotone.

Pour le modèle hétérogène, nous avons prouvé que ni la monotonie ni l'isotonie ne sont vérifiées par la politique hétérogène optimale. Par conséquent, la politique optimale ne peut pas être conçue comme une simple structure à double seuil. De plus, les résultats montrent que la taille du modèle hétérogène est supérieure à celle du modèle homogène. Cependant, à partir de résultats expérimentaux, il semble que l'hétérogénéité croissante des serveurs entraîne davantage d'économies d'énergie.

Pour les modèles avec latence, les résultats montrent que la taille du modèle est énorme, plus précisément la taille est une exponentielle de la période de latence. Cependant, à partir des résultats expérimentaux, lorsque la latence est plus importante, il semble que l'augmentation de la capacité de la queue b entraîne potentiellement une plus grande économie de l'énergie.

Afin de spécifier, résoudre, puis analyser la consommation énergétique de nos centres de données, nous utilisons un logiciel de vérification formel efficace appelé PRISM, un logiciel probabiliste utilisable pour la spécification de modèles MDP. Nous avons mis en place un outil nous permettant de générer automatiquement les spécifications PRISM de nos différents modèles de gestion dynamique de l'énergie. Avec cet outil, nous pouvons effectuer de nombreux test en modifiant facilement les paramètres sans réécrire manuellement des milliers de lignes de code PRISM.

Nous avons montré que la taille des modèles de MDP ci-dessus est très grande et conduit au problème de l'explosion d'espace d'états. De plus le calcul de la stratégie optimale nécessite un temps de calcul important. Nous avons observé que l'utilisation des modèles de MDP est souvent difficile, voire impossible, à réaliser pour les grands centres de données. Surtout si nous voulons prendre en compte des aspects réels tels que l'hétérogénéité ou la latence des serveurs. Le problème provient principalement de la taille exponentielle de la structure du MDP, puis du temps nécessaire pour la parcourir pour trouver la stratégie optimale. La complexité est donc importante tant dans l'espace que dans le temps. En effet, nous suggérons ce que nous appelons l'algorithme greedy-window qui permet de trouver une stratégie sous-optimale meilleure que celle produite lorsqu'on envisage un mécanisme spécial comme les approches à seuil. Et plus important encore, contrairement à l'approche MDP, cet algorithme glouton n'exige pas la construction complète de la structure, y compris toutes les stratégies possibles. Ainsi, notre algorithme donne une stratégie très proche de la stratégie optimale avec des complexités spatio-temporelles très faibles.

Cette dernière solution consiste à fournir un algorithme efficace qui calcule une solution assez proche à la stratégie optimale, en utilisant un espace mémoire et un temps de calcul inférieurs à ceux des solutions basées sur MDP ou les politiques de seuils. Cela rend cette solution pratique et permet son implémentation dans un contexte à temps réel. Cette solution a été aussi codée sous forme d'un outil

permettant de calculer cette stratégie pour pouvoir la comparer avec les résultats des autres méthodes.

Chapter 1

Introduction

1.1 Context of work: Background and Generalities

1.1.1 Background (Cloud in real life)

TODAY almost every feature and all aspects of our modern life are changing to become partially or sometimes completely digital, leading to an explosion of data and the global growth of the data centers. For example, Intel [Nel16] confirms that one autonomous car generates more than 4000 GB of data every day, and Cisco [Cis15] estimates that globally a Virtual Reality traffic will increase 61-fold until 2020 and data center traffic is to grow 300% by the same year. Notice that $30-40\%$ of the total cost of a data center is due to energy use. In fact, the workload will multiply in the next years and the data centers demand in terms of energy will continue to increase exponentially. Thus, in order to keep the energy demand low and flat it will be crucial to improve both the hardware and the software of electrical/electronic components, computers, digital infrastructures, cooling systems, lighting systems, and making the processing of requests in data centers more efficient. However data centers need to act very quickly as clients are demanding more and more flexible and reliable services. In fact, an important work is currently devoted by both scientists and industrialists to give data centers the opportunity to be manageable efficiently and safely in terms of consuming energetic resources. Increasingly more people and more things are becoming connected from medical and healthcare applications, transportation, building and home automation, manufacturing, agriculture, to military applications. Thus, as reported by the International Data Corporation [ZW+14], the digital universe is doubling every two years, and this large number of devices that will become online during the next few years will generate and use more data which need more clouds

services and more data centers. Scholars as in [BBA10, CM12, Sor15] confirm that it will be a daily challenge to reduce the energy consumption and it should be the responsibility we have with our environment. In fact, we will have to become smarter, more sensible, and more efficient to deal with this challenge and give computer science the opportunity to participate actively in preserving our planet when using efficiently the energy and allowing all people to appreciate serenely all the advantages of our digital society. This requires a degree of sophistication and a degree of control that is in some level inherent in the solution suggested in the work presented in this manuscript. Thus, I hope the management systems described and analyzed in this thesis will help us in that direction.

1.1.2 Data center

What is a data center? When a small company starts its activity, it needs to store all its paper data like financial information and customer details in a digital form in order to keep it safe from theft or catastrophes as fire and also to make this data more easily accessible and shared. The company also needs an Internet connection to host their website and their emails. In fact all this digital data is stored in a computer called server. This server is basically a specialized computer of the size of one or two schoolbags connected to the local network of the company and also to the Internet. It contains several hard disks that store data in a replicated way to be able to back up the data of the company if for any reason or accidentally some data is deleted. Thus the data can be shared between the different desktop computers of the company through the local network and information can be sent outside to people visiting the website of the company through Internet. Now, as the company grows up, the website becomes more popular and bigger demand for the company products grows and consequently bigger needs to store more data and ensure a good quality of the services emerge. Inevitably, the company must add more and more servers and eventually end up by building an entirely separate server room to house all this servers.

Additionally, servers need to be cooled (see Figure 1.1) and maintained by an information technology professionals to ensure the reliability and the security of the servers. Also to ensure the continuity of the services in the case of the occurrence of any failure due to electricity, servers must be equipped with their own backup uninterruptible power supplies usually in the form of large batteries or a powerful generator. In fact all these constraints can become a serious financial and logistical challenge for the company. Indeed, rather than trying to solve these problems themselves, nowadays, many companies, foundation, and organizations choose to move all their data and digital services out of their own buildings into a specialized facility called a data center. Which is a secure computer friendly

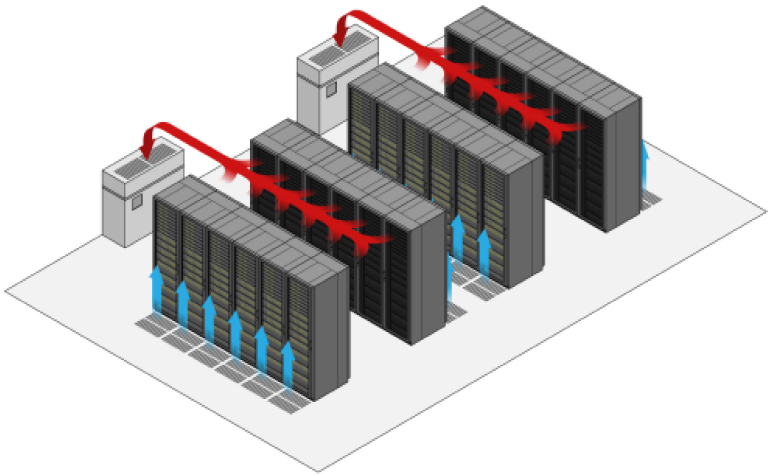


Figure 1.1: Cooling servers.

environment that is maintained continually by IT specialists, electrical engineers and experts to keep it at the right running conditions (especially temperature) to ensure the services supported inside the servers. A data center can be seen as a computing/storage collocation because this facility is shared by many companies at once. Indeed, companies rent as many racks (set of servers) as they need according to their requirements in terms of processing power, space of storage, and bandwidth (see Figure 1.2). Now, a lot of servers creates a lot of heat. So servers need to be kept cool otherwise they can fail catastrophically. In fact, to keep the servers at a constant computer friendly temperature, traditionally an air-conditioning units similar to the ones used in cars and homes are used to supply the cold air needed to keep the server rooms cold. However, providing this level of cooling using traditional air-conditioning units can be very expensive.

Finally we can say that a **data center** is a platform which houses a group of computer machines (called servers). These servers are connected through a network and physically supported by needed infrastructure, such as energy/electricity, cooling, ventilation and air conditioning system, to keep them operational to provide a service. A data center can be used to store databases and doing computations, and it can be accessed from outside through Internet (see Figure 1.3). That means that often much electricity is consumed just for keeping the facility cool [BGDG+10, Bal11, LZ12]. In fact for this reason the global data center industry produces approximately the same amount of pollution (in terms of carbon dioxide production) as the airline industry. In fact current estimates suggest that about 1.3% of the energy consumed by the entire world each year is spent just for running and cooling computers server facilities. This proportion is about 5% in Europe [Koo11]. Recently the introduction of newer eco-friendly cooling techniques into

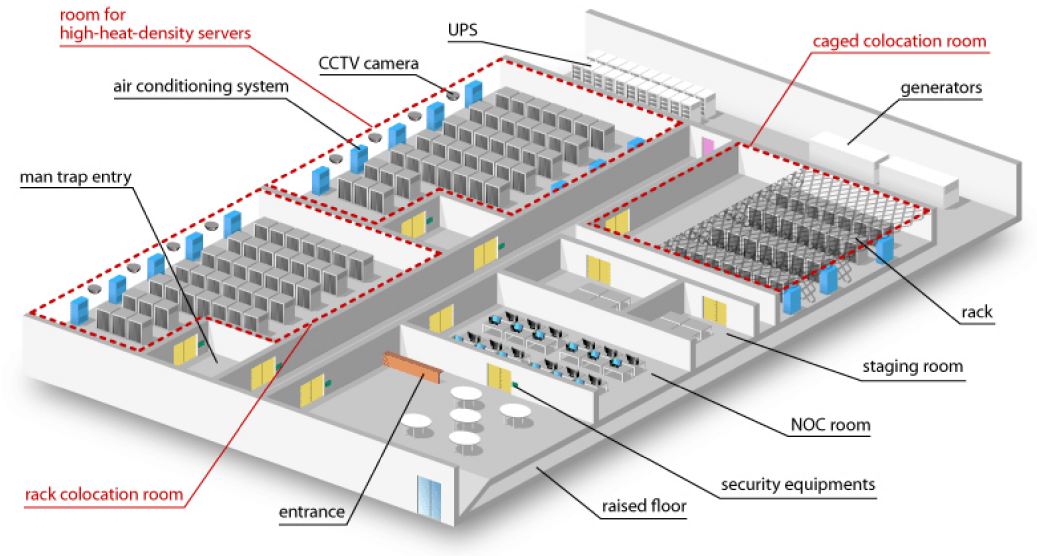


Figure 1.2: Typical overview of a data center.

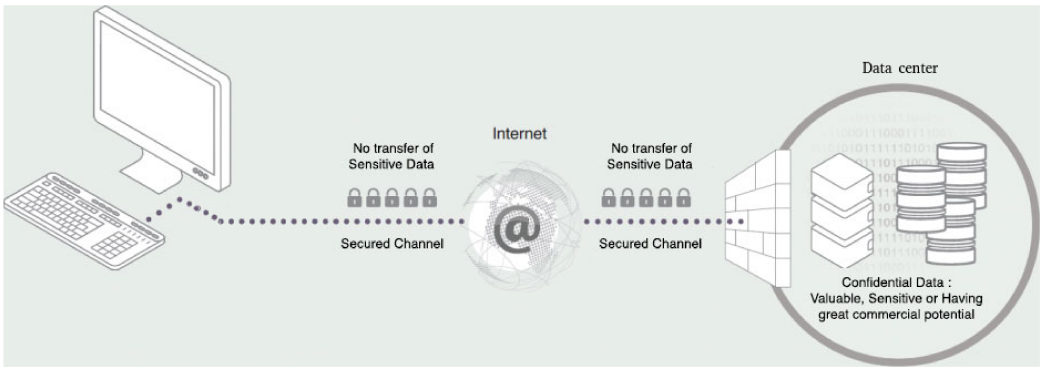


Figure 1.3: Data center access.

data centers reduce considerably the amount of electricity used and carbon dioxide ejected.

1.2 Generalities and problems

1.2.1 Queuing model theory

Let us now introduce queuing theory. It is well known that various real-life systems, like supermarket payment queues, manufactures of production, exchanging and sharing information systems, networks and communication infrastructures as well as data centers can be represented by means of queuing models. Queuing system takes it origin since the early 1900s in the design of automatic telephone exchange systems that were analyzed by Agner Krarup Erlang in order to determine how many lines had to be supplied in order to guarantee a certain level of service [Erl17]. Analogous problems emerge in numerous other cases, for instance looking at the impact of service rate on waiting time in payment queues, in order to satisfy the client's demands for quality of service (QoS). In practice, the problem that queuing theory faces principally is how to find a balance between improvement of QoS and economic related costs. In other terms, how to find the best trade-off between a gain in QoS, supplying more resources, and additional economic loss. Notice that queuing theory has succeeded to obtain many analytical and numerical results by studying formally and mathematically models including the probabilistic aspect of real systems. Those results cover various important quantitative features which characterize the behavior of systems such as queue length, waiting time distributions, loss probabilities, and throughput, etc. Classical queuing models do not include controllers that allow to inspect different strategies, when taking adequate decisions based on the state of the system. Such a controller may considerably improve the performance of the system by for example decreasing waiting time, reducing queue length, or increasing the throughput. However, the absence of a such controller may make the system going through periods of high load that induce long waiting time followed by periods during which the servers remain idle and still consuming resources. In fact, the controlling mechanism is performed by suitable actions that can be described in mathematical and formal terms and then subjected to determine the optimal control policy. Thus, by adding such controlling aspects into queuing models, the field of queuing theory was extended as a branch of optimization theory in which the subject of this thesis fits. The theoretical foundations of controlled queuing systems are based on the concept of Markov decision processes [Ryk66, MO70, Ros70, Put94, KR95, LL96, FS12].

1.2.2 Policy and strategy

In the early 1950s, Bellman was one of the first who worked on decision processes and developed the so called dynamic programming as a computational method for analyzing sequential decision processes within a finite planning period called horizon [Bel57]. Few years later Howard generalized the work of Bellman and moves from Markov chain theory and dynamic programming to a policy-iteration algorithm that provides the solution of sequential probabilistic decision processes with infinite horizon [How60]. A typical sequential controlling decision model can be viewed as the following: at a specified point of time referring to current unit of time (current slot), the controller (or the decision maker) observes the system and based on its state, the controller chooses to perform a decision as a particular action. The chosen action produces an immediate cost, and moves the system to a new state at the consecutive unit of time (see Figure 1.4).

At the next unit of time (slot), the manager faces a similar situation. Different states, in general, leads to different sets of possible actions. As the process evolves in time, the manager accumulates a sequence of costs (a sequence of negative rewards).

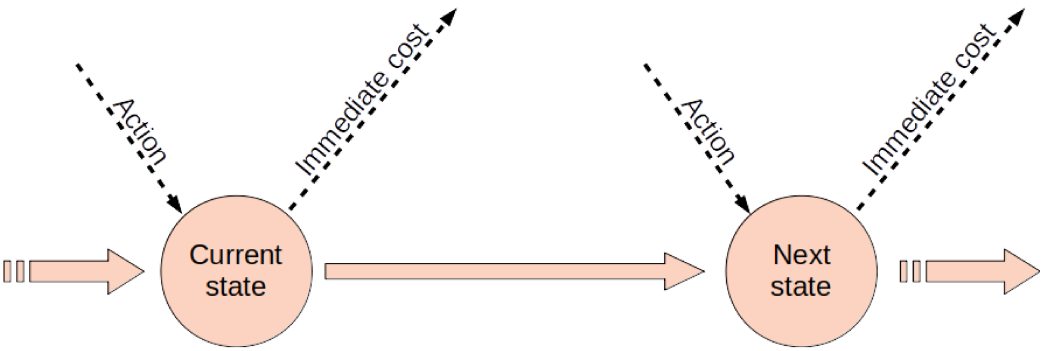


Figure 1.4: Controlling by action mechanism.

Now it is clear that such a sequential decision model is composed of the following elements that are assumed to be known to the manager at any slot:

1. discrete time: a sequence of consecutive slots,
2. the set of system states,
3. the set of possible actions,
4. the function of immediate costs,
5. the transition probabilities matrix between states.

In fact, we define policy or interchangeably strategy as the following. A **policy or a strategy** is an instruction telling the manager, depending on the state of the system and the current time, which action to choose and perform.

Thus the control problem consists in the task to choose an optimal strategy that minimizes some function over the cost sequence. There are many functions that may be considered suitable for classifying optimum system behavior.

In the present thesis, as we are more interested in the transient regime, we confine our analysis to the optimality criterion of expected total cost function accumulated over a period of time.

1.2.3 Optimality

In several applications the best approach to design a strategy can be done by simply comparing the results of different alternatives. It is clear that this approach is feasible only if the number of alternatives is relatively limited. Otherwise, optimization methods are required to find the best/good strategy. However, for complex and more realistic cases, where the number of different possible strategies is very large, it is clear that simple enumeration procedures for finding an optimal policy is inadequate. A Markov decision model is a Markov model which can use additionally the dynamic programming techniques for the computation of the optimal solution.

1.2.4 Metrics

It is very important for data center providers to measure the performance/QoS of their services. For a cloud that hosts data, performance is measured in terms of availability of data over a period of time. For a server farm hosting database or websites, performance is often measured in terms of response time. For a grid computation, performance is measured in terms of completion time or number of completed jobs in a period of time.

Moreover, it is well known that servers need an additional amount of energy to switch-on (resp. switch-off). It is also known that the switching-on (resp. switching-off) is not instantaneous and a server needs some units of time to transit from a sleeping/stopping mode to a ready-working mode. This period of time during which the server switches from a mode to another is called *latency*.

In the following we define some popular performance/energy metrics of data centers.

In the context of data centers, energy is the quantitative property that must be transferred to a server in order to be able to process jobs when providing the

requested service on those jobs. And power is the amount of energy transferred per unit of time: $power = \frac{energy}{time}$.

For a given period of time, **waiting jobs** is the average number of jobs that wait in the data center queue before being served.

We call **arrival job** a request arriving to the data center asking the service provided by the data center. If the job is received by the data center, it will be called **waiting job** until a server begins processing it. If not received by the data center (for any reason) it will be called **lost job** or **rejected job**.

Waiting time for a request is defined as the time from when the job arrives at the data center to the time when its service begins.

Response time for a request is defined as the time from when the job arrives at the data center to the time when its service is completed.

Time latency is defined as the period of time needed to switch on/off a server. It is called sometimes *setup* for the switching-on case.

Working power is the amount of energy, per unit of time, needed by a busy server to process requested jobs.

Idle power is the amount of energy, per unit of time, needed by a non busy server to keep itself on.

Energetic latency (or switching energy) is defined as the amount of additional energy needed for a server to switch on/off. In other words, it is the total energy consumed by the server during the entire latency time.

1.3 Queue model

Here we deal with discrete time models. Let *DC* be a data center composed of $\mathcal{M}$ non identical servers working under the FIFO[1] discipline. *DC* receives jobs requesting the proposed service. The number of jobs served by one server in one slot is not assumed to be constant (see Section 1.3.4). However it is modeled by a discrete distribution obtained from real data center traces. In a similar way, arrival jobs are also modeled by discrete distributions. Thus, both arrival and service processes are specified by *histograms*. The queuing model is a batch arrival queue with finite capacity buffer b (buffer size). Notice that the model considers heterogeneous servers, which means, non identical servers. This point comes from the analysis of the Google trace servers [Wil11, RWH11] which shows that data centers present a non negligible level of server heterogeneity. Several levels of heterogeneity were observed for instance: heterogeneity in CPU-speed and

[1] First In First Out.

heterogeneity in memory-size. Those criteria are good indicators for QoS and also energy consumption and taking them into account is very important for our energy optimization problem. Additionally, in order to include more realistic aspects of the data center in the context of energy saving, the model considers the latency of the servers (see Figure 1.5).

1.3.1 Evolution equations

In a queue, at discrete time, we need to accurately describe the order of events of inputs and outputs. Let us define the following variables:

- $a(t)$ number of arrival jobs during slot t,
- $n(t)$ number of waiting jobs during slot t,
- $l(t)$ number of rejected jobs during slot t,
- $m(t)$ number of operational servers during slot t.
- d number of jobs that can be severed by one server during one slot.

We begin by serving the waiting jobs of the buffer, next we fill the free operational servers by the new jobs, then we fill the buffer. This means that the new jobs are spread over all the free places of the system: in the free operational servers and in the buffer. Notice that this order of events is chosen in this specific way to ensure the FIFO discipline. In this case the number of jobs waiting (resp. rejected jobs) at instant $t+1$ can be deduced recursively as follows:

$$n(t+1) = \min(b, (n(t) + a(t) - d \times m(t))^+)$$
$$l(t+1) = (n(t) + a(t) - d \times m(t) - b)^+.$$

1.3.2 Histogram operators

As mentioned before, in the literature, modeling the arrival jobs consists in obtaining a continuous time distribution by fitting the experimental data. Another approach consists in modeling the arrival jobs by a DTMC[2] as in [BBPDM99]. In this work we suggest to use directly the discrete finite distributions that may be obtained from sampling real traces, empirical data, or incoming traffic measurements.

In fact we model the random variables of our systems (arrival jobs, waiting jobs, lost jobs, etc) by finite discrete distributions represented by finite structures called histograms.

[2]Discrete Time Markov Chain.

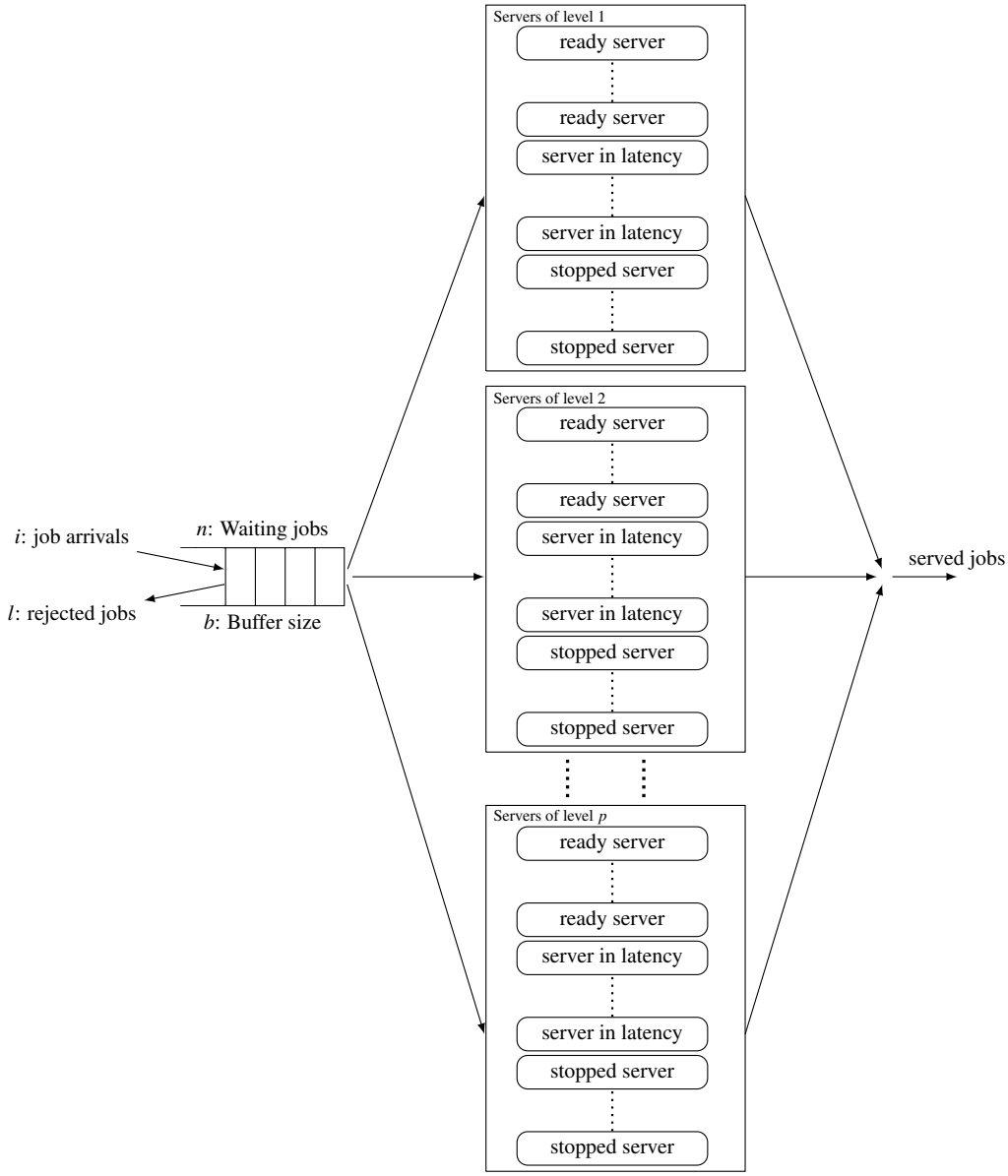


Figure 1.5: Illustration of the generic queuing model: each server belongs to a group of servers with specific energy consumption level. Each server is either ready, stopped, or in latency period.

Definition 1.1 (histogram)**.** *Let X be a discrete random variables taking values in $\mathbb{N}$. The couple $\mathcal{H}_X = (S_X, P_X)$ denotes the histogram of X where $P_X : \mathbb{N} \to [0,1]$ is the probability mass function of X and $S_X = \{i \in \mathbb{N} : P_X(i) > 0\}$ is the support of P_X. Notice that $\sum_{i \in S_X} P_X(i) = 1$.*

Example 1.1. Let A be a random variable for the arrival jobs of a system. Let $\mathcal{H}_A$ be its histogram. We have:

$$\mathcal{H}_A = \begin{cases} S_A & = & [\ 2 & , & 4 & , & 5 & , & 7\ ] \\ P_A & = & [\ 0.2 & , & 0.3 & , & 0.4 & , & 0.1\ ]. \end{cases}$$

Thus, we can say for example that we have a probability of 0.2 to receive 2 jobs during a slot (one time unit) and a probability of 0.1 for receiving 7 jobs. Implicitly, we note that the probability of receiving 0, 1, 3, 8 jobs or more is zero.

In order to do calculus on histograms, the following operators defined over histograms are needed to compute the evolution of a stochastic system:

1. Δ_v is the Dirac histogram with $v \in \mathbb{N}$, where the probability mass function is defined as:
$$\forall i \in \mathbb{N} : \ P(i) = \begin{cases} 1 & \text{if } i = v \text{ and} \\ 0 & \text{otherwise.} \end{cases} \tag{1.1}$$

2. Given a random variable X and a natural number $v \in \mathbb{N}$, the histogram of the variable $Y = X - v$ noted by $\mathcal{H}_Y = \mathcal{H}_X - v$ corresponds to the histogram of $\mathcal{H}_X$ translated by constant v:
$$P_Y(i) = P_X(i - v) \text{ if } i > v \text{ and } P_Y(0) = \sum_{i \leq v} P_X(i). \tag{1.2}$$

3. $\mathcal{H}_Y = MIN_b(\mathcal{H}_X)$ is the distribution of variable X bounded above by constant b. This operation corresponds to a minimum on the underlying random variable. It is defined by:
$$P_Y(i) = P_X(i) \text{ if } i < b \text{ and } P_Y(b) = \sum_{i \geq b} P_X(i). \tag{1.3}$$

4. Given two random variables X and Y, the histogram of the variable $Z = X + Y$ noted by $\mathcal{H}_Z = \mathcal{H}_X \oplus \mathcal{H}_Y$ corresponds to the convolution of histograms $\mathcal{H}_X$ and $\mathcal{H}_Y$:
$$\forall i \in \mathbb{N} : \ P_Z(i) = \sum_{j=0}^{i} P_X(j) \times P_Y(i - j). \tag{1.4}$$

Example 1.2. Let $\mathcal{H}_X$ be an histogram such that:

$$\mathcal{H}_X = \left\{ \begin{array}{lllllll} S_X & [\ 1 & ,\ 2 & ,\ 3 & ,\ 5 & ,\ 9 & ,\ 11\ ] \\ P_X & [\ 0.1 & ,\ 0.2 & ,\ 0.3 & ,\ 0.1 & ,\ 0.15 & ,\ 0.15\ ]. \end{array} \right.$$

We have:

$$\begin{aligned} MIN_4(\mathcal{H}_X) &= \left\{ \begin{array}{lllll} S_{MIN_4(\mathcal{H}_X)} & [\ 1 & ,\ 2 & ,\ 3 & ,\ 4 \qquad\qquad\qquad\ ] \\ P_{MIN_4(\mathcal{H}_X)} & [\ 0.1 & ,\ 0.2 & ,\ 0.3 & ,\ 0.1+0.15+0.15\ ] \end{array} \right. \\ &= \left\{ \begin{array}{lllll} S_{MIN_4(\mathcal{H}_X)} & [\ 1 & ,\ 2 & ,\ 3 & ,\ 4\ ] \\ P_{MIN_4(\mathcal{H}_X)} & [\ 0.1 & ,\ 0.2 & ,\ 0.3 & ,\ 0.7\ ]. \end{array} \right. \end{aligned}$$

$$\begin{aligned} MIN_0(\mathcal{H}_X) &= \left\{ \begin{array}{ll} S_{MIN_0(\mathcal{H}_X)} & [\ 0 \qquad\qquad\qquad\qquad\qquad\qquad\ ] \\ P_{MIN_0(\mathcal{H}_X)} & [\ 0.1+0.2+0.3+0.1+0.15+0.15\ ] \end{array} \right. \\ &= \left\{ \begin{array}{ll} S_{MIN_0(\mathcal{H}_X)} & [\ 0\ ] \\ P_{MIN_0(\mathcal{H}_X)} & [\ 1\ ] \end{array} \right. \\ &= \Delta_0. \end{aligned}$$

$$\begin{aligned} MIN_{18}(\mathcal{H}_X) &= \left\{ \begin{array}{lllllll} S_{MIN_{18}(\mathcal{H}_X)} & [\ 1, & 2, & 3, & 5, & 9, & 11\ ] \\ S_{MIN_{18}(\mathcal{H}_X)} & [\ 0.1, & 0.2, & 0.3, & 0.1, & 0.15, & 0.15\ ] \end{array} \right. \\ &= \mathcal{H}_X. \end{aligned}$$

We have also:

$$\begin{aligned} \mathcal{H}_X - 4 &= \left\{ \begin{array}{lllll} S_{\mathcal{H}_X-4} & [\ 0, & 1, & 5, & 7\ ] \\ P_{\mathcal{H}_X-4} & [\ 0.1+0.2+0.3, & 0.1, & 0.15, & 0.15\ ] \end{array} \right. \\ &= \left\{ \begin{array}{lllll} S_{\mathcal{H}_X-4} & [\ 0, & 1, & 5, & 7\ ] \\ P_{\mathcal{H}_X-4} & [\ 0.6, & 0.1, & 0.15, & 0.15\ ]. \end{array} \right. \end{aligned}$$

Example 1.3. Let $\mathcal{H}_X$ and $\mathcal{H}_Y$ be two histograms such that:

$$\mathcal{H}_X = \left\{ \begin{array}{llll} S_X & [\ 2 & ,\ 3\ ] \\ P_X & [\ 0.4 & ,\ 0.6\ ] \end{array} \right. \text{ and } \mathcal{H}_Y = \left\{ \begin{array}{llll} P_Y & [\ 10 & ,\ 20\ ] \\ S_Y & [\ 0.1 & ,\ 0.9\ ]. \end{array} \right.$$

We have:

$$\begin{aligned} \mathcal{H}_Z &= \mathcal{H}_X \oplus \mathcal{H}_Y \\ &= \left\{ \begin{array}{lllll} S_Z & [\ 2+10 & ,\ 3+10 & ,\ 2+20 & ,\ 3+20\ ] \\ P_Z & [\ 0.4\times 0.1 & ,\ 0.6\times 0.1 & ,\ 0.4\times 0.9 & ,\ 0.6\times 0.9\ ] \end{array} \right. \\ &= \left\{ \begin{array}{lllll} S_Z & [\ 12 & ,\ 13 & ,\ 22 & ,\ 23\ ] \\ P_Z & [\ 0.04 & ,\ 0.06 & ,\ 0.36 & ,\ 0.54\ ]. \end{array} \right. \end{aligned}$$

Remark 1.1. Notice that the computation of a convolution by the direct method described by Equation (1.4) requires $O(|S_X| \times |S_Y|)$ operations which can be significantly reduced with a fast algorithms typically by the use of fast convolution algorithms [PD13, Jan00], to reduce the cost of the convolution to $O(n \log n)$ complexity where n is the extent of the support of the histograms: $n = \max(S_X) + \max(S_Y)$. Figure 1.6 shows an experimental comparison between the quadratic direct method and the FFT-based method by using the `scipy` Python package that implements both methods.

Remark 1.2. In all of our work the convolution operator is used principally to compute a convolution between $\mathcal{H}_A$ a histogram of arrival jobs and $\mathcal{H}_N$ histogram of waiting jobs in the buffer.

Remark 1.3. The number of waiting jobs is bounded above by b, in fact both the support of $\mathcal{H}_N$ and its extent are bounded above by b, thus in terms of complexity $|S_N|$ and $\max(S_N)$ are in $O(b)$.

Remark 1.4 (Smoothness)**.** The histogram of arrival jobs $\mathcal{H}_A$ is obtained by sampling real traffic traces as we done with the open Google trace [Wil11, RWH11] and we observed that the obtained distribution is smooth ($|S_A| \sim \max(S_A)$). Smoothness means that both the support of $\mathcal{H}_A$ and its extent are of the same order, thus in terms of complexity $|S_A|$ and $\max(S_A)$ have the same order (see for example the distributions used in Chapter 6).

1.3.3 Description of Arrivals

As said before, in this work we consider discrete-time queue models. Our queuing model is a batch arrival queue with a finite capacity buffer b. To model job arrivals, we use real traffic traces based on the open *cluster-data-2011-2* trace [Wil11, RWH11]. As in [BDF$^+$16] and [Bay18], we focus on the part that contains the job events corresponding to the requests destined to a specific Google data center for the whole month of May 2011. The job events are organized as a table of eight attributes. Column *timestamps* refers to the arrival job instant expressed in microseconds. Thus based on this trace, the number of jobs arriving to the data center during a slot is modeled by a histogram $\mathcal{H}_A$ where $P_A(i)$ gives the probability to have i arrival jobs per a slot.

Example 1.4. Assume that, per slot, we have a probability of 0.14 to receive 3 arrival jobs, 0.19 to receive 7 arrival jobs, and 0.67 to receive 11 arrival jobs. In this case, arrivals are modeled by histogram $\mathcal{H}_A = (S_A, P_A)$ where $S_A = \{3, 7, 11\}$, $P_A(3) = 0.17$, $P_A(7) = 0.19$, and $P_A(11) = 0.67$.

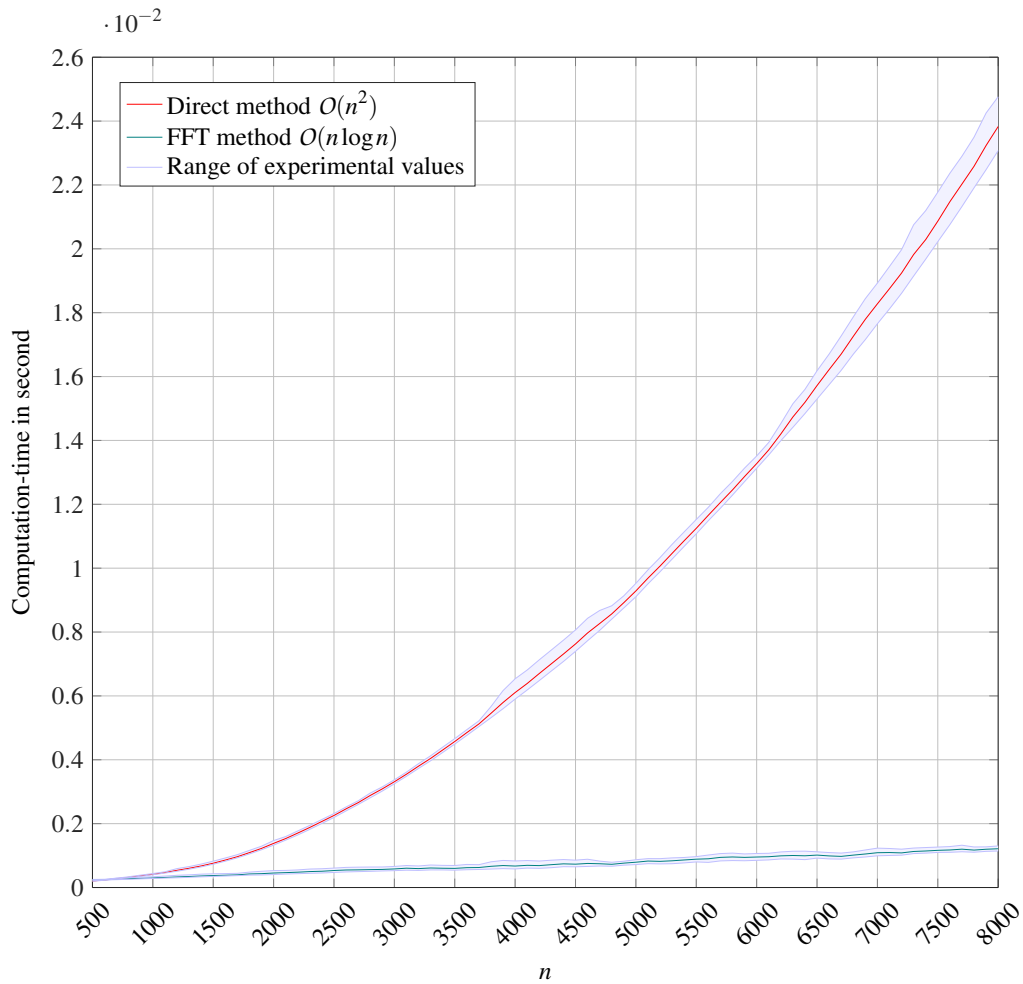


Figure 1.6: Comparison between the direct method and the FFT-based method for computing the convolution of two histograms $\mathcal{H}_X \oplus \mathcal{H}_Y$. The histograms were generated randomly where the extent of each histogram is close to the support of the histogram: $\max(S_X) = n$, $\max(S_Y) = n$, $|S_X| \sim n$, and $|S_Y| \sim n$.

Figure 1.7 shows the time series analysis on the Google trace arrival jobs [Wil11, RWH11]. The left graph shows the evolution of the number of arrival jobs during a month. The blue curve is the real traffic (real number of jobs arrived every day). This blue curve presents some regularity. The traffic is weak at the beginning of the week, then it begins to grow before stabilizing relatively during the middle of the week and after that at end of the week the traffic begins to be weaken again. In fact we used a periodic interpolation of the real traffic which is presented by red curve. By comparing the two curves, we observe two aberrant points: the first one is a heavy load on Google's servers on May 18, 2011 (perhaps for DSK case judgment). The second one is a significant drop in load on May 26, 2011, which may be related to meteorological conditions (a tornado) on the US southeast region. The two maps on the right of Figure 1.7 show tornado impact and location of Google's computing centers.

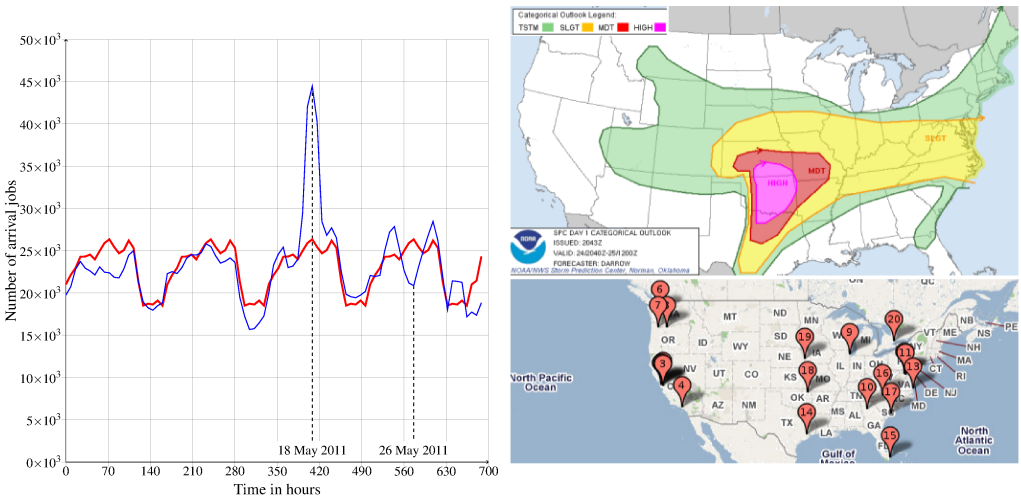


Figure 1.7: Left: Evolution of the number of arrival jobs during a months for the Google trace (blue curve is the real traffic, the red curve is a periodic interpolation of the real traffic). Top right: Tornado outbreak sequence of May 21–26, 2011 – Day 4 of outbreak which included a 45% tornado area, above minimum high risk threshold. Bottom right: location of Google's data center.

We deal with time-homogeneous discrete time Markov chains. Actually it is important to assume the i.i.d.-ness of the arrivals. However to avoid to just keep that property as an assumption, we suggested to construct an arrival model which is i.i.d. In fact we do the following: the traffic trace must be sampled with a sampling period for which the arrival jobs can be considered as i.i.d. So, we sample, literally, the data with several sampling periods between a small sampling period (one second) and a big one (five minutes), then we applied the turning point test [Ken73]

for each sampling periods in order to test the i.i.d.-ness of the sampled data. We found that sampling period between 115 and 155 second has a $p-value$ bigger than 0.05 thus at confidence level 0.95 we accept the null hypothesis, namely, the sampled data is i.i.d. If i.i.d.-ness test is accepted for several values of sampling period, as we deal with several kinds of arrivals (for instance arrivals for daytime, nighttime, and for each day of the week), we will choose the sampling period for which the turning point is accepted for all arrival kinds. In fact, we consider frames of 136 second to sample the trace and construct empirical distributions for daytime, nighttime, and for each day of the week. The support of the histogram obtained is formed of around 200 bins, and the average of arrival jobs is around 46 jobs per slot (see Figure 1.8).

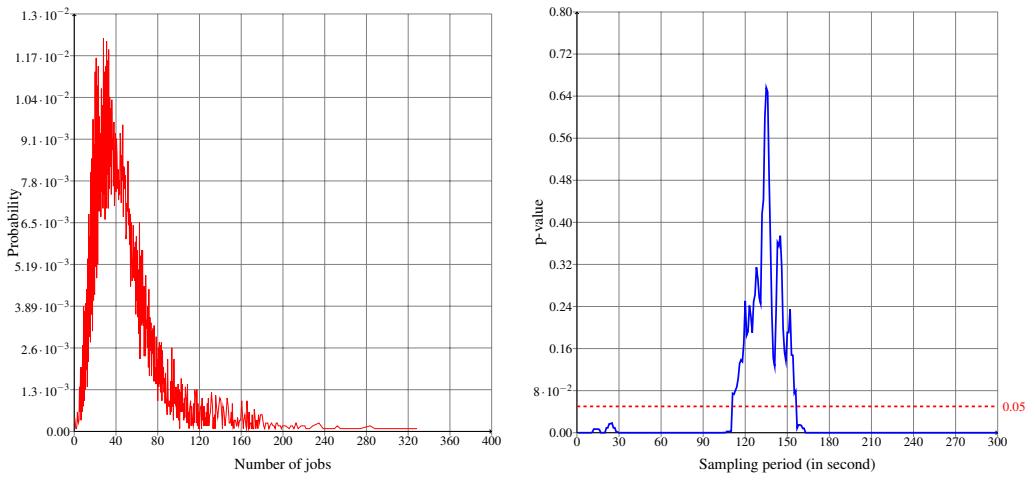


Figure 1.8: Left: Google trace arrival jobs distribution. Right: i.i.d.-ness test.

Notice that in this work we modeled arrival jobs by an i.i.d. distribution over all the period of analysis in Section 3.1, in all Chapter 4, and in Section 5.3.2. We considered a pairwise i.i.d. distributions for Section 3.2 and Section 5.3.3.

1.3.4 Service rate Description

As we know now the length of a slot (see previous section), in this work (especially in Chapter 4) we define the service rate as the number of jobs served by one server during one slot. As we avoid to consider that the service rate is constant, we use also real traffic traces (like the open *cluster-data-2011-2* trace [Wil11, RWH11]) to model the service rate. In fact, We focus on the table that refers to the time needed to process each accepted job by each server of the given collection of servers of Google's trace. Thus, the number of jobs served by one server during one slot is

modeled by a histogram $\mathcal{H}_D$ where $P_D(i)$ gives the probability to process i jobs by one server during one slot.

1.3.5 Energy and QoS metrics

According to research published [MDD10], a single server consumes around something between 238 and 376 Watts. Rajesh et al. [RDSJ08] estimate the cost of one kWh of energy to 0.0897$. These values may vary depending on where the data center is located and how electricity is generated. Using that baseline, one server costs around 300$ per year to run.

Every job may generate a profit, and the average profit per job can be computed as a ratio of the total profit over the number of served jobs. For instance, 10^6 requests (page views) may bring 1000$ of revenue. Thus, it can be said that each job brings $10^{-3}$$ on average. Work in [DM10] suggests that each successfully processed job[3] generates a profit around $6.2 \times 10^{-6}$$. In this case, a lost job costs $6.2 \times 10^{-6}$$. Table 1.1 summarizes the costs considered over one slot.

Table 1.1: Costs considered over one slot.

Cost	**Meaning**
c_N	waiting cost for one job over one slot
c_L	rejection cost for one job over one slot
c_M	energy cost for running one operational server
c_{busy}	energy cost for running one busy operational server
c_{idle}	energy cost for running one not busy server (idle)
c_{off}	energy cost for stopped/switched-off server
$c_{M_g} \in \mathbb{R}^+$	energy cost for running one operational server of group g
$c_{On_g} \in \mathbb{R}^+$	energy cost needed to switch-on of level g
$c_{M_{high}} \in \mathbb{R}^+$	energy cost for running one operational high level server
$c_{M_{low}} \in \mathbb{R}^+$	energy cost for running one operational low level server
$c_{M_{med}} \in \mathbb{R}^+$	energy cost for running one operational medium level server
$c_{On_{high}} \in \mathbb{R}^+$	energy cost needed to switch-on high level server
$c_{On_{low}} \in \mathbb{R}^+$	energy cost needed to switch-on low level server
$c_{On_{med}} \in \mathbb{R}^+$	energy cost needed to switch-on medium level server

The energy consumption takes into account the number of operational servers.

[3] Each job is a request for a web page view.

A data center may be composed of heterogeneous servers grouped essentially into several levels of energy consumption and service rate (for example a group with high, med, or low energy consumption). As servers are heterogeneous, each server of group g consumes some units of energy per slot when a server is operational and it costs at average c_{M_g} monetary unit. A server may consume a very low amount of energy when it sleeps. During the latency period (waking-up) a server may consume an additional amount of energy c_{On_g}.

A server consumes more energy when it processes a job. The total energy consumed is the sum of all units of energy consumed among a specific period. As we are considering the latency of servers, we assume that c_{On_g} is consumed uniformly during the latency period, let us say $\frac{c_{On_g}}{k}$ every slot, where k is the number time units to completely switch-on the server.

The QoS takes into account the number of waiting and lost jobs. Each waiting job costs c_N monetary unit per slot. A rejected job costs c_L monetary unit.

1.4 Outline and solution

1.4.1 Proposed solutions

In this thesis, our main work consists in designing efficient Dynamic Power Management systems (DPM) based on stochastic controlled queuing models when investigating their optimal control policy in order to apply them on data centers, which should satisfy the growing demands for minimizing the energy consumption and the digital pollution while preserving the quality of service (QoS).

First, as we observed that an important work was done for analyzing the problem of energy optimization in data centers in the context of continuous time (see Tables 2.9 and 2.8), our work is entirely devoted to the context of discrete time.

Next, and before showing how solving the energy optimization problem in an optimal way, we introduce how we can model specific mechanisms to solve the energy/QoS optimization problem by a sub-optimal solution. So, we introduce the use of DTMC to model and solve the problem of reducing the energy consumed in a data center while keeping a good QoS based on a predefined classical monotone policy which is based on the thresholds strategy. This type of policy belongs, in general, to the class of sub-optimal policies. After that, as a continuation, we present a more sophisticated stochastic optimization system that shows how *histogram operators* can be used to avoid generating explicitly a whole batch of DTMC's as done in the first approach. More precisely, we introduce the use of

histogram operators to model and solve the problem of saving the energy consumed in a data center when preserving it service performance based on the thresholds policy mechanism. Additionally we show how the use of coupling and stationarity detection algorithm [Ser99] can be used to make the computation of the best thresholds policy faster by around 40%.

Now, as a second work we focused on finding the optimal strategy for minimizing both energy consumption and QoS in data centers when formulate the model in a discrete time framework. Thus, we detail the use of the concept of *Markov Decision Process* to find the *optimal* strategy to ensure a reasonable energy consumption with a good performance. To find the optimal policy we formulate the optimization problem by a discrete-time MDP. First we focus on the modeling of the Dynamic Power Management by a stochastic model for a homogeneous data center principally to study some structural properties of the optimal policy such as monotony. Secondly, as data centers present a non negligible level of server heterogeneity in energy consumption and service rate, we generalize the homogeneous model to a heterogeneous one. Finally, as the switching-on (resp. switching-off) of a data center server is not instantaneous and needs some extra units of time to transit from a sleeping/stopping mode to a ready-working mode, we extend the model in order to include this temporal latency of the servers. Along all the chapter, arrival jobs and service rates are specified with histograms that may be obtained from real traces, empirical data, or incoming traffic measurements.

For the homogeneous model theoretical and experimental results show that the optimal policy is not monotone. Therefore we deduce that the strategy based on double-threshold structure leads only to a sub-optimal policy. Results show that MDP leads to the optimal policy when saving a significant amount of energy. But, it needs more computation time and more memory space to analyze and solve the optimization problem. We observe that threshold based approach computes the optimization strategy faster, and uses a small amount of space memory. However, it returns only a sub-optimal policy.

For heterogeneous model we proved that neither the monotony nor the isotony properties hold for optimal heterogeneous policy. Consequently, the optimal policy cannot be designed as a simple double-threshold structure. Additionally results show that the size of the heterogeneous model is bigger than the size of homogeneous model. However, from experimental results, it seems that increasing server heterogeneity leads to more potential energy savings.

For the model with latency results show that the size of model is huge, more precisely the size is an exponential of the period of latency. However, it still always worthwhile to build more complex models. From experimental results, it seems that increasing buffer size leads to more potential energy saving when the latency is bigger.

We show also that the size of any of the above MDP models is huge and leads to the state space explosion problem and an important computation time. In fact, we show that finding the optimal strategy, which requires passing through MDP, is often difficult even impossible to achieve for large data centers. Especially if we take into account real aspects like servers heterogeneity or latency. The problem comes mainly from the exponential size of the MDP structure and then the time needed to go through it to find the optimal strategy. So complexity is important both in space and in time. Indeed, we suggest what we call the greedy-window algorithm that allows to find a sub-optimal strategy better than that produced when considering special mechanism like the threshold approaches. And more important, unlike MDP approach, this greedy-window algorithm does not require the building or parsing all possible strategies. Thus our algorithm gives a strategy very close to the optimal strategy with very small space and time complexities.

1.4.2 Outline

In this chapter we showed how we can model a data center, in the context of energy consumption optimization, by a queue model when we describe its different aspects and components. Especially the arrival/service processes, the servers characterization, and the energy/QoS metrics. The rest of this document is organized as follow. In Chapter 2 we introduce the state of the art by analyzing several scientific papers that have addressed the problem of finding a trade-off between energy consumption and performance in data centers. Then, the first part of Chapter 3 introduces the use of DTMC to model and solve the problem of reducing the energy consumed in a data center when keeping a good QoS based on the thresholds strategy. After that, in the second part of Chapter 3 we show how *Histogram operators* can be used instead of generating explicitly DTMC's. Next, in Chapter 4 we detail the use of the concept of *Markov Decision Process* to find the *optimal* strategy. We first consider a homogeneous data center, after that we generalize the model and its results to the case of a heterogeneous data center. Moreover, we extend the concept of MDP for a data center when we consider the latency of servers. After that, in Chapter 5 we suggest an efficient space-time complexity greedy-window algorithm that allows to find a sub-optimal policy very close to the optimal one. Finally Chapter 6 closes the manuscript by comparing experimentally all presented approaches when arrival jobs and service rate are modeled by discrete distributions obtained from real Google traffic traces.

Chapter 2

State of the art

THE increasing development of *Data Centers* and recent expansion of *Clouds* are causing energetic problems and digital pollution issues. Several studies show a significant augmentation of energy consumed and digital pollution produced by computers. More than 1.3% of the global energy consumption is due to the electricity used by computer infrastructures. Additionally, one data center server can produce more than 10 kilogram of CO_2 per day, rates that are increasing, revealed by a survey conducted in [Koo11], which says a lot about the increasing evolution of data centers. Clouds and data centers are designed to support the expected peak traffic load, however the global load is about 60% of the peak load [BAM10]. In fact, an important number of servers are not under load and still consume about 65% of the maximal energy [GHMP09]. Thus, needs for energy saving is emerging. Studies like [BGDG+10, Bal11, LZ12] show that much of the energy consumed in the data center is due to the electricity used to run the servers and to cool them (70% of total cost of the data center). Thus the main factor of this energy consumption is related to the number of operational servers. Many efforts have focused on servers and their cooling. Works have been done to build better components and low-energy-consumption processors [GMIL+00], more efficient energy network [BAM10], more efficient cooling systems [PBSB03], and optimized kernels [JWC12]. Therefore, to ensure both a good performance of services offered by these data centers and reasonable energy consumption, a detailed analysis of the behavior of these systems is essential for designing efficient optimization algorithms to reduce the energy consumption. Against this background, a complementary approach to save energy is to consider a power policy to manage the switching-on/off of servers in a data center to ensure both a better quality of service offered by these data centers and reasonable energy consumption. Two requirements are in conflict:

1. Maintaining a high Quality of the Service (QoS).
2. Consuming less energy.

For the first requirement, we need to turn on a large number of servers which consumes more energy and leads to less waiting time and decreases the rate of losing jobs but needs a high energy consumption. For the second requirement, we need to turn on a small number of servers which leads to less energy consumption, but causes more waiting time and increases the rate of losing jobs. Thus, the goal is to design better power management algorithms which take into account these constraints to minimize waiting time, loss rate and energy consumption.

In the following section we analyze in more details several scientific papers that have addressed the problem of saving energy in data centers when keeping a optimal/reasonable QoS.

2.1 Analytical solutions based on CTMC

2.1.1 Thresholds policy without latency of servers

Schwartz et al. in [SPTG12] present a theoretical queuing model to evaluate the trade-off between waiting time and energy consumption if only a subset of servers is active all the time and the remaining servers are enabled on demand. In this paper the problem is modeled by a queuing model with an infinite buffer size $b = +\infty$. Authors assume that the jobs in their model arrive according to an independent Poisson process with rate λ and each server accepts only one job at a time with an exponentially distributed service time with mean $\frac{1}{\mu}$. Then, the system can be modeled using a $M/M/k$ queuing system. A number of m_0 servers called base-line-servers is active all the time, and a number of m_1 additional servers called reserve-servers is initially turned off and will be activated when the total number of jobs in the system increases and exceeds some threshold θ_1. The reserve-servers will be turned off again if the number of jobs in the system decreases and becomes less than another threshold θ_0. For a server, the authors consider three level of energy consumption:

1. c_{off} when the server is turned off,
2. c_{busy} when the server is turned on and proceeding a job, and
3. c_{idle} when the server is turned on without proceeding a job.

They also consider mainly two random variables. Random variable x gives the number of jobs in the system where only the base-line-servers are activated and

in this case $x(i)$ is the stationary probability that i jobs are in the system. And the random variable y gives the number of jobs in the system where additionally the reserve-servers are activated and in this case $y(i)$ is the stationary probability that i jobs are in the system. Using CTMC[1], and based on local balance equations of each state, they use analytical calculus to find the expression of each $x(i)$ and $y(i)$ in function of m_0, m_1, θ_0 and θ_1 (see Appendix A for detailed equations). Then they suggest to find the best m_0, m_1, θ_0 and θ_1 that minimize the following cost function:

$$cost = \alpha\mathbb{E}(e) + (1-\alpha)\mathbb{E}(w) \tag{2.1}$$

where e is the expected value of energetic cost and w is the expected value of waiting time. Parameter $\alpha \in [0,1]$ can be chosen in such a way that a desirable trade-off is made. Notice that a value of α close to 1 gives more importance to energy. A value of α close to 0 gives more importance to QoS.

Du to the hardness to solve analytically the above optimization problem, authors suggest to solve it in a trivial way by computing the cost function for all valid combinations of m_0, m_1, θ_0 and θ_1, sorting the cost function values and choosing the minimum.

Finally, the experimental simulation results done in this paper, show that configurations as the one given in Table 2.1, leads to an energy consumption significantly reduced while still having an acceptable waiting time. They observe that the service was guaranteed while still saving about 40% of energy compered to a situation in which all servers are always turned on.

Table 2.1: Experimental simulation settings in [SPTG12].

Parameter	**Value**	**Description**
$\mathcal{M} = m_0 + m_1$	100	Total number of servers
λ	0.10	Exponential inter-arrival job arrivals rate
μ	0.0025	Exponential service rate

Notice that in this paper:

1. The latency of waking up servers is not considered, servers are considered to switch on immediately.

2. The optimization problem is formulated by analytic equations however, neither exact nor heuristic solutions were given.

[1]Continuous Time Markov Chain.

3. The total cost does not contain the rejected jobs rate. That can be explained by the fact that the size of the buffer is infinite.

4. The servers in the reserve, are switched-on or switched-off as one block, the switching-on/off of additional servers is not done progressively.

5. The strategy is based on the number of jobs in the system and not the number of waiting jobs.

6. The problem is considered for the steady state.

2.1.2 Thresholds policy with latency of servers

In [Mit13] Mitrani considers the problem of managing servers of a data center in order to satisfy the conflicting objectives between high QoS and low energy consumption. As in [SPTG12] Mitrani modeled the problem by a queuing model with an infinite buffer size $b = +\infty$. He assumes that the jobs arrive according to an independent Poisson process with rate λ and each server accepts only one job at a time with an exponentially distributed service time with mean $\frac{1}{\mu}$. So, the system can be modeled using a $M/M/k$ queuing system. In addition, $\mathcal{M}$ is the total number of servers, a subset of m_1 servers called *reserve* is turned on when the number of waiting jobs in the system is sufficiently high and exceeds a threshold U (for Up), and is turned off when that number of jobs is sufficiently low and exceeds another threshold D (for Down). Unlike Schwartz et al. [SPTG12], Mitrani takes into account the latency of the servers, which is the period of time needed by a server to be switched on. During this period a server consumes an energy surcharge to be powered up while no job is served. All the reserve servers become operational together after an interval of time distributed exponentially with mean $\frac{1}{\nu}$. The main objective of this paper is to find the best size of the reserve m_1 and the best values of thresholds U and D that allow a minimum total cost. The author considers one level of energy consumption of a server, which means that energy consumption depends on the number of switched on servers. He mainly considers three random variables:

- Variable x gives the number of jobs in the system where the reserve is off and in this case $x(i)$ is the stationary probability that i jobs are in the system.
- Variable y gives the number of jobs in the system during latency period (the reserve is powering up). In this case $y(i)$ is the stationary probability that i jobs are in the system.

- Variable z gives the number of jobs in the system where additionally the reserve is powered on. In this case $z(i)$ is the stationary probability that i jobs are in the system.

Using CTMC, and based on local balance equations of each state, he uses analytical calculus (generating functions) to find the expression of each variable $x(i)$, $y(i)$ and $z(i)$ in function of m_1, U, and D where $0 < D < U < \mathcal{M}$ (see Appendix B for detailed equations). Mitrani suggests an heuristics solution for the above equations in order to give optimal values of m_1, U, and D that minimize the following objective cost function:

$$cost = c_1\mathbb{E}(n) + c_2\mathbb{E}(m) \tag{2.2}$$

where $\mathbb{E}(n)$ represents the average number of jobs in the system, $\mathbb{E}(m)$ gives the average number of servers consuming energy in the system, coefficients c_1 and c_2 reflect the relative importance placed on QoS and energy consumption, respectively. He obtained a heuristic value of m_1 (size of the reserve) by arguing as follows. Let $n(m_0)$ be the average number of jobs in a system where there are m_0 permanently operative servers. Consider the change in costs as the number of servers is increased from $m_0 - 1$ to m_0: the average queuing cost per unit time would decrease by $c_1\,(n(m_0-1) - n(m_0))$. At the same time, the energy consumption cost would increase by c_2 . Hence, the increase would be advantageous if $c_1\,(n(m_0-1) - n(m_0)) > c_2$. Consequently, the optimal number of permanently operative servers would be the largest m_0 for which an increase from $m_0 - 1$ to m_0 is advantageous. Mitrani proposed an heuristic that consists in approximating $n(m_0)$ by the $M/M/1$ expression: $n(m_0) = \dfrac{\frac{\lambda}{\mu}}{\mathcal{M} - \frac{\lambda}{\mu}}$, an optimal value of m_0 should be equal to:

$$m_0^* = \left\lfloor \frac{\lambda}{\mu} + \frac{1}{2}\left(1 + \sqrt{1 + 4\frac{\lambda c_1}{\mu c_2}}\right)\right\rfloor. \tag{2.3}$$

In fact the optimal size of the reserve is: $m_1^* = \max\{0, \mathcal{M} - m_0^*\}$.

Consider that we turn off the reserve when some of the non-reserve servers become idle. In this case, the heuristic value of the lower threshold D can be fixed to:

$$D^* = \mathcal{M} - m_0 - 1. \tag{2.4}$$

To derive a heuristic for the upper threshold, Mitrani uses a deterministic fluid approximation of the queuing process and he deduces:

$$U^* = \max\left\{m_0 - 1, \left\lceil \frac{m_0\mu - \lambda}{\nu} + \frac{c_2(m_0\mu - \lambda)}{c_1\mu}\left(1 + \sqrt{1 + \frac{2c_1(\mathcal{M}\,\mu - \lambda)}{c_2\nu(m_0\mu - \lambda)}}\right)\right\rceil\right\}. \tag{2.5}$$

Over parameters shown in Table B.1, Mitrani carried out several numerical experiments to evaluate the quality of his heuristics.

Table 2.2: Experimental simulation settings in [Mit13].

Parameter	**Value**	**Description**
$\mathcal{M} = m_0 + m_1$	20	Total number of servers
λ	4, 8, 10, 12	Exponential inter-arrival job arrivals rate
μ	1	Exponential service rate
ν	0.1	Exponential latency rate

He observed that when the size of reserve m_1 is well chosen, it is not important if the upper threshold U is overestimated. On the other hand, if m_1 is badly chosen, then both underestimating and overestimating the optimal threshold U can increase the cost substantially. And as expected, the heavier the offered load, the fewer servers should be reserved. Finally he compares the performance of the optimal policy (m_0, U, and D are chosen optimally), with that of the heuristic policy (using the heuristic values for m_0, U, and D), and also with the *do nothing policy* of not reserving any servers. His experimental results confirm that the heuristic policy is practically indistinguishable from the optimal policy, with only a small difference. This has been observed to be the case for a large variety of system configurations and cost coefficients. As future work, Mitrani suggests to extend his model by introducing k blocks of reserves, of sizes $m_1, m_2, \ldots, m_k$, with associated upper and lower thresholds. Reserve 1 is turned on when the queue exceeds level U_1, reserve 2 is turned on when the queue exceeds level $U_2 > U_1$, etc.

Notice that in this paper:

1. The latency of waking up servers is considered.
2. The optimization problem is formulated by an analytical equations then a heuristic solutions were given.
3. The size of the buffer is infinite, in fact rejected jobs rate was not considered.
4. The servers in the reserve, are switched-on or switched-off as one block, the switching-on/off of additional servers is not done progressively.
5. The strategy is based on the number of waiting jobs.
6. The problem is considered for the steady state.

2.1.3 Allocation policy with rejected jobs

In [DM10] Dyachuk and Mazzucco study the problem of maximizing the revenue of a service provider when preserving the performance requirement of users. Their strategy is based on a dynamic allocation policies that keep turning-on the minimum number of servers which are necessary to meet the user's QoS requirements. The solution is to dynamically power servers up and down according to the incoming load. In this paper the problem is modeled by a queuing model with an infinite buffer size $b = +\infty$ and a total number of $\mathcal{M}$ servers. The authors consider three level of energy consumption of a server:

1. c_{off} when the server is switched-off,
2. c_{busy} when the server is turned on and processing a job, and
3. c_{idle} when the server is idle, which means, turned on without proceeding any job.

Additionally, Dyachuk and Mazzucco consider that powering servers on/off takes, in average, k units of time.

The jobs are assumed to be impatient. This is means that a waiting job can leave the queue while waiting for the server to respond. In order to model this impatience authors assume that jobs sitting in the queue can time out and abandon the buffer. It is obvious that exact time-outs associated with specific jobs are not known, however their distribution can be estimated. Notice also that an abandoned job does not generate any profit. Hence, the provider should ensure a small waiting time which does not exceed the patience of jobs, otherwise jobs will start aborting their requests. Authors assume that jobs enter the system according to an independent Poisson process with rate λ, the service times are exponentially distributed with the mean of $\frac{1}{\mu}$. Job's patience (time-outs) is modeled by a random variables distributed exponentially with mean $\frac{1}{\theta}$, where θ represents the abandonment rate. ($\theta = +\infty$ corresponds to jobs with no patience, $\theta = 0$ corresponds to jobs with infinite patience).

Thus, the resulting model is an $M/M/k$ queue model with impatience. In this paper authors are interested in maximizing the average revenue earned by the service provider per unit time. That value can be estimated as:

$$revenue = c \times T - r \times P \tag{2.6}$$

where c is the income generated by each completed job, T is the number of served jobs, r is the unitary cost of energy, and P is the total energy consumed by the powered up servers.

The strategy used in this paper is to call a policy at specific times. The interval between consecutive policy invocations is called *observation epoch*. From the statistics collected during the last observation epoch, their policy estimates the arrival rate λ and average service time $\frac{1}{\mu}$. Then it allocates during the next epoch an optimal number of servers which should maximize the revenue during the epoch. They use a binary search algorithm requiring $\log \mathcal{M}$ iterations to find this optimal number. They also suggest a heuristic (based on work given in [Gra88]) to get a sub-optimal value of the number of servers to be ran as:

$$m^* = \frac{\mathbb{E}(\lambda)}{\mu} + \alpha\sqrt{\frac{\mathbb{E}(\lambda)}{\mu} + \frac{\mathrm{Var}(\lambda)}{\mu^2}} \tag{2.7}$$

where λ is estimated using some forecasting techniques, $\mathbb{E}(\lambda)$ refers to the expected value of λ, and $\mathrm{Var}(\lambda)$ describes the variance of the prediction.

Over real parameters shown in Table 2.3, Dyachuk and Mazzucco carried out several numerical experiments and simulations to evaluate the quality of their exact and heuristics strategies by using Wikipedia traces of November 2009 to model the job arrivals.

Table 2.3: Experimental simulation settings in [DM10].

Parameter	**Value**	**Description**
$\mathcal{M}$	1000	Total number of servers
λ	-	Modeled dynamically using Wikipedia traces
μ	10	The average service time is $1/\mu = 0.1$ seconds
θ	0.25	The average job patience is $1/\theta = 4$ seconds
r	0.1\$ per kWh	The cost for energy
c	6.2×10^{-6}\$	The profit generated by a successfully processed job

The various numerical analyses showed that the optimal strategy improves the revenue with a percentage between 10% and 30%, while the heuristic strategy performs with results close to the optimal strategy.

Notice that in the work of Dyachuk and Mazzucco:

1. The size of the buffer is infinite, however an abandoned jobs rate was considered.
2. The latency of waking up servers is considered.
3. The optimization problem is formulated by a dynamic optimization model then an exact and also a heuristic solutions were given.

4. The arrival jobs rate is estimated statistically by analyzing the past, each time the policy is called.

5. Dyachuk and Mazzucco suggest to consider a supplementary cost as the system's reliability is affected by switching on/off the servers (hardware components tend to degrade faster with frequent power on/off cycles).

2.1.4 Impact of data center size on the effectiveness of DPM

Most energy optimization techniques aim to minimize energy waste in data centers by switching servers off when they are not busy and then switch them on when it is needed. However, waking up a server requires a period of latency, in which the QoS can degrade. Thus, in [GHB11] Gandhi et al. examine the impact of the size of the data center (total number of servers) on the efficiency of the energetic optimization. As in [Mit13] Gandhi et al. modeled the problem by a queuing model with an infinite buffer size $b = +\infty$. They assume that the jobs arrive according to an independent Poisson process with rate λ and each server accepts only one job at a time with an exponentially distributed service time with mean $\frac{1}{\mu}$. So, the system can be modeled using a $M/M/k$ queuing system. They compare two strategies: The first one called *always-on* is to keep all the $\mathcal{M}$ servers always on. The second one called *on/off* is to turn off a server which is not busy and turn on additional server for each new waiting job. The main parameter of this paper is the latency (the period needed to turn on/off a server and the energy needed during this period). Thus, authors consider that a server becomes operational after an interval of time distributed exponentially with mean $\frac{1}{\nu}$. They model the problem by a Markov chain where each state is defined as the couple of the number of busy servers and the total number of jobs in the system. Their analysis is based on Matrix-analytic methods, which are numerical methods for analyzing complex unbounded Markov chains. In order to compare the two strategies, they use *PPW*, *Performance-per-Watt metric*, defined as:

$$PPW = \frac{1}{w \times e} \tag{2.8}$$

where w is the mean response time defined as the time from when a job arrives until it completes service. And e the mean power consumption. *PPW* measures the amount of service that can be delivered by a server for every watt of power consumed.

The numerical analysis done in this paper (configurations given in Table 2.4), find that for smaller data centers ($\mathcal{M} \leq 50$), the on/off strategy is often actually worse than always-on strategy with respect to both mean response time and mean

power, even when the load is low. However, as the size of the data center grows, the on/off strategy becomes more and more efficient, eventually outperforming the always-on strategy by saving more than the half of energy, while achieving the same response time.

Table 2.4: Experimental numerical settings in [GHB11].

Parameter	Value	Description
$\rho = \frac{\lambda}{\mathcal{M} \times \mu}$	30%	System load
ν	0.01	Exponential latency time (the mean latency is $1/\nu = 100$ seconds)

Additionally, authors find that energy optimization is very effective when the latency period is short or when the service time is large. They conclude that efficiency of energy optimization increases with the size of the data center.

2.1.5 Multiple arrival jobs process

Aidarov et al. present a theoretical queuing model in [AEM13] to evaluate the trade-off between energy and QoS in order to maximize revenues from the service provider where a penalty is paid by the supplier if the QoS provided is less than promised. In this paper the problem is modeled by a queuing model with an infinite buffer size $b = +\infty$. Authors assume that the jobs in their model may be of k different types. The jobs arrive according several independent Poisson process with rates $\lambda_1, \lambda_2, \ldots, \lambda_k$ and each server accepts only one job at a time with an exponentially distributed service time with means $\frac{1}{\mu_1}, \frac{1}{\mu_2}, \ldots, \frac{1}{\mu_k}$. Then, the system can be modeled using a queuing system. A number of $\mathcal{M}$ servers is considered. The provider must make dynamic decisions about whether or not to accept incoming jobs and how many servers to employ. More precisely, if new arrival jobs come, the following actions may be considered:

- reject the new jobs,
- accept the new jobs but do not switch on any new servers,
- accept the new jobs and power up one new server,
- accept the new jobs and power up two new servers,
- ...
- accept the new jobs and power up all servers.

If, w_i, the average of waiting time of jobs of type i exceeds some obligation o_i a penalty of q_i should be paid by provider to the user. The performance of the system is measured by the average profit R received. This is given by:

$$R = \sum_{i=1}^{k} a_i(r_i - q_i \times \mathcal{P}(w_i > o_i)) - c \times m \tag{2.9}$$

where:

1. a_i is the average number of jobs of type i that are accepted into the system,
2. r_i is the income generated by serving a job of type i,
3. $\mathcal{P}(w_i > o_i)$ is the probability that the observed average waiting time of jobs of type i, exceeds the obligation o_i,
4. c is the cost of running one server for one unit of time,
5. m is the average number of servers that are operational.

The objective of the authors is to maximize revenues from the service provider. The strategy should find a balance between minimizing the penalties, minimizing the cost of energy, and maximizing the number of served jobs. The policy proposed is an heuristic that must be applied at arrival job instances. The policy estimates for each new job and for every possible action, the income and the penalty that can be generated before accepting the job and before serving it, the strategy consists in choosing the action that maximizes the total revenue for this new job. They used $GI/G/n$ queues to approximate their model and estimate the revenue.

To test their heuristic method, authors carried out a number of simulation experiments where the total number of servers is $\mathcal{M} = 40$ and rates λ_i are close to 0.1. In all cases, the profit achieved by the suggested policy was $20\% - 35\%$ higher compared to a system where the servers are all operational.

We notice that:

1. The size of the buffer is infinite.
2. The optimization problem is formulated by a dynamic optimization model then a heuristic solutions was given.
3. The arrival jobs rate is estimated statistically by analyzing the past, each time the policy is called.
4. Authors do not consider the latency of waking up servers.
5. The servers are considered to switch on/off immediately.
6. No additional energetic cost was considered to turn on servers.

2.2 Continuous time MDP solutions

Yang et al. [YCNH11] study the saving of energy in data centers, where a data center has multiple servers to deal with jobs. The servers are switched into *sleeping mode* in periods of low traffic load to reduce energy consumption while guaranteeing the quality of service (waiting jobs). The problem was formulated as an MDP, *Markov decision process*. Unlike previous papers, in this paper, authors modeled the problem by a queuing model with a finite buffer of size $b \neq +\infty$. They assume that the jobs arrive according to an independent Poisson process with rate λ and each server accepts only one job at a time with an exponentially distributed service time with mean $\frac{1}{\mu}$. The authors consider only two levels of energy consumption of a server:

1. c_{idle} when the server is in sleeping mode (not working but consuming a small amount of energy),
2. c_{busy} when the server is turned on.

They considered no latency for switching the servers from a mode to another, however they consider c_{on} a specific energetic cost when switching server from sleeping mode to working mode. When $\mathcal{M}$ is the total number of servers and m the number of current running servers, and n the number of current waiting jobs, the state of the system is mapped to (m,n). From every state of the system they consider all possible $(\mathcal{M}+1)$ actions:

1. action to turn off all servers,
2. action to turn on only 1 server,
3. action to turn on only 2 servers,
4. ...
5. action to turn on $(\mathcal{M}-1)$ servers,
6. action to turn on all servers.

Applying an action leads the system to another state and gives rise to a local cost. The strategy consists in finding the best sequence of actions to minimize the total cost during a period of time:

$$cost = e_{work} + e_{on} + \alpha \times w \tag{2.10}$$

where e_{work} denotes the energy consumption of all working servers, e_{on} denotes the energy consumption for switching servers, and w is a penalty for keeping jobs

in waiting state. Parameter α can be used to strike a balance between energy consumption and waiting job penalty.

In fact, the problem was formulated by an MDP and the value iteration algorithm was used to calculate the optimal policy.

The authors prove that under previous conditions and assumptions, the optimal control policy has a double threshold structure. It means that there exists a set of $\mathcal{M}$ up-thresholds $\{U_1, U_2, \ldots, U_{\mathcal{M}}\}$ and another set of $\mathcal{M}$ down-thresholds $\{D_1, \ldots, D_{\mathcal{M}}\}$ such that, the optimal strategy is defined as the following. Consider that m is the number of current running servers, and n is the number of current waiting jobs. The action to perform from this state is:

$$\begin{cases} \text{turn off one server if:} & n < D_m \\ \text{keep the same number of servers if:} & D_m \leq n \leq U_m \\ \text{turn on one additional server if:} & U_m < n. \end{cases}$$

The experimental simulation done in this paper confirms this behavior, and shows that applying the optimal policy for a system with configurations as the one given in Table 2.5 saves around 19% of energy consumption.

Table 2.5: Experimental simulation settings in [YCNH11].

Parameter	**Value**	**Description**
$\mathcal{M}$	10	Total number of servers
λ	6	Exponential inter-arrival job arrivals rate
μ	1	Exponential service rate

Notice that in this paper:

1. The latency period of waking up servers is not considered, only the energetic cost of waking up servers is considered
2. The optimization problem is formulated as Markovian Decision Process, then an algorithm which gives the optimal solution was given.
3. Although the buffer size is finite, the rejected jobs were not considered.
4. The servers are switched-on or switched-off progressively one by one.
5. The problem is considered for a period of time.

2.3 Discrete time MDP solutions

2.3.1 Constrained optimization

All previous presented papers consider the problem of energy optimization in the case of continuous time. In [BBPDM99] Benini et al. consider the problem for the discrete time. They modeled the system by a queuing model with a finite buffer size $b \neq +\infty$. A server is modeled by an MDP, where possibles states can be $\{on, off, \dots\}$. Actions are defined over this server states to switch from a state to another with some probability. The Markov chain that models a server, can be used to deduce the latency to wake up a server (the transition time from the off-state to the on-state when a action has been issued with probability p is a geometric random variable with average equal to $\frac{1}{p}$ units of time). Additionally, they define matrices used to set the service rate and the energy consumption for every server state. Job arrivals are modeled by a discrete Markov chain where states represent the possible numbers of jobs that can arrive per unit of time. Authors combine the MDP that models the server with the DTMC that models arrival jobs to model the entire system by a global MDP where each state is defined as the state of the server added to the number of jobs.

As the authors are interested in the steady state behavior, a policy is modeled by an infinite sequence of actions. Hence, the goal of this paper is to search the space of all possible policies to find the one that minimizes a cost metric. Authors define several cost metrics:

1. power e: average of consumed energy per unit of time,
2. performance s and r: average number of served jobs per unit of time, and average number of rejected jobs per unit of time.

Instead of using an objective cost function, they target the optimization of one cost metric while using the second as a constraint, for example:

1. Minimize power e when keeping performance more than some threshold:

$$s \geq s_0. \tag{2.11}$$

2. Minimize power e while not exceeding some loss rate:

$$r \leq r_0. \tag{2.12}$$

3. Maximize performance s while not exceeding some energetic threshold:

$$e \leq e_0. \tag{2.13}$$

Authors used some classical results of stochastic optimization to formulate these two last stochastic optimization problems as a linear programming (LP) optimization problem, that can be solved exactly in polynomial time by using the interior point algorithm [Ros14].

Benini et al. evaluate the strength of their model by constructing the stochastic model for a real-life device (disk drive, Web server, and CPU) under a realistic workload. Then they apply their optimization algorithm and compute optimal policies. The most interesting for us is the second one. They modeled a web server with two processors for a high-traffic web site, which is an example of a system with multiple servers. They set the time unit to five seconds. Then they analyze the system for a period of one day. The two servers are not identical. The first server has higher performance (1.5 times) and higher energy consumption (2 times) than the second one. The job arrivals model was extracted from real-life traces obtained by monitoring a busy web server (Internet Traffic Archive, `http://ita.ee.lbl.gov/`). Then they constructed a simple two-state Markov model for the workload. Then they apply their optimization algorithm and compute optimal policies. Experimental results show that performance and energy consumption of the obtained policy is validated by simulation. Theoretical and experimental results were close.

Notice that in this paper:

1. The latency period of waking up servers is considered, hence, the energetic cost of waking up servers is also considered.

2. The optimization problem is formulated as Markovian Decision Process, then a polynomial algorithm which gives the exact solution was given.

3. As the buffer size is finite, the rejected jobs are considered.

4. The problem is considered for a steady state.

5. The model support heterogeneous servers.

6. The model is presented and explained by real-life examples for cases where the number of servers was very small (one, two,...). The application of this model for a data center of hundreds (or thousands) of servers leads to a huge MDP and certainly to the explosion of state space of $O(2^{\text{number of servers}})$.

2.3.2 Probabilistic model checking for DPM

In order to reach a trade-off between the QoS and the energy consumption of a system, in [NPK+05] Norman et al. present an approach to analyzing a stochastic dynamic power management strategy using the formalism of probabilistic model checking. For a given strategy, their probabilistic model checking formalism allows them to formally establish several quantified probabilistic properties over QoS, buffer size, latency, and energy consumption. In this paper, the authors used PRISM to model an energy optimization problem based on continuous-time Markov chains. The system they analyzed is composed of:

1. Service Provider (SP), which contains only one server which has several energetic levels, each level with a specific amount of energy consumption (see Table 2.7). The latency period to move form a state to another is given in Table 2.6.
2. Service Requester (SR), which issues jobs to the server.
3. Service Request Queue (SRQ), a finite buffer which stores immediately arrival jobs that are not serviced yet ($b \neq +\infty$).
4. Power Manager (PM), which issues commands to the SP, based on observations of the system.

Table 2.6: Average latency period between energy levels (millisecond) in [NPK+05]

	active	**idle**	**idlelp**	**standby**	**sleep**
active	-	1	5	220	600
idle	1	-	5	220	600
idlelp	5	-	-	220	600
standby	600	-	-	-	-

Table 2.7: Average of energy consumption in [NPK+05]

State	sleep	standby	idle	active
Energy (W)	0.1	0.3	1.5	2.5

In this work, each component (SP, SR, SRQ and PM) is represented by an individual PRISM module. Listing 2.1 shows the PRISM specification for the service requester module (SR). Listing 2.2 presents the PRISM specification for the service queue module (SRQ).

Listing 2.1: The SR has two states, *idle*, where no requests are generated, and *req*, where one request is generated per time slot.

```
// SERVICE REQUESTER
module SR
    sr : [0..1] init 0; // 0 - idle and 1 - 1req
    [tick2] sr=0 -> 0.898: (sr'=0) + 0.102: (sr'=1);
    [tick2] sr=1 -> 0.454: (sr'=0) + 0.546: (sr'=1);
endmodule
```

Listing 2.2: To model the arrival and service of requests the transitions of the SRQ are dependent on the state of both the SR and the SP. Since, either the SR is in state *idle* and no requests arrive or in state *req* and one request arrives, and the SP can only serve requests when it is in state active.

```
// SERVICE REQUEST QUEUE
module SRQ

    q : [0..2] init 0;

    // do not serve and nothing arrives
    [tick2] sr=0 & sp>0 -> true;
    // do not serve and a request arrives
    [tick2] sr=1 & sp>0 -> (q'=min(q+1,2));
    // serve and nothing arrives
    [tick2] sr=0 & sp=0 -> (q'=max(q-1,0));
    // serve and a request arrives arrives
    [tick2] sr=1 & sp=0 -> true;

endmodule
```

As said before, each component is represented by an individual PRISM module, then, the PRISM model checking tool was used to construct a generic model of the energy optimization system. Using the transition matrix of this system, a linear optimization problem was formulated as described in [BBPDM99]. After that, the linear optimization problem was passed to the MAPLE symbolic solver to calculate the optimal policy. For example to reach a system with a number of waiting jobs not exceeding 1.5 with the minimum of energy consumption, they found that the optimal stochastic strategy is described in Algorithm 1.

Under different optimization constraint on the queue size, they also use PRISM to compute and plot the expected energy consumption, the expected number of waiting jobs, and the expected number of rejected jobs. They observe that policies which consume less energy have larger queue sizes and tend to lose more jobs.

In a second part of their work, they model and formalize the same system but under continuous time when using CTMC. They again use MAPLE to perform the solution of the associated optimization problem. They compare the performance

of the system (the expected number of waiting jobs and the expected number of rejected jobs) and its energy consumption under different constrains:

1. high performance: number of waiting jobs ≤ 0.1,
2. average performance: number of waiting jobs ≤ 1,
3. low performance: number of waiting jobs ≤ 5,

where the inter-arrival time of arrival jobs is modeled by deterministic, exponential, Erlang, uniform, or Pareto distributions. So, this paper show how probabilistic model checking (especially PRISM) allows for an automatic generation of a range of QoS measurements in order to analyze energy optimization policies. In contrast to simulation, the advantage of the model checking approach lies in the exhaustiveness of the analysis. This means that the computed results are guaranteed to be accurate with respect to the probabilistic model used, and that all behaviors are included during the analysis.

Algorithm 1: Example of optimal strategy given in [NPK$^+$05].

```
if SP is active and SRQ is not full then                    1
    with probability 1 goto idle;                           2
else                                                        3
    if SR is idle, SP is in sleep and the SRQ is full then  4
        with probability 4.7 × 10^-7 goto active;           5
    else                                                    6
        if SP is idle then                                  7
            with probability 1 goto active;                 8
        end                                                 9
    end                                                     10
end                                                         11
```

2.4 Papers consolidation

In the literature most works were done in the context of continuous time [Mit13, AEM13, MD12, SPTG12, GHB11, GHBA10, GGHBK10, DM10, XT08, NPK$^+$05, MD15, YCNH11, STM08, Efr04, LV02, Li12, KAR11, EMSM17, GLNT13, GDHBSW13]. Less works were done in the context of discrete time [NIG07, LKGN16, BBPDM99].

Works as in [GDHBSW13, AEM13, MD12, SPTG12, GHB11, GHBA10, GGHBK10, DM10, XT08, NPK$^+$05, STM08, EMSM17, GLNT13, Mit13] consider models under specific management mechanisms, that leads to sub-optimal

strategies like threshold policies. They model homogeneous data center servers as a theoretical queuing model to evaluate the trade-off between QoS and energy consumption where jobs arrive according to a Poisson process then served according to an exponentially distributed service time. The size of the buffer is in general infinite but finite in some works as in [NPK+05, STM08, EMSM17, GLNT13].

Most works consider servers as homogeneous, however other papers study the case of heterogeneous data centers where arrival jobs are served by several non identical servers (servers with different speed and different power consumption) [NIG07, Li12, KAR11]. Markov chains and specific Petri nets were used to model the system then, an analysis based on analytic equation and/or experimental simulation shows that energy consumption is significantly reduced (from 10% to 40%) while still keeping a reasonable QoS.

On the other hand other works focused on computing the optimal policy. To achieve this aim, the authors in [BBPDM99, Efr04, LV02, MD15, YCNH11, LKGN16] formalize the energy saving problem by *Markov Decision Process* (MDP). MDP models can be considered in continuous or discrete time.

Additionally, most of studies consider the latency of servers, time latency (period needed to switch on/off a server) as in [Mit13, XT08], or energetic latency (additional energy needed to switch on/off a server) as in [SPTG12, DM10, NPK+05, Efr04, YCNH11, LKGN16], or both as in [MD12, GGHBK10, GHB11, GHBA10, STM08, EMSM17, GLNT13, MD15, GDHBSW13, BBPDM99].

In this chapter we detailed the most important elements of optimizing energy in data centers related to the literature. In the next chapter we will introduce the modeling of the energy/QoS optimization problem when using a predefined specific mechanism of optimization which leads to only a sub-optimal policy. However its ease will help us to prepare the reader to better embrace the optimal strategy computation using MDP. Tables 2.9 and 2.8 summarize the most important aspects of different articles studying energy optimization in data centers.

Table 2.8: Consolidated of previous detailed papers with other works

	Discrete time	**General arrivals**	**Servers heterogeneity**	**Server latency**	**Strategy optimality**
[LKGN16]	✓	✗	✗	✓	✓
[NIG07]	✓	✓	✓	✗	✗
[BBPDM99]	✓		✓	✓	✓
[AEM13]	✗	✗	✗	✗	✗
[SPTG12, DM10, STM08, Mit13, XT08, EMSM17, GDHBSW13, GLNT13, GHB11, GHBA10, MD12, GGHBK10]	✗	✗	✗	✓	✗
[YCNH11, MD15]	✗	✗	✗	✓	✓
[KAR11, Li12]	✗	✗	✓	✗	✗
[LV02]	✗	✗	✓	✗	✓
[Efr04]	✗	✗	✓	✓	✓
[NPK+05]	✗	✗	✗	✓	✗

Table 2.9: Collection of several papers on energy optimization in data centers

Paper	Cited	Time	Arrivals	Service	Servers	Buffer	QoS	Time latency	Energetic latency	Model	Solving method	Strategy	Energy saving
[EMSM17]	9	Continuous	Poisson	Exponential	Homogeneous	Finite	Response time throughput	Exponential	Yes	SAN	Fixed-point iteration	Suboptimal	5% – 10%
[LKGN16]	4	Discrete	Interrupted Bernoulli	Exponential	One server	Finite	Waiting time	No	Yes	MDP	Dynamic programming	Optimal	-
[MD15]	7	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	System jobs	Exponential	Yes	MDP	Analytic	Optimal	-
[Mit13]	70	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Waiting jobs	Exponential	No	Markov chain	Analytic simulation	Suboptimal	-
[GDHBSW13]	60	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Response time	Exponential	Yes	Markov chain	Analytic simulation	Suboptimal	40%
[AEM13]	2	Continuous	Poisson family	Exponential family	Homogeneous	$+\infty$	Waiting time	No	No	Markov chain	Analytic simulation	Suboptimal	20% – 35%
[GLNT13]	82	Continuous	Poisson	Exponential	Homogeneous	Finite	Response time rejected jobs	Exponential	Yes	SRN	Fixed-point iteration	Suboptimal	-
[Li12]	22	Continuous	Poisson family	General	Heterogeneous	$+\infty$	Response time	No	No	Queuing model	Analytic numerical	Suboptimal	-
[MD12]	43	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Rejected jobs	Deterministic	Yes	Markov chain	Analytic simulation	Suboptimal	10% – 30%
[SPTG12]	25	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Waiting time	No	Yes	Markov chain	Analytic simulation	Suboptimal	40%
[KAR11]	22	Continuous	Poisson	Exponential	Heterogeneous	$+\infty$	Waiting time	No	No	Queuing model	Analytic	Suboptimal	-
[GHB11]	48	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Response time	Exponential	Yes	Markov chain	Analytic simulation	Suboptimal	-
[YCNH11]	3	Continuous	Poisson	Exponential	Homogeneous	Finite	Waiting jobs	No	Yes	MDP	Value iteration algorithm	Optimal	19%
[GHBA10]	120	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Response time	Exponential	Yes	Markov chain	Analytic simulation	Suboptimal	-
[GGHBK10]	225	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Response time	Deterministic	Yes	Markov chain	Analytic simulation	Suboptimal	-
[DM10]	17	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Rejected jobs	No	Yes	Markov chain	Experimental	Suboptimal	10% – 30%
[XT08]	14	Continuous	Poisson	Exponential	Homogeneous	$+\infty$	Waiting time	Exponential	No	Markov chain	Analytic	Suboptimal	
[STM08]	22	Continuous	Bi-Poisson	Exponential	Homogeneous	Finite	Response time	Exponential	Yes	MDP	Analytic heuristics	Suboptimal	-
[NIG07]	186	Discrete	Benchmarks	Benchmarks	Heterogeneous	Finite	Throughput service	No	No	Analytical prediction	Experimental	Suboptimal	20%
[NPK+05]	86	Continuous	Markov chain	Markov chain	Homogeneous	Finite	Waiting jobs	No	Yes	Markov chain	Linear programming	Suboptimal	-
[Efr04]	24	Continuous	Poisson	Exponential	Heterogeneous	Finite	System jobs	No	Yes	MDP	Value iteration algorithm	Optimal	-
[LV02]	28	Continuous	Poisson	Exponential family	Heterogeneous	$+\infty$	System jobs	No	No	MDP	Linear programming	Optimal	-
[BBPDM99]	499	Discrete	Markov chain	Markov chain	Heterogeneous	Finite	Waiting jobs rejected jobs	Markov chain	Yes	MDP	Linear programming	Optimal (Constrained)	-

Chapter 3

Thresholds policy

BEFORE solving the energy optimization problem in an optimal way, Let us begin by introducing how we can model specific mechanism to solve the energy/QoS optimization problem by a sub-optimal solution. At first, this chapter introduces the use of DTMC to model and solve the problem of reducing the energy consumed in a data center when keeping a good QoS based on a monotone strategy which is based on the thresholds strategy that belongs, in general, to the class of sub-optimal policies. After that, and as a continuation of DTMC based approach, we will present a more involved stochastic optimization system when we still solve it by special mechanism that leads to a sub-optimal policy. Thus, this second approach shows how *histogram operators* can be used to avoid generating explicitly a whole batch of DTMC's as done in the first approach. More precisely, we introduce the use of histogram operators to model and solve the problem of saving the energy consumed in a data center when preserving it service performance based on the thresholds.

3.1 Thresholds policy based on Markov chain

Let DC be a data center composed of $\mathcal{M}$ identical servers (see Section 4.2 for Markov model with no identical servers). DC receives jobs requesting the offered service. In order to keep the chapter easy, we will consider the following assumptions. The maximal number of jobs that can be served by one server in one slot is assumed to be constant and denoted by d. So the service rate can be modeled by a Dirac histogram $\mathcal{H}_D = \Delta_d$ (see Section 4.2.4 for Markov model with no constant service rate). Thus, the number of jobs arriving to the data center during a slot is modeled by a histogram $\mathcal{H}_A$ where $P_A(i)$ gives the probability to have i arrival

jobs per a slot. Note that we assume that arrivals of jobs are independent, and their distribution P_A is obtained from real traces, empirical data, or incoming traffic measurements. Thus, the queuing model is a batch arrival queue with constant services and finite capacity buffer b (buffer size). As said before to keep this first modelization simple, neither energetic latency nor time latency are considered, we assume that a server switches on instantaneously without consuming any additional amount of energy. Otherwise considering latency in a DTMC model makes the model heavily complex (see Section 4.3 for Markov model with latency).

The number of waiting jobs in the buffer is denoted by n. The number of operational servers is denoted by m. The number of rejected (lost) jobs is not considered in this chapter. We assume that initially the number of operational servers, the number of waiting jobs are 0. The maximal number of servers that can be operational is $\mathcal{M}$. It is assumed that the input arrivals are i.i.d. and under these assumptions, the model of the queue is a time-homogeneous Discrete Time Markov Chain.

3.1.1 Optimization mechanism

A Markov process is a stochastic process having the property to analyze the future only by knowing the present. In the following we focus on Markov chains with discrete time. A discrete time Markov chain is a sequence $X_0, X_1, \ldots, X_t, \ldots$ of random variables with values in some state space E. The characteristic property of a Markov chain is: analyzing the future from the present may not be more precise by considering the past, because all the useful information for the analysis of the future is contained in the present state of this process. In general, this memory-less property is expressed by the following formula:

$$\forall t \geq 0, \forall (s_0, \ldots, s_t, s_{t+1}) \in E^{t+2} : \tag{3.1}$$

$$\mathcal{P}\Big(X_{t+1} = s_{t+1} \mid X_0 = s_0, \ldots, X_t = s_t\Big) = \mathcal{P}\left(X_{t+1} = s_{t+1} \mid X_t = s_t\right) \tag{3.2}$$

$\mathcal{P}$ denotes the probability of an event. If the transition mechanism does not change over time the Markov chain is called homogeneous. The homogeneity property is expressed as follows:

$$\forall t \geq 0, \forall (e, e') \in E^2 : \qquad \mathcal{P}\left(X_{t+1} = e' \mid X_t = e\right) = \mathcal{P}\left(X_1 = e' \mid X_0 = e\right). \tag{3.3}$$

In the following we only consider homogeneous Markov chains. The concept of Markov chain can be used to analyze a predefined energy optimization strategy. We will consider an energy/performance management strategy inspired (but not identical) by the work given in [SPTG12] and [Mit13] in which a group of servers

is active all the time and the remaining servers are activated one by one on request. So we will define two thresholds:

1. σ: a number of servers always running (minimum service), the rest of servers $(\mathcal{M} - \sigma)$ are initially stopped and switched-on one by one gradually if needed.
2. β: if the number of waiting jobs n is greater than β, an additional reserve server will be switched on, otherwise if the number of waiting jobs is smaller than β, a server will be stopped.

This energy management strategy and performance is depicted in Figure 3.1 and described in Algorithm 2.

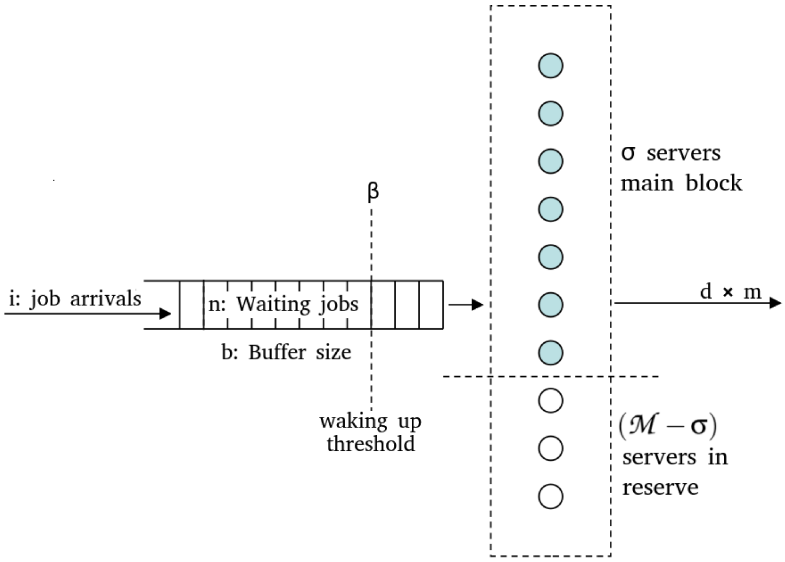


Figure 3.1: Optimization based on β/σ thresholds mechanism.

Algorithm 2: Management algorithm for thresholds policy.

```
Data: m, n, M, σ, β
Result: Update m
if n > β and m < M then
    m ← m + 1;
else
    if n ≤ β and m > σ then
        m ← m − 1;
    end
end
return m
```

The state space of the underlying DTMC, E is:

$$E = \{(m,n) \mid 0 \leq m \leq \mathcal{M} \text{ and } 0 \leq n \leq b\}. \tag{3.4}$$

We have a probability of $\mathcal{P}_{ss'}$ to move from state $s = (m,n)$ to $s' = (m',n')$ under a transition from state s to state s'. This probability is defined as:

$$\mathcal{P}_{ss'} = \sum_{\begin{cases} \text{for} & \text{each } i \in S_A \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-d\times m\}\} \\ m' & = \max(\mathcal{M}, m+1) \text{ if } n' > \beta \text{ or } \min(\sigma, m-1) \text{ if } n' \leq \beta \end{cases}} P_A(i). \tag{3.5}$$

The goal is to find the best strategy by defining the pair (σ,β) that minimizes at the same time the cost of energy and the cost of QoS. In other words minimizing a cost function that combines the cost of consuming energy and the cost of the degradation of the QoS. In order to keep things simple in this first part we will consider the cost function: $c = c_N \times n + c_M \times m$.

In order to find the best (σ,β) that minimizes c, we have to build for each pair (σ,β) a corresponding Markov chain by using Algorithm 2 to compute the transition matrix. After that, we use it to evaluate the average number of operational servers $\overline{m}$ and the average number of waiting jobs $\overline{n}$ over a finite horizon h for the transient case. Or we solve the equilibrium equations of the steady state of the system. This steady state allows us to estimate $\overline{m}$ and $\overline{n}$. As a total cost we will consider:

$$c = c_N \times \overline{n} + c_M \times \overline{m}. \tag{3.6}$$

Table 3.1: Model and DTMC Parameters.

Parameters	**Description**
h finite or $+\infty$	duration of analysis
$\mathcal{M}$	total number of servers
b	buffer size
m	number of operational servers
$\overline{m}$	average number of operational servers (steady state)
n	number of waiting jobs
$\overline{n}$	average number of waiting jobs (steady state)
$\mathcal{H}_A$	histogram of job arrivals
$\mathcal{H}_D = \Delta_d$	histogram modeling the processing capacity of a server
c_M	energetic cost of one working server during one slot
c_N	cost of one waiting job in the buffer during one slot
E	set of all possible states

3.1.2 Application example

In this section we will apply the previous modelization on a very simple data center with settings shown in Table 3.2. For arrival jobs assume that, per slot, we have a probability of 0.2 to no arrival job, 0.5 to receive two arrival jobs, and 0.3 to receive five arrival jobs. In this case, arrivals are modeled by histogram $\mathcal{H}_A = (S_A, P_A)$ where $S_A = \{0, 2, 5\}$, $P_A(0) = 0.2$, $P_A(2) = 0.5$, and $P_A(5) = 0.3$.

Table 3.2: Settings used for testing thresholds policy based on Markov chain.

Parameters	**Value**	**Unit**	**Description**
$\mathcal{M}$	3	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	4	jobs	buffer size
$\mathcal{H}_A$	described above	jobs/slot	histogram of arrival jobs
$\mathcal{H}_D$	Δ_1	jobs/server/slot	processing capacity of a server
c_M	1	/server/slot	cost of energy needed by a server
c_N	1	/job/slot	cost of a waiting job
ρ	83%	percentage	load factor

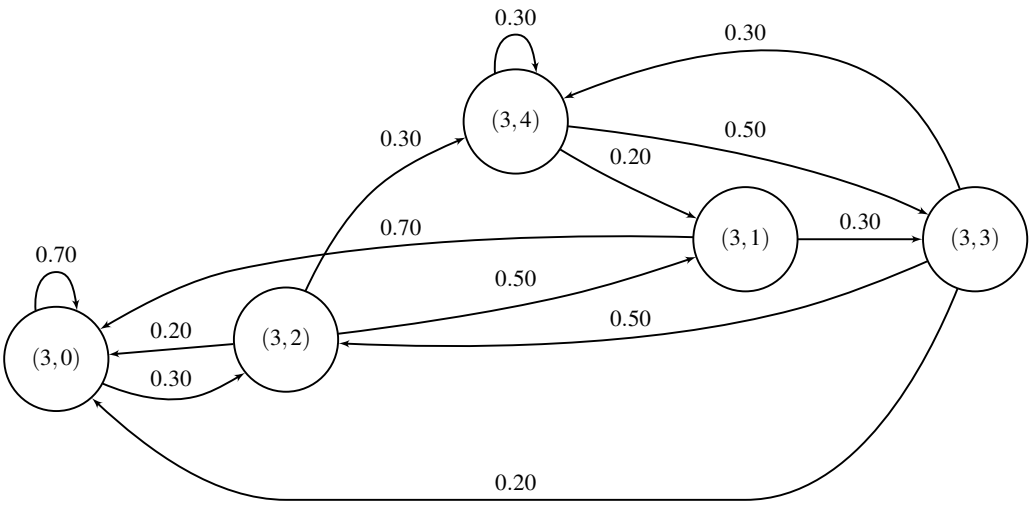
We can see from the analysis of the steady state of all possible Markov chains that the best strategy, according to the minimal total cost is to choose (σ, β) as $(2, 1)$. So the best policy in the long term is to keep two servers always on, and set the threshold of the buffer to one.

Table 3.3: Strategy comparison for thresholds policy based on Markov chain.

σ	β	$\overline{n}$	$\overline{m}$	cost = $c_N \times \overline{n} + c_M \times \overline{m}$
0	1	2.071	2.337	4.408
0	2	2.489	2.120	4.609
0	3	2.489	2.120	4.609
1	1	2.016	2.381	4.397
1	2	2.350	2.216	4.566
1	3	2.631	2.054	4.685
2	**1**	**1.578**	**2.572**	**4.150**
2	2	1.790	2.501	4.291
2	3	2.062	2.337	4.399
3	1	1.283	3.000	4.283
3	2	1.283	3.000	4.283
3	3	1.283	3.000	4.283

In the following three pages, we give the graph, the transition matrix, the equilibrium equations, the steady state, the average number of operational servers, the average number of waiting jobs, and the total cost of some pairs (σ, β).

Figure 3.2: DTMC for $\sigma = 3$ and $\beta = 3$



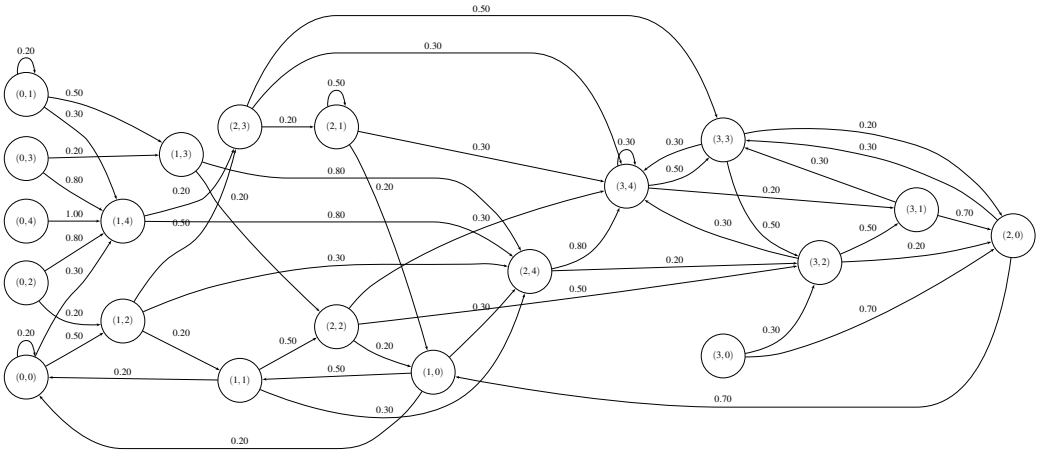
$$
\begin{array}{c}
 \\ (0,0) \\ (0,1) \\ (0,2) \\ (0,3) \\ (0,4) \\ (1,0) \\ (1,1) \\ (1,2) \\ (1,3) \\ (1,4) \\ (2,0) \\ (2,1) \\ (2,2) \\ (2,3) \\ (2,4) \\ (3,0) \\ (3,1) \\ (3,2) \\ (3,3) \\ (3,4)
\end{array}
\begin{array}{c}
\begin{array}{ccccccccccccccccccccc}
(0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (3,0) & (3,1) & (3,2) & (3,3) & (3,4)
\end{array} \\
\begin{pmatrix}
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.7 & - & 0.3 & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.7 & - & - & 0.3 & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & 0.5 & - & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & - & 0.5 & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & - & 0.5 & 0.3
\end{pmatrix}
\end{array}
$$

$$
\begin{cases}
0.7\pi_{15} + 0.7\pi_{16} + 0.2\pi_{17} + 0.2\pi_{18} = \pi_{15} \\
0.5\pi_{17} + 0.2\pi_{19} = \pi_{16} \\
0.3\pi_{15} + 0.5\pi_{18} = \pi_{17} \\
0.3\pi_{16} + 0.5\pi_{19} = \pi_{18} \\
0.3\pi_{17} + 0.3\pi_{18} + 0.3\pi_{19} = \pi_{19} \\
\displaystyle\sum_{i=0}^{19} \pi_i = 1
\end{cases}
$$

$$
\pi \simeq \begin{array}{c}
\begin{array}{ccccccccccccccccccccc}
 & (0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (3,0) & (3,1) & (3,2) & (3,3) & (3,4)
\end{array} \\
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.470 & 0.120 & 0.190 & 0.097 & 0.123
\end{pmatrix}
\end{array}
$$

$\overline{n} \simeq 1.283$ and $\overline{m} \simeq 3.000$.

Figure 3.3: DTMC for $\sigma = 0$ and $\beta = 1$



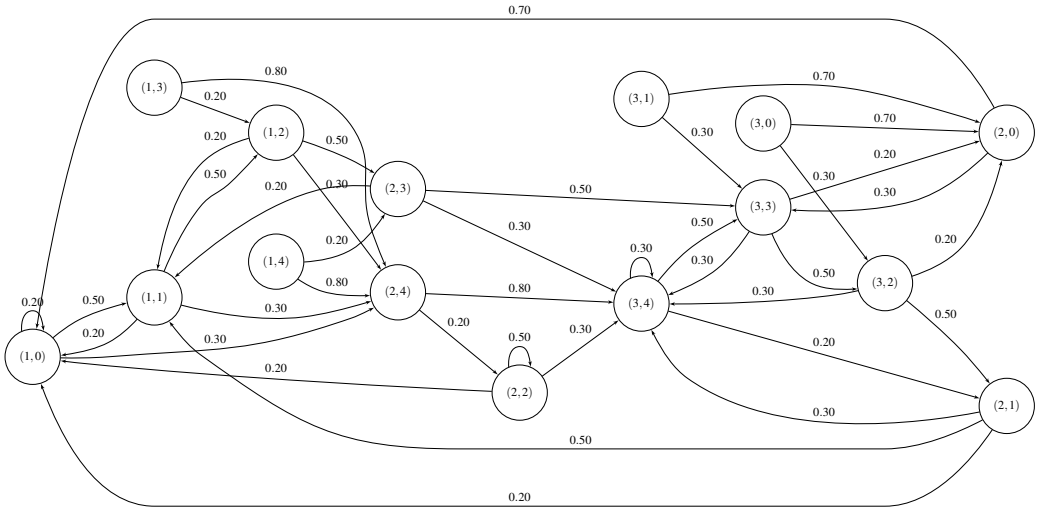
$$
\begin{array}{c}
\\
(0,0)\\(0,1)\\(0,2)\\(0,3)\\(0,4)\\(1,0)\\(1,1)\\(1,2)\\(1,3)\\(1,4)\\(2,0)\\(2,1)\\(2,2)\\(2,3)\\(2,4)\\(3,0)\\(3,1)\\(3,2)\\(3,3)\\(3,4)
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccccccccccc}
(0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (3,0) & (3,1) & (3,2) & (3,3) & (3,4)
\end{array}\\
\begin{pmatrix}
0.2 & - & - & - & - & - & - & 0.5 & - & 0.3 & - & - & - & - & - & - & - & - & - & - \\
- & 0.2 & - & - & - & - & - & - & 0.5 & 0.3 & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & 0.2 & - & 0.8 & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & 0.2 & 0.8 & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & 1.0 & - & - & - & - & - & - & - & - & - & - \\
0.2 & - & - & - & - & - & 0.5 & - & - & - & - & - & - & - & 0.3 & - & - & - & - & - \\
0.2 & - & - & - & - & - & - & - & - & - & - & - & 0.5 & - & 0.3 & - & - & - & - & - \\
- & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & 0.3 & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & 0.2 & - & 0.8 & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & 0.8 & - & - & - & - & - \\
- & - & - & - & - & 0.7 & - & - & - & - & - & - & - & - & - & - & - & - & 0.3 & - \\
- & - & - & - & - & 0.2 & - & - & - & - & - & 0.5 & - & - & - & - & - & - & - & 0.3 \\
- & - & - & - & - & 0.2 & - & - & - & - & - & - & - & - & - & - & - & 0.5 & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & - & 0.8 \\
- & - & - & - & - & - & - & - & - & - & 0.7 & - & - & - & - & - & - & 0.3 & - & - \\
- & - & - & - & - & - & - & - & - & - & 0.7 & - & - & - & - & - & - & - & 0.3 & - \\
- & - & - & - & - & - & - & - & - & - & 0.2 & - & - & - & - & - & 0.5 & - & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & - & 0.5 & 0.3
\end{pmatrix}
\end{array}
$$

$$
\begin{cases}
0.2\pi_0 + 0.2\pi_5 + 0.2\pi_6 = \pi_0 \\
0.2\pi_1 = \pi_1 \\
0.7\pi_{10} + 0.2\pi_{11} + 0.2\pi_{12} = \pi_5 \\
0.5\pi_5 + 0.2\pi_7 = \pi_6 \\
0.5\pi_0 + 0.2\pi_2 = \pi_7 \\
0.5\pi_1 + 0.2\pi_3 = \pi_8 \\
0.3\pi_0 + 0.3\pi_1 + 0.8\pi_2 + 0.8\pi_3 + 1.0\pi_4 = \pi_9 \\
0.7\pi_{15} + 0.7\pi_{16} + 0.2\pi_{17} + 0.2\pi_{18} = \pi_{10} \\
0.5\pi_{11} + 0.2\pi_{13} = \pi_{11} \\
0.5\pi_6 + 0.2\pi_8 = \pi_{12} \\
0.5\pi_7 + 0.2\pi_9 = \pi_{13} \\
0.3\pi_5 + 0.3\pi_6 + 0.3\pi_7 + 0.8\pi_8 + 0.8\pi_9 = \pi_{14} \\
0.5\pi_{17} + 0.2\pi_{19} = \pi_{16} \\
0.5\pi_{12} + 0.2\pi_{14} + 0.3\pi_{15} + 0.5\pi_{18} = \pi_{17} \\
0.3\pi_{10} + 0.5\pi_{13} + 0.3\pi_{16} + 0.5\pi_{19} = \pi_{18} \\
0.3\pi_{11} + 0.3\pi_{12} + 0.3\pi_{13} + 0.8\pi_{14} + 0.3\pi_{17} + 0.3\pi_{18} + 0.3\pi_{19} = \pi_{19} \\
\sum_{i=0}^{19} \pi_i = 1
\end{cases}
$$

$$
\pi \simeq \begin{array}{c}
\begin{array}{cccccccccccccccccccc}
(0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (3,0) & (3,1) & (3,2) & (3,3) & (3,4)
\end{array}\\
\begin{pmatrix}
0.035 & 0 & 0 & 0 & 0 & 0.092 & 0.049 & 0.018 & 0 & 0.011 & 0.123 & 0.004 & 0.025 & 0.011 & 0.056 & 0 & 0.095 & 0.109 & 0.171 & 0.201
\end{pmatrix}
\end{array}
$$

$\overline{n} \simeq 2.071$ and $\overline{m} \simeq 2.337$.

Figure 3.4: DTMC for $\sigma = 1$ et $\beta = 2$



$$
\begin{array}{c}
\\ (0,0)\\ (0,1)\\ (0,2)\\ (0,3)\\ (0,4)\\ (1,0)\\ (1,1)\\ (1,2)\\ (1,3)\\ (1,4)\\ (2,0)\\ (2,1)\\ (2,2)\\ (2,3)\\ (2,4)\\ (3,0)\\ (3,1)\\ (3,2)\\ (3,3)\\ (3,4)
\end{array}
\begin{array}{c}
\begin{array}{cccccccccccccccccccc}
(0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (3,0) & (3,1) & (3,2) & (3,3) & (3,4)
\end{array}\\
\begin{pmatrix}
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - & - \\
- & - & - & - & - & 0.2 & 0.5 & - & - & - & - & - & - & - & 0.3 & - & - & - & - & - \\
- & - & - & - & - & 0.2 & - & 0.5 & - & - & - & - & - & - & 0.3 & - & - & - & - & - \\
- & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & 0.3 & - & - & - & - & - \\
- & - & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.8 & - & - & - & - & - \\
- & - & - & - & - & - & - & - & - & - & - & - & - & 0.2 & 0.8 & - & - & - & - & - \\
- & - & - & - & - & 0.7 & - & - & - & - & - & - & - & - & - & - & - & - & 0.3 & - \\
- & - & - & - & - & 0.2 & 0.5 & - & - & - & - & - & - & - & - & - & - & - & - & 0.3 \\
- & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & - & - & - & - & - & - & 0.3 \\
- & - & - & - & - & - & 0.2 & - & - & - & - & - & - & - & - & - & - & - & 0.5 & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.8 \\
- & - & - & - & - & - & - & - & - & - & 0.7 & - & - & - & - & - & - & 0.3 & - & - \\
- & - & - & - & - & - & - & - & - & - & 0.7 & - & - & - & - & - & - & - & 0.3 & - \\
- & - & - & - & - & - & - & - & - & - & 0.2 & 0.5 & - & - & - & - & - & - & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & - & 0.3 \\
- & - & - & - & - & - & - & - & - & - & - & 0.2 & - & - & - & - & - & - & 0.5 & 0.3
\end{pmatrix}
\end{array}
$$

$$
\begin{cases}
0.2\pi_5 + 0.2\pi_6 + 0.7\pi_{10} + 0.2\pi_{11} + 0.2\pi_{12} = \pi_5 \\
0.5\pi_5 + 0.2\pi_7 + 0.5\pi_{11} + 0.2\pi_{13} = \pi_6 \\
0.5\pi_6 + 0.2\pi_8 = \pi_7 \\
0.7\pi_{15} + 0.7\pi_{16} + 0.2\pi_{17} + 0.2\pi_{18} = \pi_{10} \\
0.5\pi_{17} + 0.2\pi_{19} = \pi_{11} \\
0.5\pi_{12} + 0.2\pi_{14} = \pi_{12} \\
0.5\pi_7 + 0.2\pi_9 = \pi_{13} \\
0.3\pi_5 + 0.3\pi_6 + 0.3\pi_7 + 0.8\pi_8 + 0.8\pi_9 = \pi_{14} \\
0.3\pi_{15} + 0.5\pi_{18} = \pi_{17} \\
0.3\pi_{10} + 0.5\pi_{13} + 0.3\pi_{16} + 0.5\pi_{19} = \pi_{18} \\
0.3\pi_{11} + 0.3\pi_{12} + 0.3\pi_{13} + 0.8\pi_{14} + 0.3\pi_{17} + 0.3\pi_{18} + 0.3\pi_{19} = \pi_{19} \\
\sum_{i=0}^{19} \pi_i = 1
\end{cases}
$$

$$
\pi \simeq \begin{array}{c}
\begin{array}{cccccccccccccccccccc}
(0,0) & (0,1) & (0,2) & (0,3) & (0,4) & (1,0) & (1,1) & (1,2) & (1,3) & (1,4) & (2,0) & (2,1) & (2,2) & (2,3) & (2,4) & (3,0) & (3,1) & (3,2) & (3,3) & (3,4)
\end{array}\\
\begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0.096 & 0.108 & 0.054 & 0 & 0 & 0.045 & 0.087 & 0.031 & 0.027 & 0.077 & 0 & 0 & 0.075 & 0.151 & 0.249
\end{pmatrix}
\end{array}
$$

$\overline{n} \simeq 2.350$ and $\overline{m} \simeq 2.216$.

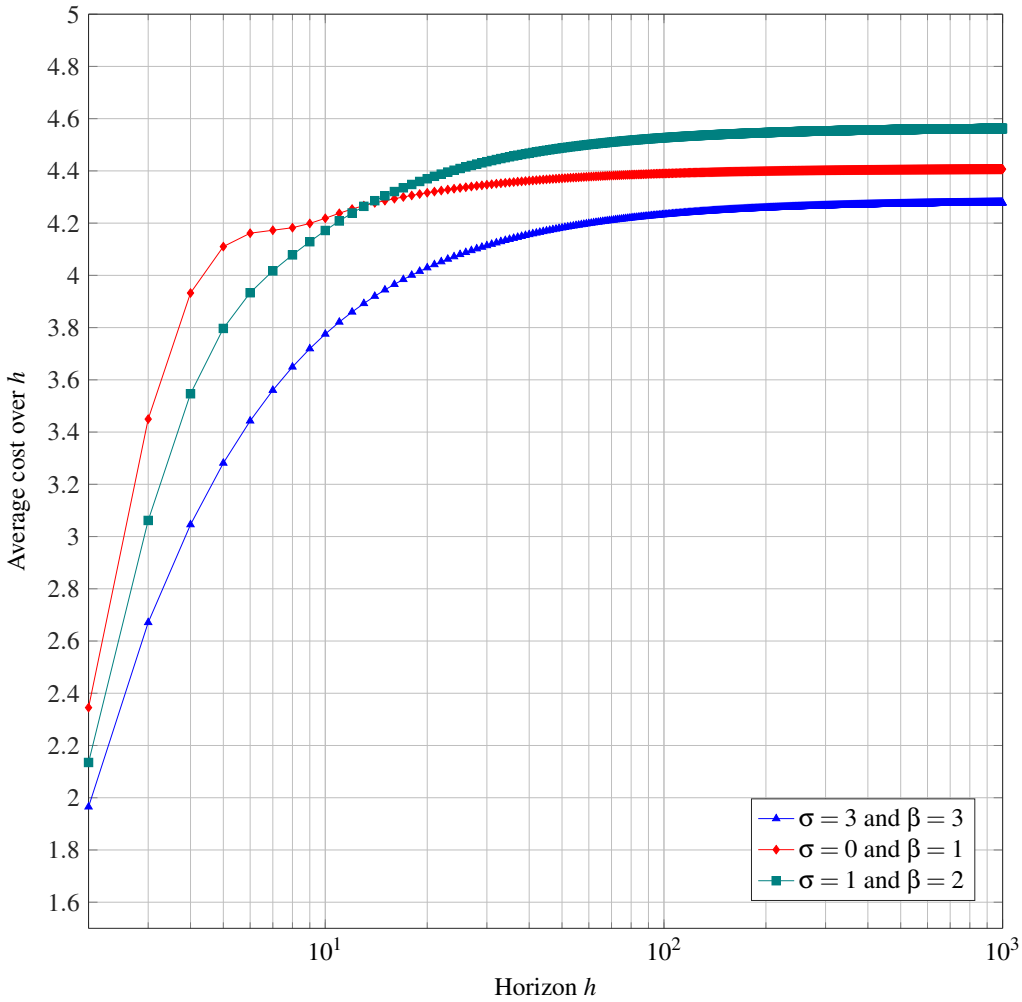


Figure 3.5: Comparison of average cost of different strategies for a finite horizon (transient behavior).

As said before at the end of the previous section (Section 3.1.1) for the transient behavior, we use the transition matrix to evaluate the average number of operational servers $\overline{m}$ and the average number of waiting jobs $\overline{n}$ over a finite horizon h. Figure 3.5 shows the evolution of the average cost over a finite period h for the three different strategies presented in the last pages. We can see that for instance, despite that the red strategy ($\sigma = 0$ and $\beta = 1$) is better than the green one ($\sigma = 1$ and $\beta = 2$) in the long term or for a big horizon, the green one is better for a period $h < 13$.

In this first part, we have shown how DTMC can be used to model and compute a sub-optimal policy for an energy/QoS optimization problem for a data center when the strategy mechanism is predefined as a threshold structure. The model

was restricted to:

1. identical servers (see Section 4.2 for heterogeneous servers),
2. constant service rate (see Section 4.2.4 for service rate modeled by general distribution),
3. no latency (see Section 4.3 for model with latency).

Those restrictions are not limitations of the model, we only choose them to limit the model to simplify this first part. We considered those restrictions in Chapter 4 where we model the problem with a more sophisticated Markov formalism. However the real limitation of the DTMC model is related principally to two points:

1. The need to consider a predefined strategy: the DTMC model is not able to compute the optimal policy, it computes the performance of only a sub-optimal policy related to the predefined strategy.
2. The need to build explicitly all DTMC's that model the predefined strategy: which leads to an explosion of state space and computation time.

In the next chapter we will give a second example of the analysis of a predefined policy when we will use the histogram operators instead of building explicitly all DTMC's.

3.2 Thresholds policy based on histogram operators

As a continuation of the previous section, we will present here a more sophisticated stochastic optimization system. In this second section we consider also a special mechanism. Thus, we show how *histogram operators* can be used to avoid generating explicitly a whole batch of DTMC's as done in last section. More precisely, this section introduces the use of histogram operators to model and solve the problem of saving the energy consumed in a data center when preserving it service performance.

3.2.1 Problem specification

Here we assume that the arrival process is modeled by the same probabilistic distribution for short period of time and changes between periods. This allows us to model for instance hourly or daily variations of the job arrivals. Suppose that the

analysis will be done over a whole period of time denoted h which is decomposed into p time intervals:

$$T_1 = [t_0..t_1[, T_2 = [t_1..t_2[, \cdots, T_k = [t_{k-1}..t_k[, \cdots, T_p = [t_{p-1}..t_p[\tag{3.7}$$

where the first instant t_0 equals 0, and the last one t_p equals h. So during these time intervals, the traffic is assumed to be stationary and we impose that the number of operational servers is unchanged during a time interval. Which means that we will be able to change the number of operational servers (by switching on or off some servers) only at the beginning of each time interval T_k. This assumption is related to our predefined optimization strategy that will be explained further in this section.

Thus, the number of jobs arriving to the data center per slot during time interval T_k is modeled by a histogram $\mathcal{H}_{A_k}$ where $P_{A_k}(i)$ gives the probability to have i arrival jobs per a slot. Note that we assume that arrivals of jobs are independent, and their distribution P_{A_k} may be obtained from real traces, empirical data, or incoming traffic measurements. In this manner, the queuing model is a batch arrival queue with constant services and finite capacity buffer b (buffer size). In this chapter we assume that a server needs lat_{on} unit of time to be switched on and lat_{off} unit of time to be switched off. We assume also that the switching-on leads instantaneously to an additional energetic cost c_{on} however the switching-off takes place without consuming any additional amount of energy.

The number of waiting jobs in the buffer is denoted by n and its distribution by $\mathcal{H}_N$. The number of operational servers is denoted by m. The number of rejected (lost) jobs is denoted by l and its distribution by $\mathcal{H}_L$. We assume that initially the number of operational servers and the number of waiting jobs are 0. The maximal number of servers that can be operational is $\mathcal{M}$.

Within time interval $T_k = [t_{k-1}..t_k[$, the histogram operators given in Section 1.3.2 can be used to inductively compute the evolution of distributions $\mathcal{H}_N$:

$$\mathcal{H}_{N(t)} \leftarrow MIN_b\big((\mathcal{H}_{N(t-1)} \oplus \mathcal{H}_{A_k}) - m\big) \tag{3.8}$$

where the arrival jobs are added to the system (buffer and free operational servers), a maximum of $d \times m$ jobs will be on servers to be processed, and the rest of jobs will be rejected (see Section 1.3.1 for the exact order of events). In fact, the distribution of the number of lost jobs is computed as:

$$\mathcal{H}_{L(t)} \leftarrow \big(\mathcal{H}_{N(t-1)} \oplus \mathcal{H}_{A_k} - (m+b)\big). \tag{3.9}$$

The energy consumption takes into account the state and the transitions of each server:

- c_{idle} the cost of energy consumed per slot per server when the server is operational but idle (operational but without proceeding a job),
- c_{busy} the cost of energy consumed per slot per server when the server is operational but busy (when proceeding a job).
- c_{on} energetic cost needed to switch on a server.
- $c_{off} = 0$ it is assumed that a non operational server does not consume energy.

If at slot t we have $m(t)$ operational servers, $\widetilde{m}(t) = \max\{0, m(t) - m(t+1)\}$ is the number of additional servers switched on. The expected cost of energy consumed by all operational servers during a slot t is:

$$energy(t) = \widetilde{m}(t) \times c_{on} + \sum_{i=0}^{b} P_{N(t)}(i)\big(c_{idle} \times (m(t) - i)^{+} + c_{busy} \times \min(i, m(t))\big) \tag{3.10}$$

where:

1. $\min(i, m(t))$ is the number of busy servers if the number of waiting jobs is i,
2. $(m(t) - i)^{+}$ is the number of idle servers if the number of waiting jobs is i,
3. $\widetilde{m}(t) \times c_{on}$ is the cost to switch on $\widetilde{m}(t)$ additional servers.

In order to simplify the model, the energy used to switch on server is assumed to be consumed instantaneously at the beginning of the slot where the decision is taken although the server becomes available only after lat_{on} slot due to the latency. The total cost of energy used is the sum of these quantities among the sample path.

As QoS metric, we take into account the number of rejected jobs which is modeled by a distribution of lost jobs for every time slot. Each rejected job causes a penalty that costs c_L. In fact the expected cost of QoS is computed as:

$$\sum_{i \in S_{L(t)}} P_{L(t)}(i) \times i \times c_L. \tag{3.11}$$

Additionally, we take into account the number of waiting job which is modeled by a distribution of waiting jobs for every time slot. Each waiting job causes a penalty that costs c_N. In fact the expected cost of QoS is computed as:

$$\sum_{i \in S_{N(t)}} P_{N(t)}(i) \times i \times c_N. \tag{3.12}$$

In fact the total expected cost of QoS can be estimated as:

$$QoS(t) = \sum_{i \in S_{L(t)}} P_{L(t)}(i) \times i \times c_L + \sum_{i \in S_{N(t)}} P_{N(t)}(i) \times i \times c_N. \tag{3.13}$$

Finally the total expected cost function can be estimated as:

$$cost(t) = energy(t) + QoS(t). \tag{3.14}$$

3.2.2 Optimization specification

In this section we consider a specific mechanism (not necessarily optimal) to minimize the energy consumption of the data center where keeping a good QoS. To do so, we will use a double thresholds strategy based on two integers: an up-threshold U and a down-threshold D.

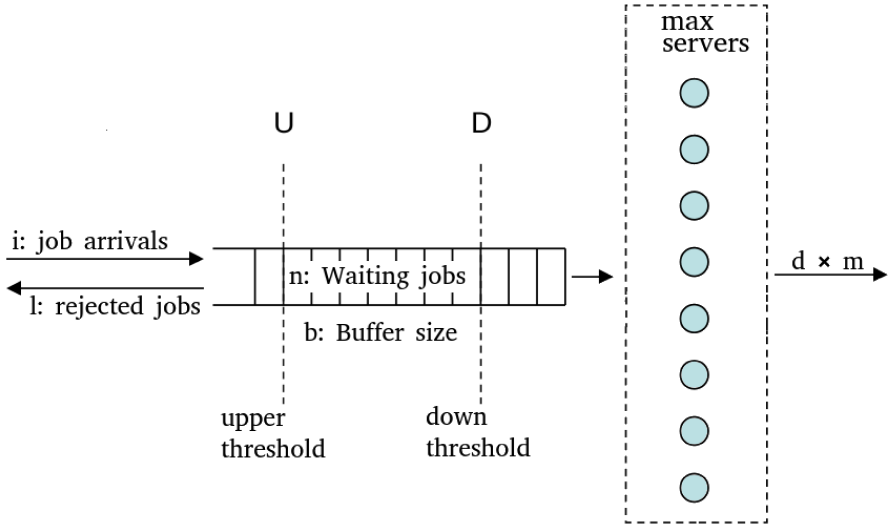


Figure 3.6: Optimization based on Up/Down thresholds mechanism

At the beginning of every time interval $T_k = [t_{k-1}..t_k[$ we will somehow observe the number of waiting jobs n and switch on more servers if n exceeds U, or turn off some servers if n is less than D.

Algorithm 3: Management algorithm for thresholds policy (special mechanism, in general sub-optimal and not necessarily optimal).

```
Data: n, m, M, U, D
Result: Update m
if n ≥ U and m < M then
    m ← m + 1;
else
    if n < D and m > 0 then
        m ← m − 1;
    end
end
return m
```

As we are dealing with numerical analysis based on histogram operators, we will base the action of running or stopping servers by observing the expectation of the number of waiting jobs. This expectation is compared with threshold values D and U, if larger than U, the number of operational servers is increased. If the expectation is smaller than D, the number of operational servers is decreased.

The goal is to find the best strategy by defining the pair (U,D) that minimizes at the same time the expected cost of energy and the expected cost of QoS. In other words minimizing a cost function that combines the cost of consuming energy and the cost of the degradation of the QoS.

3.2.2.1 Time interval analysis by coupling

During a time interval T_k, since the arrivals are i.i.d. and if the number of servers is constant and unchanged, the underlying model is a time-homogeneous DTMC taking values in a totally ordered state space (i.e. the buffer size). Thus, theoretical results related to stochastic monotonicity can be used [MS02, BF12, ASCTFP16] to improve the analysis.

Let $\mathcal{G} = \{0,1,2,\ldots,v \in \mathbb{N}\}$ be a state space endowed with $\leq$ a total order. Let $\mathcal{H}_X$ and $\mathcal{H}_Y$ be histograms with support in $\mathcal{G}$.

Definition 3.1. *Histogram $\mathcal{H}_X$ is said to be stochastically smaller than $\mathcal{H}_Y$ (denoted as $\mathcal{H}_X \leq_{st} \mathcal{H}_Y$) iff:*

$$\forall i,\, 1 \leq i,\ \sum_{j=i}^{v} P_X(j) \leq \sum_{j=i}^{v} P_Y(j). \tag{3.15}$$

Definition 3.2. *A DTMC $\{X_{(t)}, t \geq 0\}$ is stochastically monotone if*

$$\forall t:\ X_{(0)} \leq_{st} X_{(1)} \implies X_{(t)} \leq_{st} X_{(t+1)}.$$

Theorem 3.1. *During each time interval T_k, as arrival jobs are i.i.d. and the number of operational servers is not changing, the DTMC of our queuing model is stochastically monotone (see [ACFP13] for a proof).*

These theoretical results have many applications. Here, these results are used to prove the convergence of the transient distributions to the steady-state.

Let $\mathcal{H}_{N_{min}}$ and $\mathcal{H}_{N_{max}}$ be two histograms modeling the corresponding transient distributions of probability of the number of waiting jobs. $\mathcal{H}_{N_{min}}$ and $\mathcal{H}_{N_{max}}$ represent two realizations of the stochastic process of the queue size with two initial values equal respectively to the Dirac distribution in 0 and b. $\mathcal{H}_X^{(t)}$ denotes the distribution during slot t.

Corollary 3.1 ([ASCTFP16]). *For any time slot $t > 0$:*

$$\mathcal{H}_{N_{min}}^{(t)} \leq_{st} \mathcal{H}_{N}^{(t)} \leq_{st} \mathcal{H}_{N_{max}}^{(t)}. \tag{3.16}$$

Therefore the convergence can be detected by computing the norm of the difference between $\mathcal{H}_{N_{max}}^{(t)}$ and $\mathcal{H}_{N_{min}}^{(t)}$.

First note that in time interval $T_k = [t_{k-1}..t_k[$, the traffic is i.i.d. Thus, the transient distribution of a homogeneous Markov chain has to be analyzed during each time interval. The whole analysis consists in a numerical computation of the distribution at any time. By taking into account the monotonicity of the system and the coupling detection algorithm, unnecessary computations are avoided as stated in Corollary 3.1 and depicted in Figure 3.7.

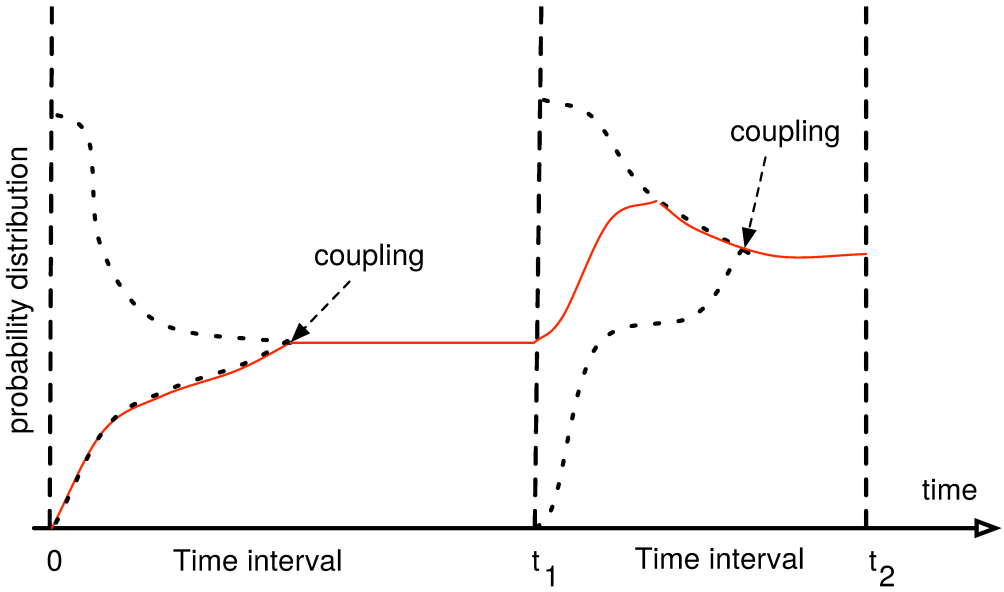


Figure 3.7: Coupling and stationarity detection: The bounds are plotted in dotted lines while the exact distribution is drawn as a plain red line.

When $\mathcal{H}_{N_{max}}^{(t)}$ and $\mathcal{H}_{N_{min}}^{(t)}$ become equal, they couple and they will stay equal for the remaining life of the process.

Two cases may occur: one may observe the convergence due to the coupling before the end of the time interval T_k or the boundary is reached before the occurrence of the coupling. If the first case, we jump to T_{k+1} as soon as we have detected

Algorithm 4: Computation with coupling detection over time interval $T_k = [t_{k-1}..t_k[$

Data: b, t_{k-1}, t_k, d, m 1
Result: $\mathcal{H}_N$ 2

$\mathcal{H}_{N_{min}} = \Delta_0$; 3
$\mathcal{H}_{N_{max}} = \Delta_b$; 4
foreach t *in* $T_k = [t_{k-1}..t_k[$ **do** 5
  $\mathcal{H}_{N_{min}} \leftarrow MIN_b\big((\mathcal{H}_{N_{min}} \oplus \mathcal{H}_{A_k}) - (d \times m)\big)$; 6
  $\mathcal{H}_N \leftarrow MIN_b\big((\mathcal{H}_N \oplus \mathcal{H}_{A_k}) - (d \times m)\big)$; 7
  $\mathcal{H}_{N_{max}} \leftarrow MIN_b\big((\mathcal{H}_{N_{max}} \oplus \mathcal{H}_{A_k}) - (d \times m)\big)$; 8
  if $\mathcal{H}_{N_{max}} \equiv \mathcal{H}_{N_{min}}$ **then** 9
    return $\mathcal{H}_N$ and jump to time interval T_{k+1} 10
    without any new computation; 11
  end 12
end 13
return $\mathcal{H}_N$ and jump to time interval T_{k+1}; 14

the coupling without any computation of the distribution. Indeed, as the two distributions have coupled, the steady-state distribution is reached thus it is not necessary to continue the numerical process. This numerical procedure has strong connections with the stationarity detection heuristic proposed by Sericola in [Ser99] for the efficient computation of reliability. Here, the stationarity is proved by the coupling while it was only numerically checked in [Ser99]. Note that we may use the infinite norm to check the equality in distributions mentioned in Instruction 9:

$$\mathcal{H}_{N_{max}} \equiv \mathcal{H}_{N_{min}} \iff \|\mathcal{H}_{N_{max}} - \mathcal{H}_{N_{min}}\| < \varepsilon \tag{3.17}$$

where ε is a small value enough close to zero. ε should be equal to zero but as we deal with numerical analysis we have to choose a value enough close to zero (10^{-6} for example).

3.2.2.2 Numerical analysis

Let $t_0 = 0$, $p \in \mathbb{N}$, and p periods $T_k = [t_{k-1}..t_k[$ for $k \in 1..p$. We take as input a sequence of instants $t_0 = 0, t_1, \ldots, t_p = h$ and the corresponding arrival traffic distribution $\mathcal{H}_{A_1}, \mathcal{H}_{A_2}, \ldots, \mathcal{H}_{A_p}$. $\mathcal{H}_{A_k}, k \in 1..p$ refers to the traffic during $T_k = [t_{k-1}..t_k[$. The data center is described by its parameters: b, c_{busy}, c_{idle}, c_{on}, l_1, l_2, $\mathcal{M}$. The modeler has to provide the QoS measure that will be monitored to control the number of servers (see Section 3.2.2).

Algorithm 5: Main function to evaluate the expected accumulated cost for a given U and D.

```
Data: p ∈ ℕ, Instants t_0 = 0, t_1, ..., t_p = h, distributions H_{A_1}, ..., H_{A_p}
Data: b, d, lat_on, lat_off, M, U, D ∈ ℕ, c_on, c_idle, c_busy ∈ ℝ^+
Result: overall expected accumulated cost C(h)
H_N ← Δ_0, latency_state ← "no_latency";
m ← 0, C ← 0, cpt ← 0;
foreach all time intervals T_k = [t_{k-1}..t_k[ do
    foreach t ∈ T_k do
        if latency_state = "no_latency" then
            n ← Σ_{i∈S_N} P_N(i) × i;
            if n ≥ U then
                C ← C + c_on;
                latency_state ← "switching_on_latency";
                cpt ← lat_on;
                /* increase m by 1 after lat_on slots */
            end
            if n < D then
                C ← C + c_off;
                latency_state ← "switching_off_latency";
                cpt ← lat_off;
                /* decrease m by 1 after lat_off slots */
            end
        end
        if latency_state = "switching_on_latency" then
            cpt ← cpt − 1;
            if cpt = 0 then
                m ← m + 1, latency_state ← "no_latency";
            end
        end
        if latency_state = "switching_off_latency" then
            cpt ← cpt − 1;
            if cpt = 0 then
                m ← m − 1, latency_state ← "no_latency";
            end
        end
        H_L ← (H_N ⊕ H_{A_k} − (d × m + b));
        H_N ← MIN_b((H_N ⊕ H_{A_k}) − (d × m));
        Loss ← Σ_{i∈S_L} P_L(i) × i × c_L;
        Waiting ← Σ_{i∈S_N} P_N(i) × i × c_N;
        QoS ← Waiting + Loss;
        Energy ← Σ_{i=0}^{b} P_N(i)(c_idle × (m − i)^+ + c_busy × min(i, m));
        c ← QoS + Energy;
        C ← C + c;
    end
end
return C;
```

The main function is called for each period T_k. Under the i.i.d.-ness of the traffic specified by distribution $\mathcal{H}_{A_k}$, the function computes the chosen QoS measure for every time slot. For a given slot, if the QoS measure exceeds the upper threshold U then one additional server will be switched on after lat_{on} time units and if the QoS measure becomes less than the lower threshold D then one server will be switched off after a latency of lat_{off} time units. Finally we will return the overall cost (accumulated cost) that combines the energy cost and the QoS cost:

$$C(h) = \sum_{t=0}^{h} cost(t) \tag{3.18}$$

For every possible threshold couple (U,D) such that $0 \leq D \leq U \leq b$ where D/U is the lower/upper threshold, depending on this cost outputs, we choose the best thresholds.

Theorem 3.2. *The computation-time to evaluate the strategy by Algorithm 5 is in* $O(h \times b^2 \times q \times \log(q))$ *where* $q = b + \max(S_A)$.

Proof. To find the best policy we have to use Algorithm 5 to compute the overall expected cost over the period h for every possible threshold couple (U,D) such that $0 \leq D \leq U \leq b$. As U and D are between 0 and b, we need $\frac{(b+1)\times(b+2)}{2}$ to test all possible couples (U,D). Otherwise, for each slot we need to compute the distribution of the number of waiting jobs in the buffer by computing a convolution between $\mathcal{H}_A$ and $\mathcal{H}_N$. This task needs at most $b \times |S_A|$ operations, however, as being said before in Remark 1.1, by using the fast Fourier transform the convolution can be performed in a reduced time of $O(q \times \log(q))$ with $q = \max(S_A) + \max(S_N)$ where $\max(S_N) = b$. □

Theorem 3.3. *The space complexity of algorithm 5 is in* $O(b^2)$.

Proof. At each step of the calculation, Algorithm 5 needs to keep track of the evolution of the distribution of the number of waiting jobs in the buffer. As the buffer is finite, the number of waiting jobs is bounded above by b and the algorithm can use only an array of length $O(b)$ to store the distribution probability of waiting jobs and a space memory of $O(q\ \log(q))$ to compute the convolution, with $q = \max(S_A) + b$. Algorithm 5 needs also to keep track of the evolution of the distribution of the number of lost jobs. The number of lost jobs is bounded above by $\max(S_A)$ and the algorithm can use only an array of length $O(\max(S_A))$ to store the distribution probability of lost jobs. Otherwise we need to store the computed cost for each possible threshold couple (U,D) which needs a matrix of size $O(b^2)$.

Finally we need a total space of $q\ \log(q) + b + \max(S_A) + b^2$ which is in $O(b^2)$ if $\max(S_A)$ and b are of the same order of magnitude. □

3.2.3 Case study

In this section we present in details an analysis to find the best thresholds U and D to control the number of servers with respect to the average queue size. Notice that we have done other analyses in the following chapters. Those analyses were done to compare the result of the methodology "Thresholds policy based on histogram operators" to the other methods (see Table 4.9 and Figure 4.16 of Section 4.1.5.5 for three additional experimentations, and Sections 6.1.1, 6.1.2, 6.1.3, 6.1.4, 6.1.5, and 6.1.6 of Chapter 6 for six more experimentations).

For the experimentation of this section we use the open *clusterdata-2011-2* trace [Wil11, RWH11], and we focus on the part that contains the job events corresponding to the requests destined to a specific Google data center for the whole month of May 2011. The job events are organized as a table of eight attributes; we only use the column *timestamps* that refers to the arrival times of requests expressed in microsecond. This timestamps allows us to model for instance hourly, daily, or weekly variations of the job arrivals. In this case study timestamps are used to detect whether the job was received during nighttime or daytime in order to consider an analysis on a period of one day divided into nighttime and daytime. In fact we need to sample the trace and construct two empirical distributions (histograms) one corresponding to arrivals during daytime and the other during nighttime.

It is important to assume the i.i.d.-ness of the arrivals. However to avoid to just keep that property as an assumption, we suggested to construct an arrival model which is i.i.d. In fact we do the following: the traffic trace is sampled with a sampling period for which both the nighttime and daytime arrival jobs can be considered as i.i.d. So, we sample, literally, the data with several sampling periods between a small sampling period (one second) and a big one (five minutes), then we applied the turning point test [Ken73] for each sampling periods in order to test the i.i.d.-ness of the sampled data. We found that a sampling period of 136 second has a $p - value$ bigger than 0.05 for both period, thus at confidence level 0.95 we accept the null hypothesis, namely, the sampled data is i.i.d. Thus, we consider frames of 136 second to sample the trace and construct two empirical distributions (histograms) one corresponding to arrivals during daytime and the other during nighttime (see Figure 3.8).

These distributions have different statistical properties reflecting the fluctuation of traffic between day and night (see Figure 3.9). For instance, we observe an average of 46 jobs per minute during daytime against an average of 50 jobs per

minute during nighttime (see Figure 3.9).

The daytime distribution $\mathcal{H}_{A_1}$ was considered between $t_0 = 0$ and $t_1 = 720$ minute, and the nighttime distribution $\mathcal{H}_{A_2}$ was considered between $t_1 = 720$ and $t_2 = 1440$ minute. The methodology showed in this chapter was used to analyze one day under the configuration of the Table 3.4.

Table 3.4: Settings of the study case

p	b	d	lat_{on}	lat_{off}	$\mathcal{M}$	c_{idle}	c_{busy}	c_{on}	c_L	c_N
2	100	1	1	0	100	13	20	30	30	20

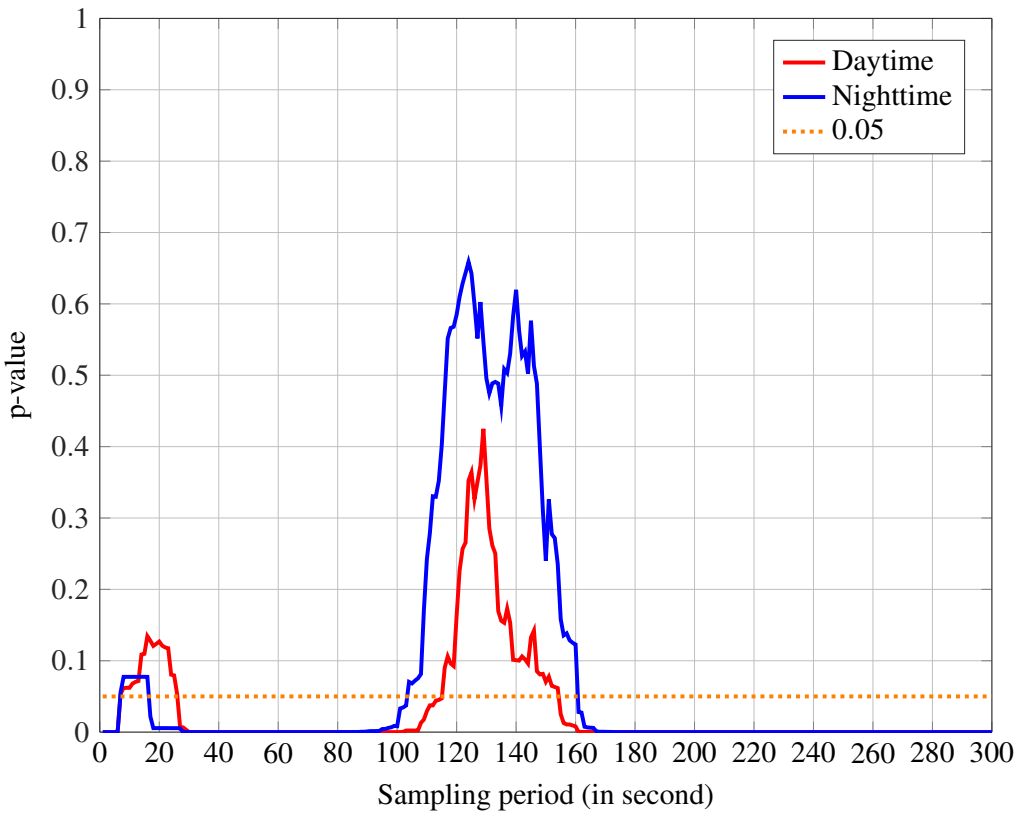


Figure 3.8: Google trace arrivals and i.i.d.-ness test for nighttime and daytime: The turning point test for the nighttime arrivals is accepted, at confidence level 95%, for a sampling period in $[104, 166]$ second. However, for the daytime it is accepted for a sampling period in $[116, 154]$ second. In fact the arrival jobs are i.i.d. for nighttime and daytime for any sampling period in $[104, 166] \cap [116, 154]$.

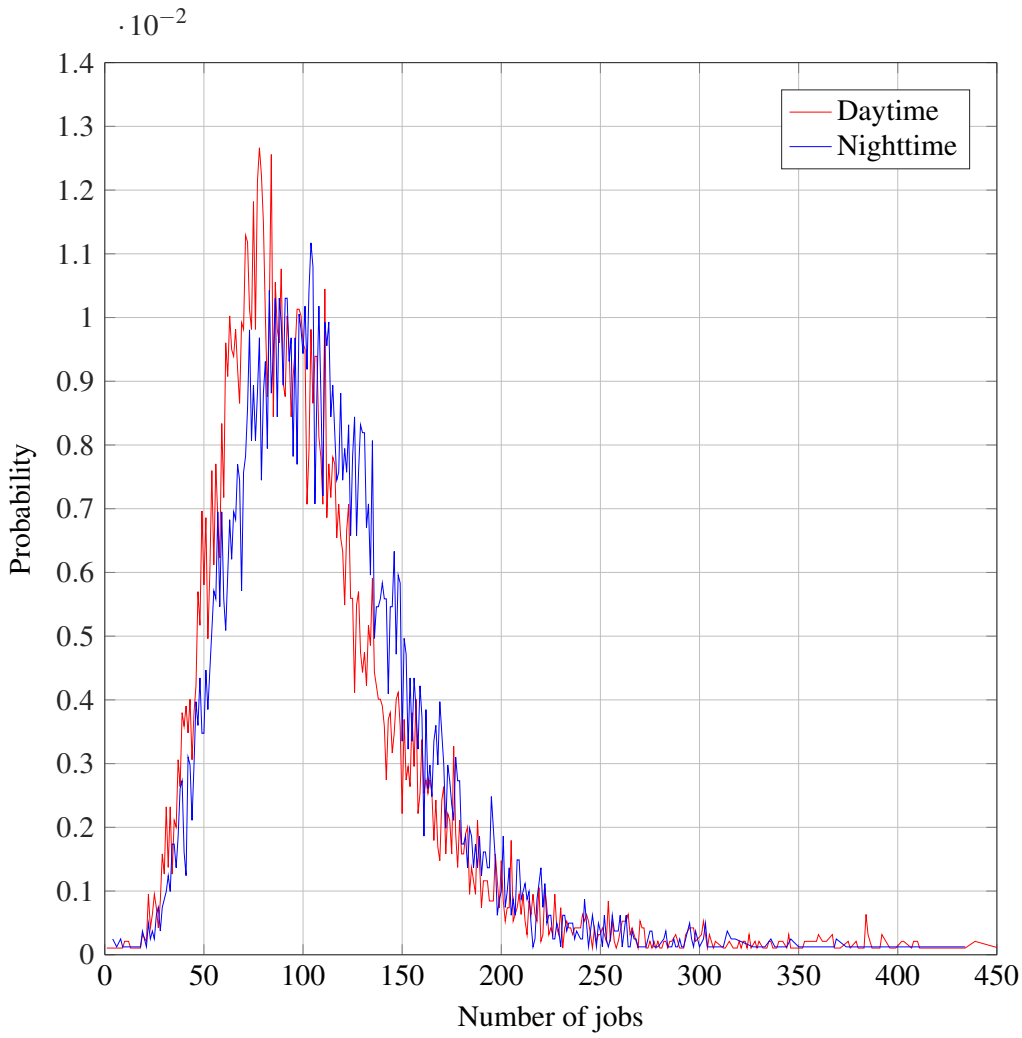


Figure 3.9: Discrete distributions for nighttime and daytime obtained by sampling real data with an appropriate slot length to preserve the i.i.d.-ness property.

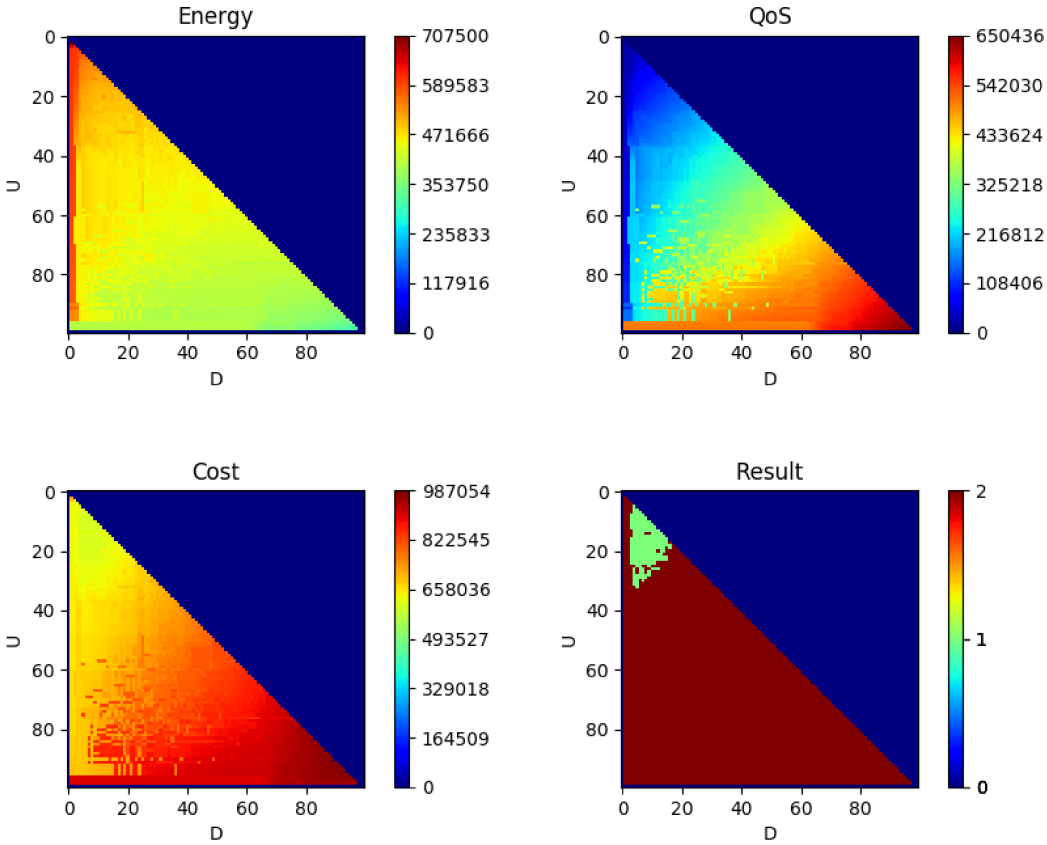


Figure 3.10: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible $(D > U)$.

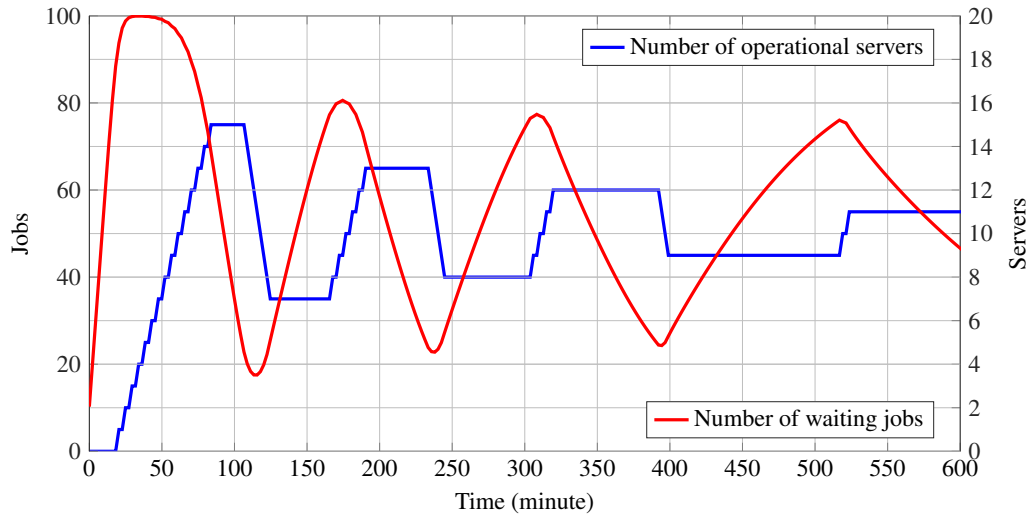


Figure 3.11: The evolution of the number of operational servers (blue) and the average queue size (red) during the first 600 minutes of the first period T_1.

Figure 3.12: The evolution of the number of operational servers (blue) and the average of rejected jobs (red) during the first 600 minutes of the first period T_1.

It can be seen from Figure 3.10 that when the energy consumption is high, the QoS is low and vice versa. The modeler can combine the QoS and the energy in a global index to find the best thresholds (see left graphic of Figure 3.10). Otherwise, Figure 3.12 shows the evolution of the average number of waiting jobs and the number of operational servers over time when using the best optimal strategy computed by Algorithm 5. We can observe that when the number of waiting jobs is low (less then D) the strategy tends to reduce the number of operational servers, this action will cause a diminution of the served jobs and an augmentation of the waiting jobs until that the waiting jobs exceed U. In this case the strategy will tend to switch on more and more operational servers to process more jobs until that the number of waiting jobs returns ton a reasonable quantity between U and D.

The methodology adopted in this section was built in a tool based on the numerical simulation of Equations (3.8) and (3.9) associated with a controller defined in Section 3.2.2. Algorithm 5 shows the core control function of the tool in a simplified way. Note that the tool is written in C/C++ with a parallelized code in which the coupling detection algorithm is implemented. Notice that the analysis was done on an *Intel(R) Core(TM) i7-4800MQ CPU @2.70GHz* computer in 70 minutes, and the use of coupling detection reduced the computation time to 49 minutes (42% faster). Notice that the value of ε was set to 10^{-6}.

3.3 Conclusion

In this chapter we showed how we can model a specific mechanism to model and to solve the problem of reducing the energy consumed in a data center when keeping a good QoS based on a predefined classical monotone policy which is the thresholds strategy. The first part was devoted to a solution based on DTMC. The second part shows how we used the histogram operators to address a similar problem. The two methods are both based on a predefined management mechanism which is the threshold strategy, both are monotone (because they are based on a threshold strategy structure, see Definition 4.2), both lead to a sub optimal strategy (in Section 4.1.5.5 of the following chapter we will show the sub optimality of the thresholds strategy). However, we notice that if we implement the two methods with the same considerations (same nature of thresholds, same arrivals model) the second method is more efficient computationally than a solution based on DTMC, because it uses less space memory to compute the strategy as it does not build explicitly all the needed DTMC's. Instead of that, it computes only the distributions of lost and waiting jobs and keeps tracing of there evolution by using histogram operators. Additionally this second method uses coupling (stationarity detection algorithm) to make the computation faster.

Chapter 4

Markov Decision Process

AFTER introducing, in the previous chapter, some specific mechanisms to sub-optimizing the cost induced by the energy consumption and the QoS, in this chapter we will use a *Markov Decision Process* and we will give details on how we can apply this concept in order to find the *optimal* strategy that ensures at the same time a low energy consumption and a high performance. In fact, to compute this optimal policy we first formulate the optimization problem by a discrete-time MDP. So, we focus on the modeling of the Dynamic Power Management by a stochastic model in the case of a data center with homogeneous servers. We do that principally to study structural properties of the optimal policy like the hysteresis, monotony, and isotony. Secondly, as data centers present a non negligible level of server heterogeneity in energy consumption and service rate, we generalize the homogeneous model to a heterogeneous one. Finally, as the switching-on (resp. switching-off) of a data center server is not instantaneous and needs additional units of time to transit from a sleeping/stopping mode to a ready-working mode, we extend the model in order to include this temporal latency of the servers. Along all the chapter, arrival jobs and service rates are specified with histograms that may be obtained from real traces, empirical data, or incoming traffic measurements.

4.1 MDP for homogeneous data center

In this section we show first how MDP can be used to model and solve our optimization problem to ensure a reasonable energy consumption with a good performance. After that we examine the space complexity of the model and the time complexity of the algorithm used to find the optimal strategy. Then we inspect more closely some structural properties of the optimal policy such as monotony

and the possibility to write the optimal policy as a double threshold structure, and how those properties are effected by the variation of the system parameters. Finally, MDP is applied on and tested for several system with arrivals obtained from real Google traffic traces to show the practical limits of the use of MDP.

4.1.1 Problem specification

Here we deal with a discrete-time Model. Let *DC* be a data center composed of $\mathcal{M}$ identical servers. *DC* receives jobs requesting the offered service. The queuing model is a batch arrival queue with finite capacity buffer b (buffer size). As said before we model arrival jobs and service rate with histograms. Thus, the number of jobs arriving to the data center during a slot is modeled by a histogram $\mathcal{H}_A$ where $P_A(i)$ gives the probability to have i arrival jobs per slot. Similarly, during one time unit, the number of jobs that can be served by a server is modeled by a histogram $\mathcal{H}_D$ where $P_D(d)$ gives the probability to process d jobs.

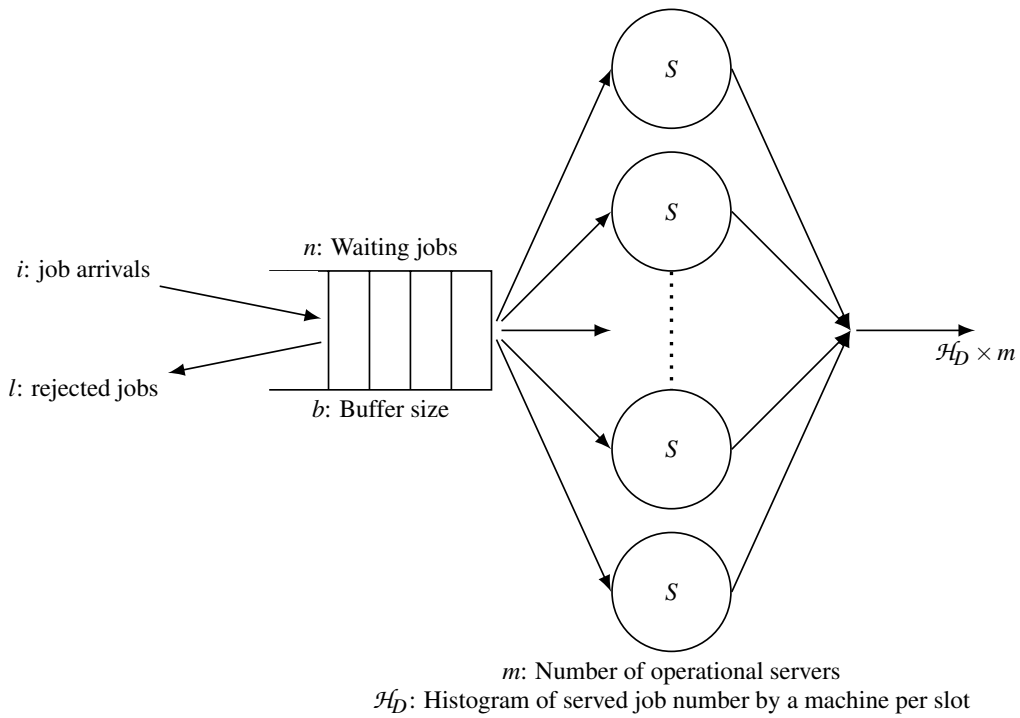


Figure 4.1: Illustration of the queuing model.

The number of waiting jobs in the buffer is denoted by n. The number of operational servers is denoted by m. The number of rejected (lost) jobs is denoted by l. We assume that initially the number of operational servers, the number of

waiting jobs, and the number of rejected jobs are 0. The maximal number of servers that can be operational is $\mathcal{M}$. The number of waiting jobs n can be computed by induction where the arrival jobs are added to the system (buffer and free operational servers), a maximum of $\sum_{k=1}^{\mathcal{M}} d_k$ jobs will be on servers to be processed, and the rest of jobs will be rejected (see Section 1.3.1 for the exact order of events). For a number of i arrival jobs, and if each server k processes at most d_k jobs per slot, we will have:

$$\begin{cases} l & \leftarrow & \max\{0, n+i-\sum_{k=1}^{\mathcal{M}} d_k - b\} \\ n & \leftarrow & \min\{b, \max\{0, n+i-\sum_{k=1}^{\mathcal{M}} d_k\}\}. \end{cases} \tag{4.1}$$

It is assumed that the input arrivals are i.i.d. and under these assumptions, the model of the queue is a time-homogeneous Discrete Time Markov.

4.1.1.1 Energy and Performance metric

The energy consumption takes into account the number of operational servers. Each server consumes some units of energy per slot when a server is operational and it costs an average of $c_M \in \mathbb{R}^+$ monetary units. A server may consume a very low amount of energy when it is turned off. However, during the latency period a server may consume an additional amount of energy that costs an average of $c_{On} \in \mathbb{R}^+$ monetary units which is the energetic cost needed to switch a server on. Additionally, we consider that a server switches-on immediately. The total energy consumed is the sum of all units of energy consumed among a specific period. The QoS metric takes into account the number of waiting and lost jobs. Each waiting job costs $c_N \in \mathbb{R}^+$ monetary unit per slot, and a rejected job costs $c_L \in \mathbb{R}^+$ monetary unit.

4.1.1.2 Objective function

For a given period of time, a dynamic power management system consists in doing at each slot an action (turn on or turn-off a specific number of servers) in order to adapt the number of operational servers to incoming job changes. The optimal strategy consists in finding the best sequence of actions to minimize the overall cost (accumulated cost) (energetic cost plus performance cost). The cost generated for each slot is a linear combination of energetic and QoS costs called objective function:

$$n \times c_N + l \times c_L + m \times c_M + \max\{0, m' - m\} \times c_{On} \tag{4.2}$$

where n is the number of waiting jobs, l is the number of lost jobs, m is the current number of operational servers, and m' is the number of operational servers after doing an action.

The problem we have to consider is to find a trade-off between the performance (i.e. waiting and loss jobs) and the energy consumption (i.e. number of operational servers). However, as the number of servers changes with time, the system becomes more complex to analyze than a system with servers operational all the time.

4.1.2 Optimization approach

In order to find the optimal strategy and then analyze the performance and the energy consumption of the data center under this optimal strategy, we will use the concept of *Markov Decision Process* to formulate our optimization problem.

4.1.2.1 Modelization

Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ be an MDP where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition probability, and $\mathcal{C}$ the immediate cost of each action. Let $\mathcal{H}_A = (S_A, P_A)$ be the histogram used to model the arrival of jobs. The state of the system is defined by the couple $(m,(n,l))$ where m is the number of operational servers, n is the number of waiting jobs, and l is the number of lost jobs. Indeed the state space $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{(m,(n,l)) \mid m \in [0..\mathcal{M}] \text{ and } n \in [0..b] \text{ and } l \in [0..\max(S_A)]\}. \tag{4.3}$$

Let $\mathcal{H}_{D^j}$ be the histogram defined as:

$$\mathcal{H}_{D^j} = \underbrace{\mathcal{H}_D \oplus \mathcal{H}_D \oplus \cdots \oplus \mathcal{H}_D}_{j \text{ times}}, \tag{4.4}$$

which is the distribution of the number of served jobs from j servers during one slot. In fact $P_{D^j}(d)$ gives the probability that j servers serve together d jobs.

At the beginning of each slot, and based on the current state of the system, an action $\alpha_j \in \mathcal{A}$ will be made to determine how many servers will be operational during the current slot. In fact the action space $\mathcal{A}$ is defined as $\mathcal{A} = \{\alpha_j \mid 0 \leq j \leq \mathcal{M}\}$, where action α_j consists in keeping exactly j operational servers during the current slot. We have a probability of $\mathcal{P}_{ss'}^{\alpha_j}$ to move from state $s = (m,(n,l))$ to $s' = (j,(n',l'))$ under action α_j. This probability is defined as:

$$\mathcal{P}_{ss'}^{\alpha_j} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_A \text{ and each } d \in S_{D^j} \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-d\}\} \\ l' & = \max\{0, n+i-d-b\} \end{array}\right.} P_A(i) \times P_{D^j}(d). \tag{4.5}$$

If we do the assumption that all the servers process the same number of jobs during the same slot then the probability will be equal to:

$$\mathcal{P}_{ss'}^{\alpha_j} = \sum_{\begin{cases} \text{for} & \text{each } i \in S_A \text{ and each } d \in S_D \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-d\times j\}\} \\ l' & = \max\{0, n+i-d\times j-b\} \end{cases}} P_A(i) \times P_D(d). \tag{4.6}$$

Consequently moving from state $s = (m,(n,l))$ to $s' = (j,(n',l'))$ under action α_j induces immediately a cost $\mathcal{C}_s^{\alpha_j}$ defined as:

$$\mathcal{C}_s^{\alpha_j} = j \times c_M + \max\{0, j-m\} \times c_{On} + n \times c_N + l \times c_L. \tag{4.7}$$

The immediate cost $\mathcal{C}_s^{\alpha_j}$, includes four parts:

1. The first part is $j \times c_M$, where c_M is the cost of energy consumption of one working server per slot and j is the number of working servers during the current slot. This part presents the total cost of energy consumed by the operational servers during the current slot.

2. The second part is $\max\{0, j-m\} \times c_{On}$, where c_{On} is the energetic cost of switching-on one server from stopping mode to working mode and $\max\{0, j-m\}$ is the number of servers switched-on at the beginning of the slot. This part presents the total cost of energy used to switch-on servers at the beginning of the current slot.

3. The third part is $n \times c_N$, where c_N is the cost of keeping one job in the buffer during one slot and n is the number of waiting jobs. This part presents the total cost of maintaining waiting jobs in the buffer during the current slot.

4. The last part is $l \times c_L$, where c_L is the cost of losing one job and l is the number of lost jobs. This part presents the total cost of losing jobs during the current slot.

Finally Table 4.1 summarizes parameters used in our model and our MDP formulation.

Remark 4.1. Notice that adding l in the state increases the number of states, however, as we will use the model checker PRISM to do the experimentation, we add directly l into the state of the system in order to be able to catch the cost induced by losing some jobs.

Table 4.1: Model and MDP Parameters.

Parameters	**Description**
h	duration of analysis
$\mathcal{M}$	total number of servers
b	buffer size
m	number of operational servers
n	number of waiting jobs
l	number of rejected jobs
$\mathcal{H}_A$	histogram of job arrivals
$\mathcal{H}_D$	histogram of service jobs rate
c_{On}	energetic cost of switching-on one server
c_M	energetic cost of one working server during one slot
c_N	cost of one waiting job in the buffer during one slot
c_L	cost of one lost job during one slot
$\mathcal{S}$	set of all possible states
$\mathcal{A}$	set of all possible actions
$s = (m,(n,l))$	system state
$s_0 = (0,(0,0))$	starting state
α_j	action to keep exactly j operational servers
$\mathcal{P}_{ss'}^{\alpha_j}$	probability of transition from s to s' under action α_j
$\mathcal{C}_s^{\alpha_j}$	immediate cost from s under action α_j

4.1.2.2 Space Complexity

In this section we will evaluate the space complexity of the data structure behind our MDP formulation.

Theorem 4.1. *If each state is represented by the couple $(m,(n,l))$ then, the number of states of the MDP is in $O(\mathcal{M} \times b \times \max(S_A))$. Otherwise, if each state is represented by the couple (m,n), without considering lost l then, the number of states of the MDP is in $O(\mathcal{M} \times b)$.*

Proof. Every state of the MDP includes three element:

1. the number of operational servers which is between 0 and $\mathcal{M}$,
2. the number of waiting jobs in the buffer which is bounded above by b, and
3. the number of rejected jobs which can be at most equals to the maximum number of arrival jobs given by $\max(S_A)$.

So, $|\mathcal{S}|$ is bounded above by $(\mathcal{M}+1) \times (b+1) \times (\max(S_A)+1)$ or by $(\mathcal{M}+1) \times (b+1)$ depending whether we consider or not lost l. □

Theorem 4.2. *If each state is represented by the couple $(m,(n,l))$ then, the number of transitions of the MDP is in $O(\mathcal{M}^2 \times b \times |S_A| \times \max(S_A))$. Otherwise, if each state is represented by the couple (m,n) without considering lost l then, the number of transitions of the MDP is in $O(\mathcal{M}^2 \times b \times |S_A|)$.*

Proof. From each state of the MDP we have at most $(\mathcal{M}+1)$ actions, and each action leads to a number of transitions equals to $|S_A|$ (one transition for each bin in the support of the arrival distribution). In fact, as the number of states was already evaluated in Theorem 4.1, we deduce that the number of transitions is bounded above by $\mathcal{M} \times b \times \max(S_A) \times (\mathcal{M}+1) \times |S_A|$ or by $\mathcal{M} \times b \times (\mathcal{M}+1) \times |S_A|$ depending whether we consider or not lost l. □

Remark 4.2. As shown previously in Theorems 4.1 and 4.2, the size (size can be seen as the number of states or the number of transitions) of the MDP is more important when we consider the number of rejected jobs l. Nevertheless, the spatial complexity of both models (considering or not l) still important (see experimental results in Annex C).

4.1.2.3 Solver algorithm

As we formulate our optimization problem as an MDP, an action consists in turning-on each unit of time a specific number of servers and turning-off the rest of the

servers. The optimal strategy is the best sequence of actions to be done in order to minimize the overall cost (accumulated cost) during a finite period of time called *horizon* and noted h. The value function $V : \mathcal{S} \times [0..h] \rightarrow \mathbb{R}^+$ has as objective minimizing the expected sum of costs over time:

$$V_t(s) = \min_{\pi} \mathbb{E}\left[\sum_{k=h-t+1}^{h} \mathcal{C}_{s_k}^{\pi_k(s_k)}\right]. \tag{4.8}$$

The value function can be expressed recursively as:

$$V_t(s) = \min_{\alpha_j}\left\{\mathcal{C}_s^{\alpha_j} + \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{\alpha_j} V_{t-1}(s')\right\}, \tag{4.9}$$

where α_j is the action taken by the system, and $\mathcal{P}_{ss'}^{\alpha_j}$ is the transition probability from state s to state s'. In this case the optimal policy for each state s is:

$$\pi_t^*(s) \in \operatorname*{argmin}_{\alpha_j \in \mathcal{A}}\left\{\mathcal{C}_s^{\alpha_j} + \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{\alpha_j} V_{t-1}(s')\right\}. \tag{4.10}$$

In order to find the optimal policy associated to each state in S, Algorithm 6, which is a value iteration algorithm, is applied.

Algorithm 6: Value iteration algorithm for MDP.

```
Data: M, C, P, S, horizon h
Result: (V_h, π*_h)
foreach s ∈ S do
    V_0(s) = 0
end
for k ← 1 to h do
    foreach s ∈ S do
        (m, n, l) ← s
        for a_j ← a_0 to a_M do
            C_s^{α_j} ← j × c_M + max{0, j − m} × c_On + n × c_N + l × c_L
            Q_k(s, a_j) ← C_s^{α_j} + Σ_{s'} P_{ss'}^{α_j} V_{k−1}(s')
        end
        π*_k(s) ← argmin_a Q_k(s, a)
        V_k(s) ← Q_k(s, π*_k(s))
    end
end
return (V_h, π*_h)
```

This algorithm minimizes the cost of the current slot plus the expected cost of all future slots. The costs of future slots are updated by using Bellman's backwards equations [Bel57, Put94, Ber95] as shown in the Algorithm. Where $V(s)$ is the value of state s, $\pi^*(s)$ is the optimal policy for state s, and $Q(s,a)$ is the value of taking action a in state s.

In the following of this section we will evaluate the time and space complexity of the value iteration algorithm 6 that computes the optimal strategy.

Theorem 4.3. *If each state is represented by the couple $(m,(n,l))$ then, the time complexity of Algorithm 6 is in $O(h \times \mathcal{M}^2 \times b \times |S_A| \times \max(S_A))$. Otherwise, if each state is represented by the couple (m,n), without considering lost l then, the time complexity of Algorithm 6 is in $O(h \times \mathcal{M}^2 \times b \times |S_A|)$.*

Proof. To compute the summation of the line 11 of Algorithm 6 we need to take into account all transitions from state s to any state s' under action α_j. This number of transitions is bounded above by $|S_A|$, in fact the time complexity of Algorithm 6 is in $O(h \times |\mathcal{S}| \times |\mathcal{A}| \times |S_A|)$: h iterations for the second loop at line 6, $|\mathcal{S}|$ iterations for the third loop at line 7, and $|\mathcal{A}|$ iterations for the last loop at line 9. In fact, the time complexity is in $O(h \times \mathcal{M}^2 \times b \times |S_A| \times \max(S_A))$ or $O(h \times \mathcal{M}^2 \times b \times |S_A|)$ depending whether we consider or not lost l. □

Theorem 4.4. *If each state is represented by the couple $(m,(n,l))$ then, the space complexity of Algorithm 6 is in $O(h \times \mathcal{M} \times b \times \max(S_A))$. Otherwise, if each state is represented by the couple (m,n), without considering lost l then, the space complexity of Algorithm 6 is in $O(h \times \mathcal{M} \times b)$.*

Proof. The value iteration algorithm use the V matrix to store intermediate values of the computation and also to be able to retrace the optimal strategy that induce the minimal cost. This matrix is of a size of $h \times |\mathcal{S}|$ which equals $O(h \times \mathcal{M} \times b \times \max(S_A))$ or $O(h \times \mathcal{M} \times b)$ depending whether we consider or not lost l. □

4.1.3 Example

To illustrate our formalization Let us show an MDP for a very simple data center:

- One server which means that $\mathcal{M} = 1$.
- Buffer size equals one which means that $b = 1$.
- Job arrivals are modeled as the following: assume that, per slot, the probability to receive one arrival job is 0.59, and 0.41 to receive no arrival jobs. In this case, arrivals are modeled by histogram $\mathcal{H}_A = (S_A, P_A)$ where $S_A = \{0,1\}$, $P_A(0) = 0.41$, and $P_A(1) = 0.59$.

- We set service rate of each machine as a histogram with only one bin: $P_D(1) = 1$.

So, MDP$= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ where:

$$\begin{cases} \mathcal{S} & = \{(0,(0,0)),(0,(1,0)),(0,(1,1)),(1,(0,0)),(1,(1,0)),(1,(1,1))\} \\ \mathcal{A} & = \{\alpha_0, \alpha_1\}. \end{cases}$$

And $\mathcal{P}$ can be deduced from the graph of Figure 4.2. For instance, state $s_3 = (1,(0,0))$ means that only one server is turned on, and no jobs are waiting in the buffer nor jobs are lost. Action α_0 switches-off the server and α_1 switches-on the server.

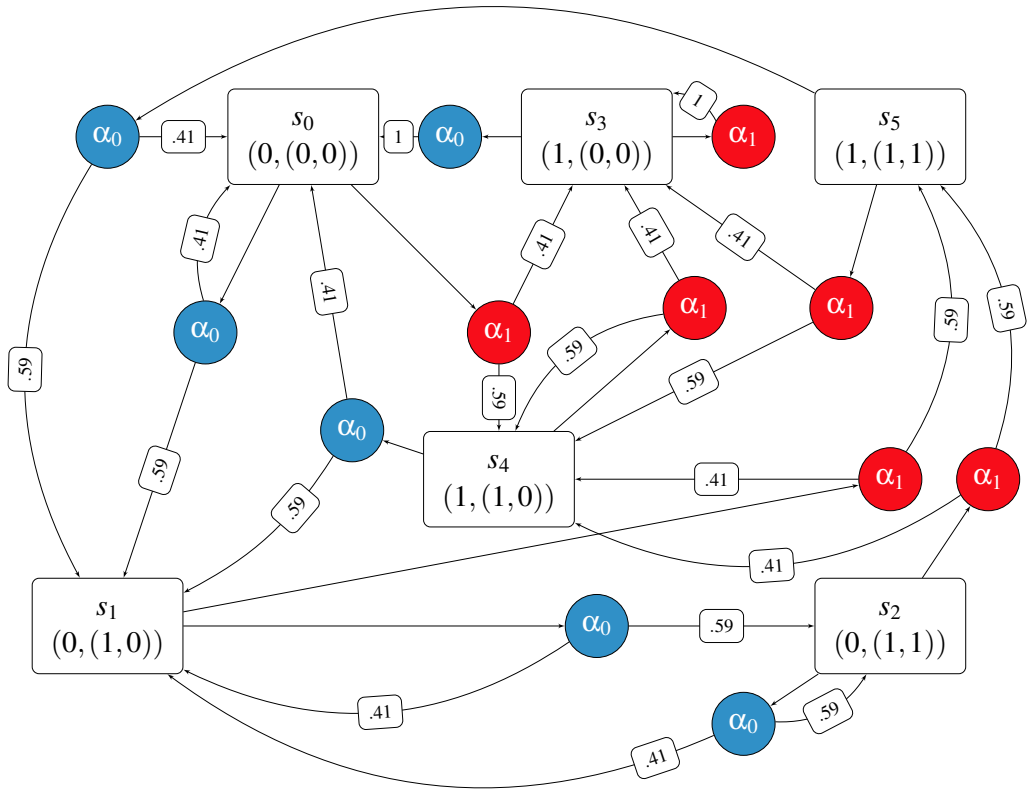


Figure 4.2: MDP example for a data center with two servers.

4.1.4 Optimal Strategy Structure

As shown previously, the size of the MDP is important, and the computation of the optimal policy can be hard even impossible for a big data center. In fact, it

is essential to analyze the structural properties of the optimal policy to make the computation efficient. In the following we will be interested in some properties of the optimal policy, and we show in particular that the double-threshold structure property does not hold for our homogeneous data center model.

The following definitions are extensions of the hysteretic, monotony, and isotony properties for our model based on previous work done by Topkis [Top78], Hipp and Holzbaur [HH88], Lu and Serfozo [LS84], Plum [Plu91], and Serfozo [Ser79].

Definition 4.1. *A policy* π *is called* **hysteretic** *if* $\forall t \in [0..h]$, $\exists m \in [0..\mathcal{M}]$, $\exists \alpha_j \in \mathcal{A}$ *such that* $\pi_t(m,(n,l)) = \alpha_j \implies \pi_t(j,(n,l))) = \alpha_j$.

Theorem 4.5. *The optimal policy (4.10) is hysteretic.*

Proof. For each slot t, Let us define function $f_t(m,j) : [0..\mathcal{M}] \times [0..\mathcal{M}] \to \mathbb{R}^+$ as the cost for switching the number of operational servers from m to j. And function $w_t(m,(n,l)) : \mathcal{S} \to \mathbb{R}^+$ represents the expected future cost starting with n waiting jobs in the buffer, losing l jobs, serving with m operational servers during one slot, and then following an optimal policy. So, our optimal policy (4.10) can be formulated as:

$$\pi_t^*(s) \in \underset{\alpha_j \in \mathcal{A}}{\operatorname{argmin}}\{f_t(m,j) + w_t(j,(n,l))\}, \tag{4.11}$$

$$\textit{where} \begin{cases} w_t(j,(n,l)) & = j \times c_M + n \times c_N + l \times c_L + \sum_{s'} \mathcal{P}_{ss'}^{\alpha_j} V_t(s') \\ f_t(m,j) & = \max\{0, j-m\} \times c_{On}. \end{cases} \tag{4.12}$$

According to Theorem 1 of [HH88], if the function f_t satisfies the following condition:

$$\begin{cases} \forall m \in [0..\mathcal{M}] & : f_t(m,m) = 0 \\ \forall m,p,q \in [0..\mathcal{M}] & : f_t(m,q) \leq f_t(m,p) + f_t(p,q) \end{cases} \tag{4.13}$$

then the optimal policy π^* is a hysteretic policy. Thus, to prove that our optimal policy is hysteretic we need just to prove that f_t satisfies conditions (4.13). We have $f_t(m,m) = \max\{0, m-m\} \times c_{On} = 0$ which implies the first condition of (4.13), and the following summarizes all possible cases for the second condition of (4.13):

1. if $m \geq q$ we have $f_t(m,q) = 0$ and as f_t is positive then $\forall p \in [0..\mathcal{M}]$: $f_t(m,p) + f_t(p,q) \geq 0 = f_t(m,q) \implies f_t(m,q) \leq f_t(m,p) + f_t(p,q)$.

2. if $m \leq p \leq q$ we have $f_t(m,p) + f_t(p,q) = (p-m) \times c_{On} + (q-p) \times c_{On} = (q-m) \times c_{On} = f_t(m,q) \implies f_t(m,q) \leq f_t(m,p) + f_t(p,q)$.

3. if $m \leq q \leq p$ we have $f_t(m,p) + f_t(p,q) = (p-m) \times c_{On} + 0 = (p-m) \times c_{On} \geq (q-m) \times c_{On} = f_t(m,q) \implies f_t(m,q) \leq f_t(m,p) + f_t(p,q)$.

4. if $p \leq m \leq q$ we have $f_t(m,p) + f_t(p,q) = 0 + (q-p) \times c_{On} = (q-p) \times c_{On} \geq (q-m) \times c_{On} = f_t(m,q) \implies f_t(m,q) \leq f_t(m,p) + f_t(p,q)$.

In conclusion, each condition of (4.13) holds and our optimal policy is hysteretic. □

A dynamic power management model is monotone if one can show that its optimal strategy is structured in pairs of thresholds. Thus, if an optimal strategy is shown to be monotonous then:

1. in practice, the implementation of the optimal strategy will be easy,
2. finding the optimal strategy can be done while avoiding the use of heavy algorithms that explore all possible strategies like value iteration. In other words, we need to explore only the set of strategies that are structured in pairs of thresholds.

Definition 4.2. *A policy π is called* **monotone** *if π is hysteretic and, $\forall t \in [0..h]$ there exist a set of integer thresholds $\{D_0^t \leq U_0^t, D_1^t \leq U_1^t, \ldots, D_{\mathcal{M}}^t \leq U_{\mathcal{M}}^t\}$ such that for every $s = (m,(n,l))$ we have:*

$$\pi_t(m,(n,l)) = \begin{cases} \pi_t(m-1,(n,l)) & \text{if} \quad n < D_m^t & \text{for } m > 0 \\ \alpha_m & \text{if} \quad D_m^t \leq n \leq U_m^t & \text{for } 0 \leq m \leq \mathcal{M} \\ \pi_t(m+1,(n,l)) & \text{if} \quad n > U_m^t & \text{for } m < \mathcal{M}. \end{cases} \tag{4.14}$$

Additionally, if for each $t \in [0..h]$, $D_0^t \leq D_1^t \leq \cdots \leq D_{\mathcal{M}}^t$ and $U_0^t \leq U_1^t \leq \cdots \leq U_{\mathcal{M}}^t$, then, the policy π is called **isotone**.

Notice that the policies presented in Chapter 3 are necessarily monotone because they are based on a double threshold structures by definition and by construction.

4.1.4.1 Monotone policy example

Before showing that in the general case, our model is not monotone, Let us give an example which is monotone. Assume that, per slot, we have a probability of 0.5 to receive one arrival job, and 0.5 to receive no arrival jobs. In this case, arrivals are specified by histogram $\mathcal{H}_A = (S_A, P_A)$ where $S_A = \{0,1\}$, $P_A(0) = 0.5$, and $P_A(1) = 0.5$. We set $b = 10$, $\mathcal{M} = 2$, $c_M = 15$, $c_N = 1$, $c_{On} = 10$, $c_L = 0$. The service rate is modeled by a histogram with only one bin such as $P_D(1) = 1$, and $h = 20$. Under those parameters, solving the optimality equation (4.9) leads to the following optimal policy for $t = 10$:

Observed state		Action
$(m=0,n=0)$	$\rightarrow$	α_0
$(m=0,n=1)$	$\rightarrow$	α_0
$(m=0,n=2)$	$\rightarrow$	α_1
$(m=0,n=3)$	$\rightarrow$	α_1
$(m=0,n=4)$	$\rightarrow$	α_1
$(m=0,n=5)$	$\rightarrow$	α_1
$(m=0,n=6)$	$\rightarrow$	α_2
$(m=0,n=7)$	$\rightarrow$	α_2
$(m=0,n=8)$	$\rightarrow$	α_2
$(m=0,n=9)$	$\rightarrow$	α_2
$(m=0,n=10)$	$\rightarrow$	α_2

Observed state		Action
$(m=1,n=0)$	$\rightarrow$	α_0
$(m=1,n=1)$	$\rightarrow$	α_1
$(m=1,n=2)$	$\rightarrow$	α_1
$(m=1,n=3)$	$\rightarrow$	α_1
$(m=1,n=4)$	$\rightarrow$	α_1
$(m=1,n=5)$	$\rightarrow$	α_1
$(m=1,n=6)$	$\rightarrow$	α_2
$(m=1,n=7)$	$\rightarrow$	α_2
$(m=1,n=8)$	$\rightarrow$	α_2
$(m=1,n=9)$	$\rightarrow$	α_2
$(m=1,n=10)$	$\rightarrow$	α_2

Observed state		Action
$(m=2,n=0)$	$\rightarrow$	α_0
$(m=2,n=1)$	$\rightarrow$	α_1
$(m=2,n=2)$	$\rightarrow$	α_1
$(m=2,n=3)$	$\rightarrow$	α_2
$(m=2,n=4)$	$\rightarrow$	α_2
$(m=2,n=5)$	$\rightarrow$	α_2
$(m=2,n=6)$	$\rightarrow$	α_2
$(m=2,n=7)$	$\rightarrow$	α_2
$(m=2,n=8)$	$\rightarrow$	α_2
$(m=2,n=9)$	$\rightarrow$	α_2
$(m=2,n=10)$	$\rightarrow$	α_2

Server Number	Lower threshold	Upper threshold
$m=0$	$D_0^{10}=0$	$U_0^{10}=1$
$m=1$	$D_1^{10}=1$	$U_1^{10}=5$
$m=2$	$D_2^{10}=3$	$U_2^{10}=10$

It is clear that lower and upper thresholds of the optimal policy for all possible values of $m=0,1,2$ are consistent with the Definition 4.2 of monotony. In fact the optimal policy can be written as the following double threshold structure:

$\pi_{10}((0,(n,l)))=$	$\pi_{10}((1,(n,l)))=$	$\pi_{10}((2,(n,l)))=$
$\begin{cases} \alpha_0 & \text{if} \quad 0\le n\le 1 \\ \pi_{10}((1,(n,l))) & \text{if} \quad n>1 \end{cases}$	$\begin{cases} \pi_{10}((0,(n,l))) & \text{if} \quad n<1 \\ \alpha_1 & \text{if} \quad 1\le n\le 5 \\ \pi_{10}((2,(n,l))) & \text{if} \quad n>5 \end{cases}$	$\begin{cases} \pi_{10}((1,(n,l))) & \text{if} \quad n<3 \\ \alpha_2 & \text{if} \quad 3\le n\le 10 \end{cases}$

4.1.4.2 Optimal policy and Monotony

In this section we will show that in general our model is not monotone. However for the same model, depending on specific values of the parameters of the system the resulting optimal policy can be monotone or not. The following theorem announces that monotony of the optimal policy of our MDP model does not hold.

Theorem 4.6. *The optimal policy (4.10) is not monotone.*

Proof. In this proof we give a counterexample that shows that monotony of the optimal policy (4.10) does not hold in general. Let us model the arrival by the following histogram: Assume that, per slot, we have a probability of 0.59 to receive one arrival job, and 0.41 to receive no arrival jobs. In this case, arrivals are modeled by histogram $\mathcal{H}_A=(S_A,P_A)$ where $S_A=\{0,1\}$, $P_A(0)=0.41$, and $P_A(1)=0.59$. We set $b=5$, $\mathcal{M}=5$, $c_M=9$, $c_N=8$, $c_{On}=7$. The service rate is modeled by a

histogram with only one bin such as $P_D(1) = 1$, and $h = 7$. In order to simplify the counter example[1], we set $c_L = 0$ so we do not need to consider rejected jobs in the MDP model. Under those parameters, solving the optimality equation (4.9) leads to the following optimal policy for $t = 5$:

Observed state		Action
$(m=0,n=0)$	$\rightarrow$	α_1
$(m=0,n=1)$	$\rightarrow$	α_2
$(m=0,n=2)$	$\rightarrow$	α_2
$(m=0,n=3)$	$\rightarrow$	α_3
$(m=0,n=4)$	$\rightarrow$	α_4
$(m=0,n=5)$	$\rightarrow$	α_5

Observed state		Action
$(m=1,n=0)$	$\rightarrow$	α_1
$(m=1,n=1)$	$\rightarrow$	α_1
$(m=1,n=2)$	$\rightarrow$	α_2
$(m=1,n=3)$	$\rightarrow$	α_2
$(m=1,n=4)$	$\rightarrow$	α_3
$(m=1,n=5)$	$\rightarrow$	α_4

Observed state		Action
$(m=2,n=0)$	$\rightarrow$	α_1
$(m=2,n=1)$	$\rightarrow$	α_1
$(m=2,n=2)$	$\rightarrow$	α_1
$(m=2,n=3)$	$\rightarrow$	α_2
$(m=2,n=4)$	$\rightarrow$	α_2
$(m=2,n=5)$	$\rightarrow$	α_3

Observed state		Action
$(m=3,n=0)$	$\rightarrow$	α_1
$(m=3,n=1)$	$\rightarrow$	α_1
$(m=3,n=2)$	$\rightarrow$	α_1
$(m=3,n=3)$	$\rightarrow$	α_1
$(m=3,n=4)$	$\rightarrow$	α_2
$(m=3,n=5)$	$\rightarrow$	α_3

Observed state		Action
$(m=4,n=0)$	$\rightarrow$	α_1
$(m=4,n=1)$	$\rightarrow$	α_1
$(m=4,n=2)$	$\rightarrow$	α_1
$(m=4,n=3)$	$\rightarrow$	α_1
$(m=4,n=4)$	$\rightarrow$	α_1
$(m=4,n=5)$	$\rightarrow$	α_2

Observed state		Action
$(m=5,n=0)$	$\rightarrow$	α_1
$(m=5,n=1)$	$\rightarrow$	α_1
$(m=5,n=2)$	$\rightarrow$	α_1
$(m=5,n=3)$	$\rightarrow$	α_1
$(m=5,n=4)$	$\rightarrow$	α_1
$(m=5,n=5)$	$\rightarrow$	α_1

For $m = 2$, it is clear that lower and upper thresholds of the optimal policy can not be other than $D_2^5 = 3$ and $U_2^5 = 4$, because if they exist:

$$\begin{cases} D_m^5 &= \min\{n \mid \pi_5^*(m,(n,l)) = \alpha_m\} \\ U_m^5 &= \max\{n \mid \pi_5^*(m,(n,l)) = \alpha_m\}. \end{cases}$$

In fact, if the policy is monotone we must have:
π_5 monotone $\implies \forall n < D_2^5 :\ \pi_5^*(m=2,(n,l)) = \pi_5^*(m=1,(n,l))$
However this last implication does not hold because $\pi_5^*(m=2,(n=2,l)) = \alpha_1$ however $\pi_5^*(m=1,(n=2,l)) = \alpha_2$. □

Corollary 4.1. *The optimal policy (4.10) is not isotone.*

Proof. From Definition of isotony, monotony is a necessary condition for isotony. However form Theorem 4.6, we deduce that the optimal policy (4.10) is not isotone. □

Corollary 4.2. *At least one of the function w and f defined in Equation (4.12) is not sub-modular.*

[1]The counter example holds even for some positive value of c_L.

Proof. According to Corollary 1 of [HH88], if condition (4.13) holds and functions w and f are both sub-modularity then the policy is monotony. However Theorem 4.6 claims that the optimal policy is not monotone, in fact we deduce that necessarily, at least, one of the function w and f is not sub-modular (because condition (4.13) is verified). □

So we showed that the model is in general non-monotone. But it could be monotone for some values of the system parameters. In order to have the whole picture of the monotony/non-monotony of the optimal strategy, we have to:

1. either, study analytically the modularity of functions f and w of relations defined in the proof of Theorem 4.5,
2. or, compute numerically for each configuration the optimal policy and check whether it is monotone or note based on Definition 4.2,

As the arrival and the service processes are modeled by histograms and not by an analytical formulation, it is not obvious to use the analytical analysis to study the modularity of functions f and w. However as we are already in the context of numerical analysis, in the following we produce an entire image of the monotony depending on the variation of the system parameters by analyzing numerically the system, producing the optimal policy then checking whether it is monotone or not.

Figure 4.3 gives the monotony/non-monotony of the optimal strategy when service rate is modeled by $\mathcal{H}_D = \Delta_1$, arrival jobs by histogram of Example 1.4, $h = 200$, and $t = 100$:

1. The top left graphic of the figure gives results when varying values of c_M and c_N in interval $[0, 100]$ and $b = 8$, $\mathcal{M} = 5$, $c_L = 0$, $c_{on} = 15$.
2. The top right graphic of the figure gives results when varying values of c_M and b in interval $[0, 100]$ and $\mathcal{M} = 5$, $c_L = 0$, $c_N = 20$, $c_{on} = 15$.
3. The bottom left graphic of the figure gives results when varying values of c_M and $\mathcal{M}$ in interval $[0, 100]$ and $b = 8$, $c_L = 0$, $c_N = 20$, $c_{on} = 15$.
4. The bottom right graphic of the figure gives results when varying values of c_M and c_{on} in interval $[0, 100]$ and $b = 8$, $\mathcal{M} = 5$, $c_L = 0$, $c_N = 20$.

The patterns observed in those graphics confirm the complexity of studying the modularity of the monotony in an analytical way.

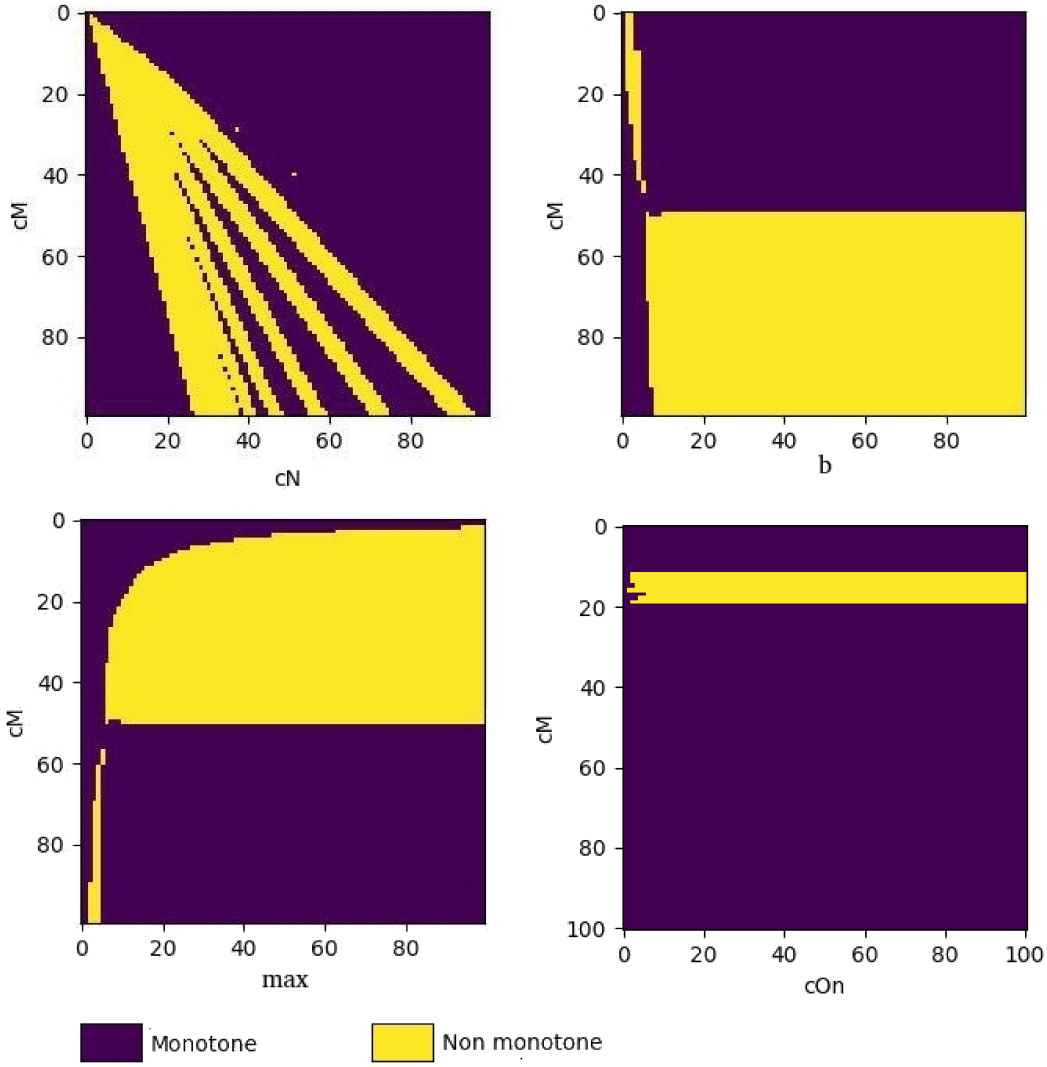


Figure 4.3: Monotony/non-monotony of the optimal strategy when varying, b, $\mathcal{M}$, c_M, c_N, and c_{on}.

4.1.5 Experimental results

To model arrivals and service rate, this work uses real traffic traces based on the open *clusterdata-2011-2* trace [Wil11, RWH11]. Before starting our experiments, Let us describe the most important probabilistic properties that can be expressed for an MDP under PRISM.

4.1.5.1 MDP under PRISM

For MDP modeling, PRISM can be used to check some expected reachability properties. The two principal properties are: computing the minimum or maximum expected accumulated reward/cost to reach some set of states in an MDP, i.e. PRISM properties of the form:

Listing 4.1: Expected reachability properties in PRISM

```
1  Rmin=? [...] or Rmax=? [...]
```

This properties are expressed under *Probabilistic Computation Tree Logic* (PCTL) which is an extension of computation tree logic (CTL) that allows the specification of probabilistic quantification properties [HJ94].

Based on the minimization or the maximization of the reward (or the cost), PRISM using PCTL allows a wide range of quantitative measures of the system. This is can be grouped into three principal types of properties.

Instantaneous properties Given a period of h units of time, PRISM can be used to find the best policy that minimizes/maximizes the reward over the given period then returns the instantaneous reward at slot h:

Listing 4.2: Instantaneous properties

```
1  Rmin=? [I=h] or Rmax=? [I=h]
```

This can be used to estimate for example the maximum expected queue size at noon (12 o'clock).

Cumulative properties Given a period of h unit of time, PRISM can be used to find the best policy that minimizes/maximizes the reward over the given period then return the cumulative reward of the whole period of h slots:

Listing 4.3: Cumulative properties

```
1  Rmin=? [C<=h] or Rmax=? [C<=h]
```

This can be used to estimae for example the energy consumption over a day.

Reachability properties Given an MDP, PRISM can be used to find the best policy that minimizes/maximizes the accumulated reward until reaching some set of states defined by a formula phi in an MDP:

Listing 4.4: Reachability properties

```
1 Rmin=? [F phi] or Rmax=? [F phi]
```

This can be used to estimate for example the maximum expected time for the system to be idle.

As PRISM can be used for the specification and analysis of a Markov decision process (MDP) model, in this section we use this probabilistic model-checker software tool to perform our experimentation.

In the following, and in an incremental way, we give detailed experimental results for principally three data center settings:

1. Small model: with a low number of bins in the histogram modeling arrival jobs and a small number of servers. All principal parameters except unitary costs, where chosen to be low in order to produce a small MDP model in terms of state and transition numbers, which allowed us to easily check the correct functioning of the methodology.

2. Medium model: to push further the experimentation, the parameters of the small model have been enlarged by about ten times.

3. Big model: in order to push the model to its practical limits, the parameters of the small model have been enlarged by about one hundred times, high number of bins in the histogram modeling arrival jobs and high number of servers. All principal parameters except unitary costs, where chosen to be high in order to produce an MDP model for a realistic parameters.

The choice of parameter values was done as follows:

1. The queue size b was varied from 1 to the value that makes the system behavior constant.

2. The number of machines and the number of bins in the job arrival histogram, were chosen to be low for experimentation done for the first model, medium for the second one, and large for the large last model.

3. The unitary costs are not fixed: instead, the unitary costs are varied in intervals according to several constraints. For example, we vary c_n, c_m, and c_l with the constraint $c_n > c_m > c_l$ or $c_n < c_m < c_l$ or other constraints of the same style in order to observe the individual impact of each unitary cost.

4. The duration horizon h on which we observe the system, we took a period of 1000 slots which corresponds to a period bigger than on day. This choice allows us to observe the system in a reasonable period of activity.

Notice that the size of the MDP in terms of transition number depends also on number of bins in the histogram modeling arrival jobs (see Theorem 4.2). For that reason the arrival number of bins is low in the first model and high in the third one.

4.1.5.2 Small Model

In this section we analyze a small model. The parameters of this model are shown in Table 4.2 where arrival jobs are modeled by the following simple histogram: $\mathcal{H}_A = (S_A, P_A)$ where $S_A = \{0, 2, 5\}$, $P_A(0) = 0.2$, $P_A(2) = 0.3$, and $P_A(5) = 0.5$.

Table 4.2: Settings of the first numerical analysis for small data center.

Parameters	**Value**	**Unit**	**Description**
b	1-60		buffer size
$\mathcal{M}$	3	servers	total number of servers
$\mathcal{H}_D$	Δ_1	jobs/server	processing capacity of a server
$\lvert S_A \rvert$	3	bins	size of the arrival jobs histogram
c_M	10		cost of energy needed by a server
c_N	10		cost of waiting a job
c_L	10		cost of rejecting a job
c_{on}	0		cost of switching on a server
h	1000	slot	horizon or period of analysis

As we have 3 machines, the number of possible actions is 4:

1. action a_0 to switch-off all servers,
2. action a_1 to switch-on only one server,
3. action a_2 to switch-on only two servers, and
4. action a_3 to switch-on all servers.

Listing 4.6 gives the associated PRISM specification. Asking PRISM to check the following Formula:

Listing 4.5: Formula to check for Small Data Center

```
1 Rmin=?[C<=1000]
```

PRISM returns the minimum cumulative cost of the best strategy over all possible strategies. Table 4.3 summarizes the results.

Listing 4.6: Example of a simple PRISM specification.

```
mdp
const int B=30;
const int m=3;
const int d=1;
const double cN=10, cM=11, cL=11;
const int    d0=0;const double p0=0.200000;
const int    d1=5;const double p1=0.500000;
const int    d2=2;const double p2=0.300000;
module system1
M : [0..m] init 0;
N : [0..B] init 0;
L : [0..5] init 0;

[a0]  true ->
 p0:(N'=min(B,max(0,N+d0-M*d)))&(M'=0)&(L'=max(0,N+d0-M*d-B))
+p1:(N'=min(B,max(0,N+d1-M*d)))&(M'=0)&(L'=max(0,N+d1-M*d-B))
+p2:(N'=min(B,max(0,N+d2-M*d)))&(M'=0)&(L'=max(0,N+d2-M*d-B));

[a1]  true ->
 p0:(N'=min(B,max(0,N+d0-M*d)))&(M'=1)&(L'=max(0,N+d0-M*d-B))
+p1:(N'=min(B,max(0,N+d1-M*d)))&(M'=1)&(L'=max(0,N+d1-M*d-B))
+p2:(N'=min(B,max(0,N+d2-M*d)))&(M'=1)&(L'=max(0,N+d2-M*d-B));

[a2]  true ->
 p0:(N'=min(B,max(0,N+d0-M*d)))&(M'=2)&(L'=max(0,N+d0-M*d-B))
+p1:(N'=min(B,max(0,N+d1-M*d)))&(M'=2)&(L'=max(0,N+d1-M*d-B))
+p2:(N'=min(B,max(0,N+d2-M*d)))&(M'=2)&(L'=max(0,N+d2-M*d-B));

[a3]  true ->
 p0:(N'=min(B,max(0,N+d0-M*d)))&(M'=3)&(L'=max(0,N+d0-M*d-B))
+p1:(N'=min(B,max(0,N+d1-M*d)))&(M'=3)&(L'=max(0,N+d1-M*d-B))
+p2:(N'=min(B,max(0,N+d2-M*d)))&(M'=3)&(L'=max(0,N+d2-M*d-B));
endmodule
rewards "r"
[a0]  true : M*cM+N*cN+L*cL;
[a1]  true : M*cM+N*cN+L*cL;
[a2]  true : M*cM+N*cN+L*cL;
[a3]  true : M*cM+N*cN+L*cL;
endrewards
```

Table 4.3: Result of the numerical analysis for small data center.

Parameters	**Value**	**Unit**
Size of the Prism specification	40	line
Building model time	0.003	second
Number of states	36	state
Number of transitions	420	transition
Needed memory space	12.0	KB
Verification time	0.023	second
Best policy cost	44251.81	

More experiments where done to analyze the behavior of the system when changing the buffer size b and the unitary costs c_N, c_M and c_L. Results are shown in the following figures.

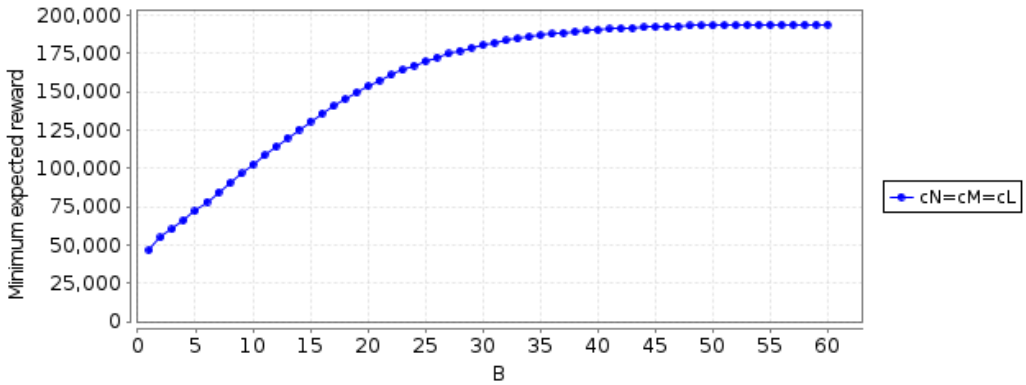


Figure 4.4: By solving the MDP of the small data center described in Section 4.1.5.2, this experiment shows the total cost (minimum expected reward) for different values of buffer size $0 < b < 60$ where c_N, c_M and c_L are equal. We observe that the total cost increases when b is less than some value b_0 (for example around 45 here). However, when $b > b_0$ the total cost seems to be constant. We can explain this behavior as follows: for a small size of buffer, the number of rejected jobs is important but the number of waiting jobs and operational servers is low which leads to a low total cost. As b increases the number of served jobs increases in fact the total cost will increase. When the buffer size is bigger than b_0 the number of rejected jobs becomes negligible but as the arrival jobs are bounded, the number of waiting jobs becomes relatively stable, which leads to a constant number of running servers and to a constant total cost.

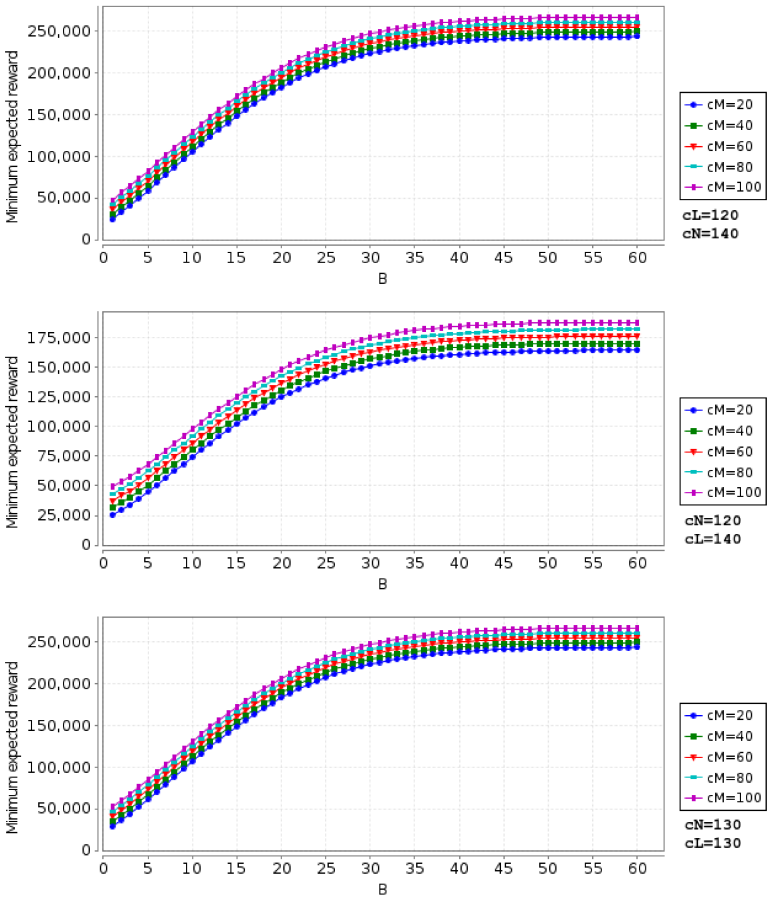


Figure 4.5: By solving the MDP of the small data center described in Section 4.1.5.2, this experiment shows the total cost (minimum expected reward) for different values of buffer size $0 < b < 60$ where varying the energetic cost c_M and keeping it less than c_N and c_L. In the top figure the unitary costs are chosen in a way that $c_M < c_L < c_N$. In the middle figure the unitary costs are chosen so $c_M < c_N < c_L$. Finally in the bottom figure the unitary costs are chosen to verify: $c_M < c_L = c_N$. We can observe also that when c_M is lower than c_N and c_L, the order between c_N and c_L does not affect the behavior of the system. And a more important value of c_M, just increases the overall cost (accumulated cost).

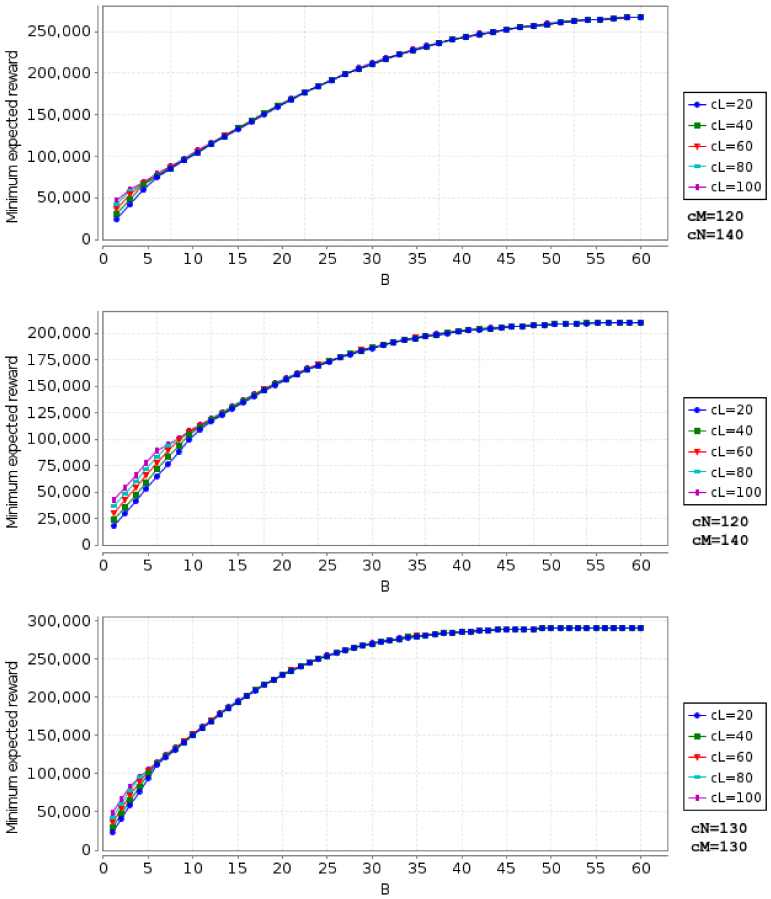


Figure 4.6: In this experiment we are varying the rejection cost c_L and keeping it less than c_N and c_M. In the top figure the unitary costs are chosen in a way that $c_L < c_M < c_N$. In the middle figure the unitary costs are chosen so $c_L < c_N < c_M$. Finally in the bottom figure the unitary costs are chosen to verify: $c_L < c_M = c_N$. We can observe also that when c_L is lower than c_N and c_L, the order between c_N and c_M does not affect the behavior of the system. For a value of b sufficiently large the value of c_L does not affect the behavior of the system. We can explain that by the fact that a big value of b leads to a low rejection rate. In fact the total cost will not increase so much even if we increase c_L. However, for small value of b the rejection rate is more important and increasing c_L leads to a higher overall cost (accumulated cost).

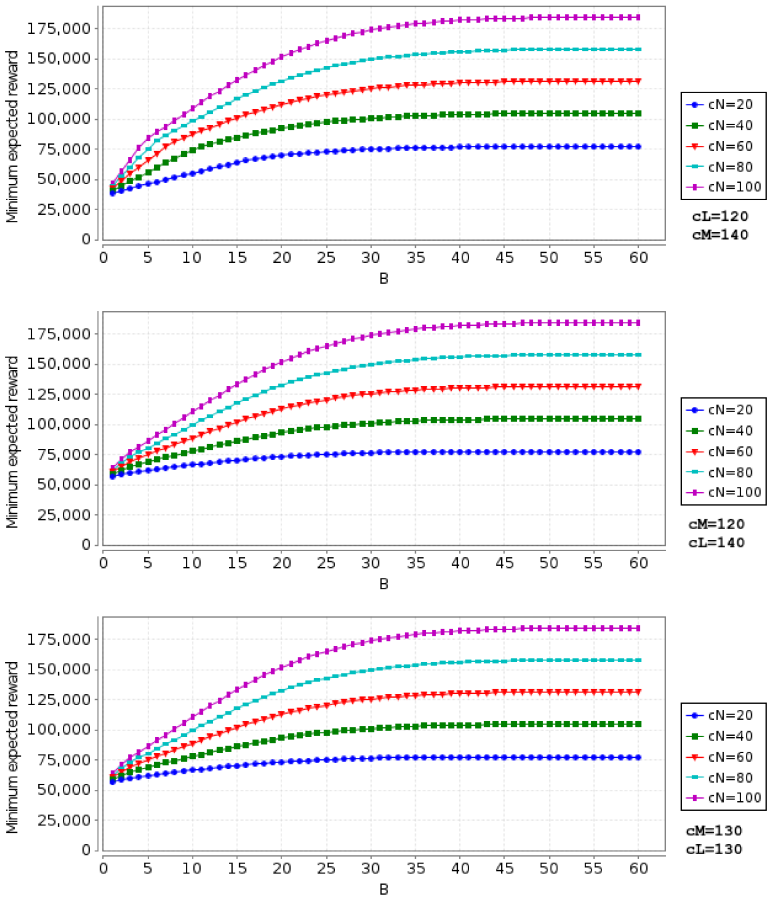


Figure 4.7: In this experiment we are varying the waiting cost c_N and keeping it less than c_L and c_M. In the top figure the unitary costs are chosen in a way that $c_N < c_L < c_M$. In the middle figure the unitary costs are chosen so $c_N < c_M < c_L$. Finally in the bottom figure the unitary costs are chosen to verify: $c_N < c_M = c_L$. We can observe also that when c_N is lower than c_L and c_M, the order between c_L and c_M may affect the behavior of the system. For the middle figure, where c_L is more important than c_M the system turns more servers on to avoid rejecting jobs. In fact the constant behavior is reached faster. We observe also that in three cases a more important waiting cost leads to a more important overall cost (accumulated cost).

4.1.5.3 Medium Model

In this section we analyze a medium model with medium number of bins in the histogram modeling arrival jobs, and with a medium number of servers. All principal parameters except unitary costs, where chosen to be medium in order to produce a medium MDP model in terms of state and transition numbers. The parameters of this model are shown in Table 4.8 where arrival jobs are modeled by the discrete distributions showed in Figure 4.8.

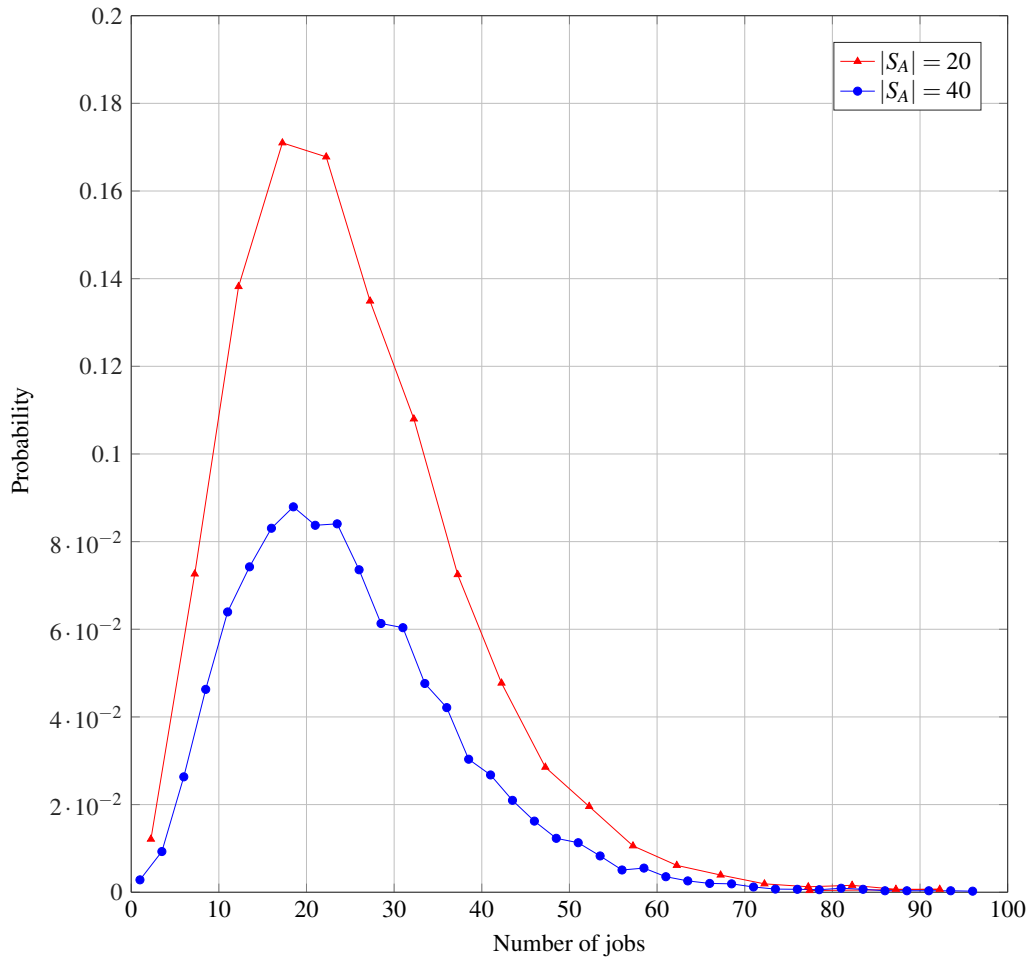


Figure 4.8: Discrete distributions for a medium model.

Table 4.4: Settings of the numerical analysis for a medium data center.

Parameters	**Value**	**Unit**	**Description**
$\mathcal{M}$	50	servers	total number of servers
$\mathcal{H}_D$	Δ_1	jobs/server	processing capacity of a server
$\lvert S_A \rvert$	20 and 40	bins	size of the arrival jobs histogram
c_M	10		cost of energy needed by a server
c_N	10		cost of waiting a job
c_L	10		cost of rejecting a job
h	1000	slot	horizon or period of analysis

As we have 50 machines, the number of possible actions is 51. As in the previous section we ask PRISM to check the following Formula:

Listing 4.7: Formula to check for a medium Data Center

```
1 Rmin=?[C<=1000]
```

to get the minimum cumulative cost of the best strategy over all possible strategies. Table 4.5 summarizes the results:

Table 4.5: Result of the numerical analysis for a medium data center.

Parameters	**Value for** $\lvert S_A \rvert = 20$	**Value for** $\lvert S_A \rvert = 40$
Size of the Prism specification	1200 line	2400 line
Building model time	9.967 second	17.097 second
Number of states	15 049	16 665
Number of transitions	26 541 790	59 897 646
Needed memory space	619.5 MB	1.4 GB
Verification time	5.55 minute	16.74 minute
Best policy cost	479 729.61	561 246.45

More experiments where done to analyze the behavior of the system when changing the buffer size b and the unitary costs c_N, c_M and c_L. Results are shown in the following figures.

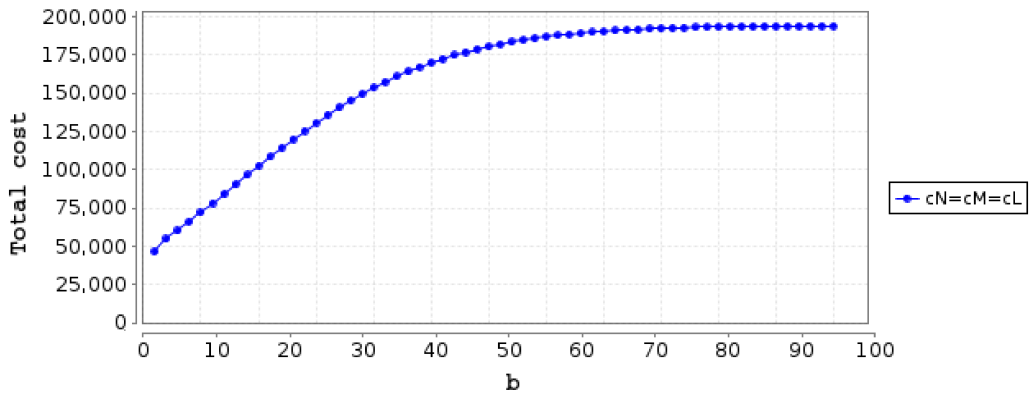


Figure 4.9: By solving the MDP of the medium data center described in Section 4.1.5.2, this experiment shows the total cost (minimum expected reward) for different values of buffer size $0 < b < 100$ where c_N, c_M and c_L are equal. We have the same observation as for Figure 4.4.

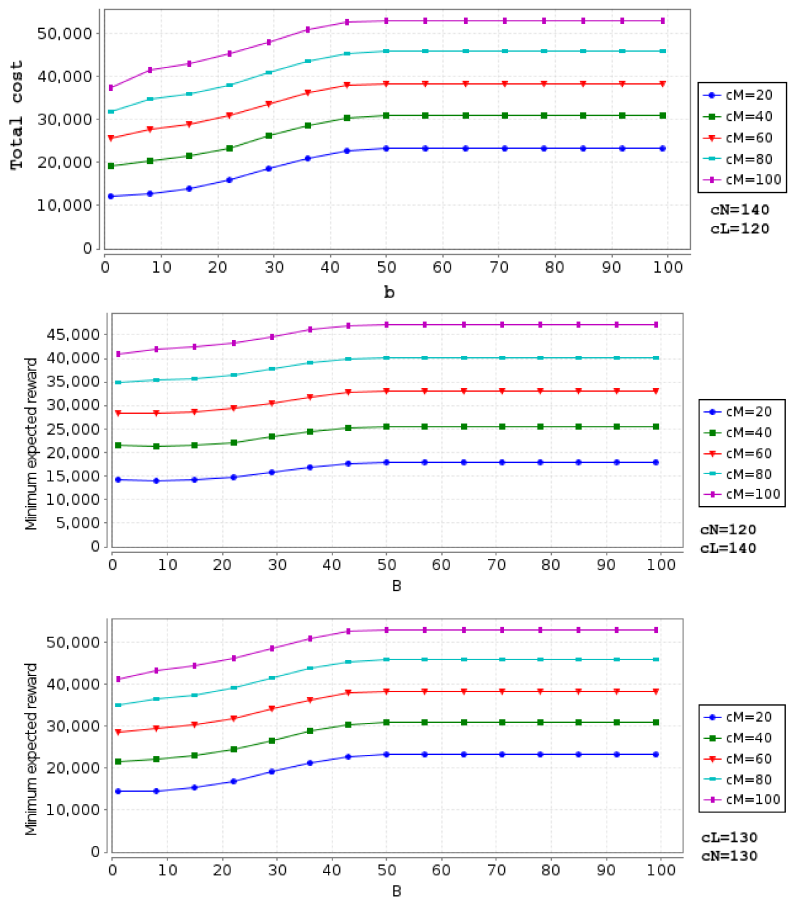


Figure 4.10: In this experiment we are varying the waiting cost c_N and keeping it less than c_L and c_M. In the top figure the unitary costs are chosen in a way that $c_N < c_L < c_M$. In the middle figure the unitary costs are chosen so $c_N < c_M < c_L$. Finally in the bottom figure the unitary costs are chosen to verify: $c_N < c_M = c_L$. We have the same observation as for Figure 4.5.

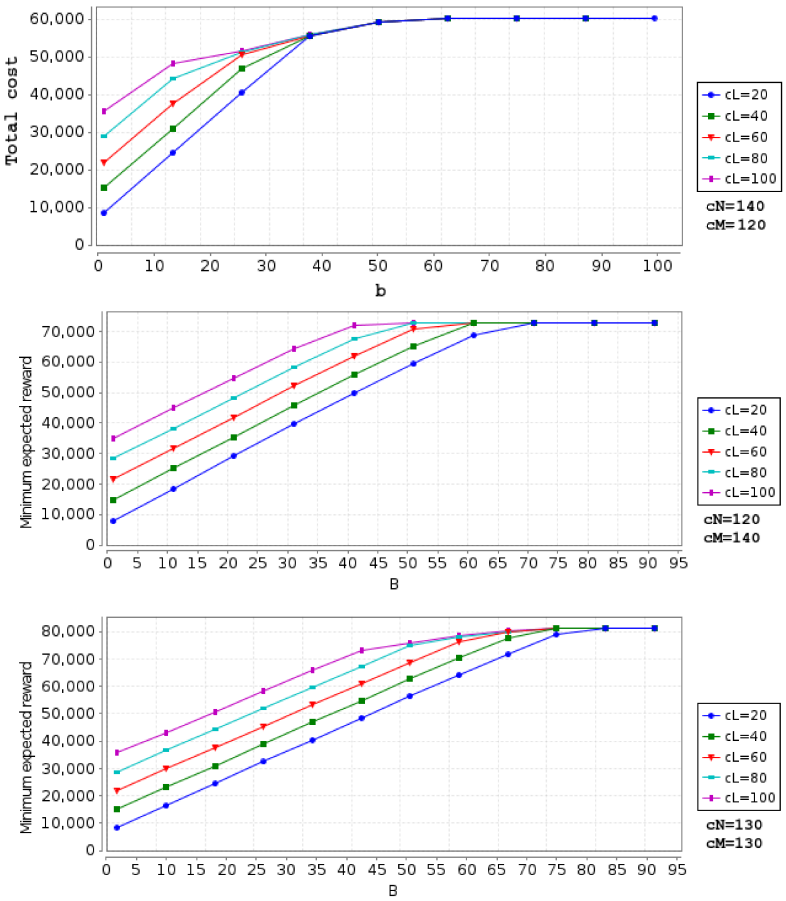


Figure 4.11: In this experiment we are varying the rejection cost c_L and keeping it less than c_N and c_M. In the top figure the unitary costs are chosen in a way that $c_L < c_M < c_N$. In the middle figure the unitary costs are chosen so $c_L < c_N < c_M$. Finally in the bottom figure the unitary costs are chosen to verify: $c_L < c_M = c_N$. We have the same observation as for Figure 4.6.

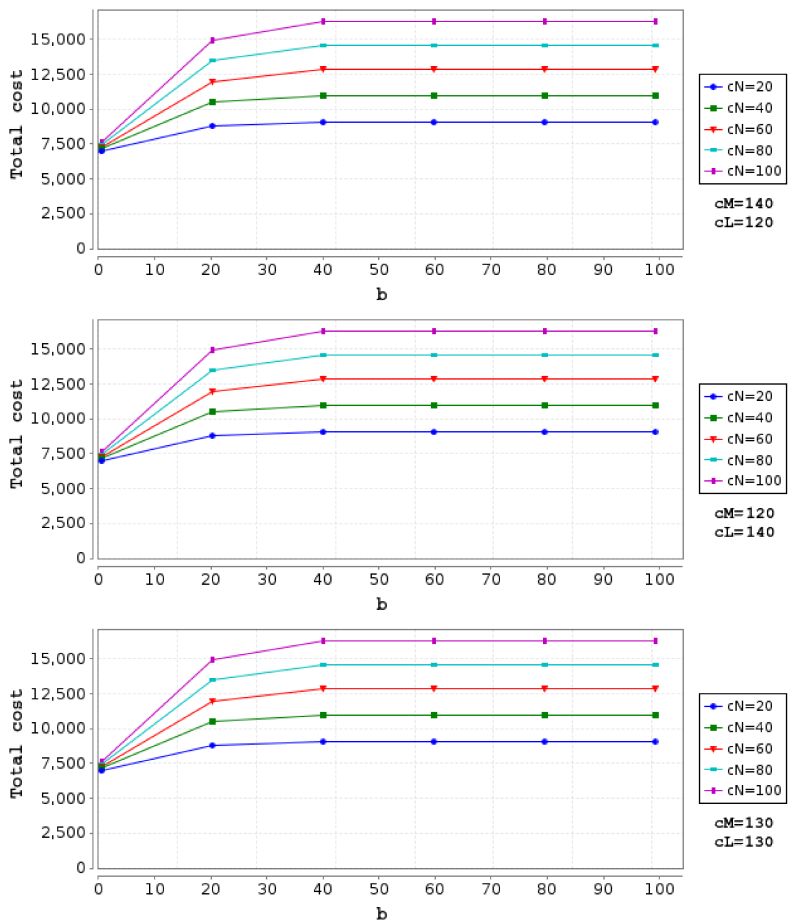


Figure 4.12: In this experiment we are varying the waiting cost c_N and keeping it less than c_L and c_M. In the top figure the unitary costs are chosen in a way that $c_N < c_L < c_M$. In the middle figure the unitary costs are chosen so $c_N < c_M < c_L$. Finally in the bottom figure the unitary costs are chosen to verify: $c_N < c_M = c_L$. We have the same observation as for Figure 4.7.

4.1.5.4 Big Model

In this section we analyze a big model with large number of bins in the histogram modeling arrival jobs, and with a large number of servers. All principal parameters except unitary costs, where chosen to be large in order to produce an MDP model with more realistic parameters. The parameters of this model are shown in Table 4.6 where arrival jobs are modeled by the discrete distributions showed in Figure 4.13.

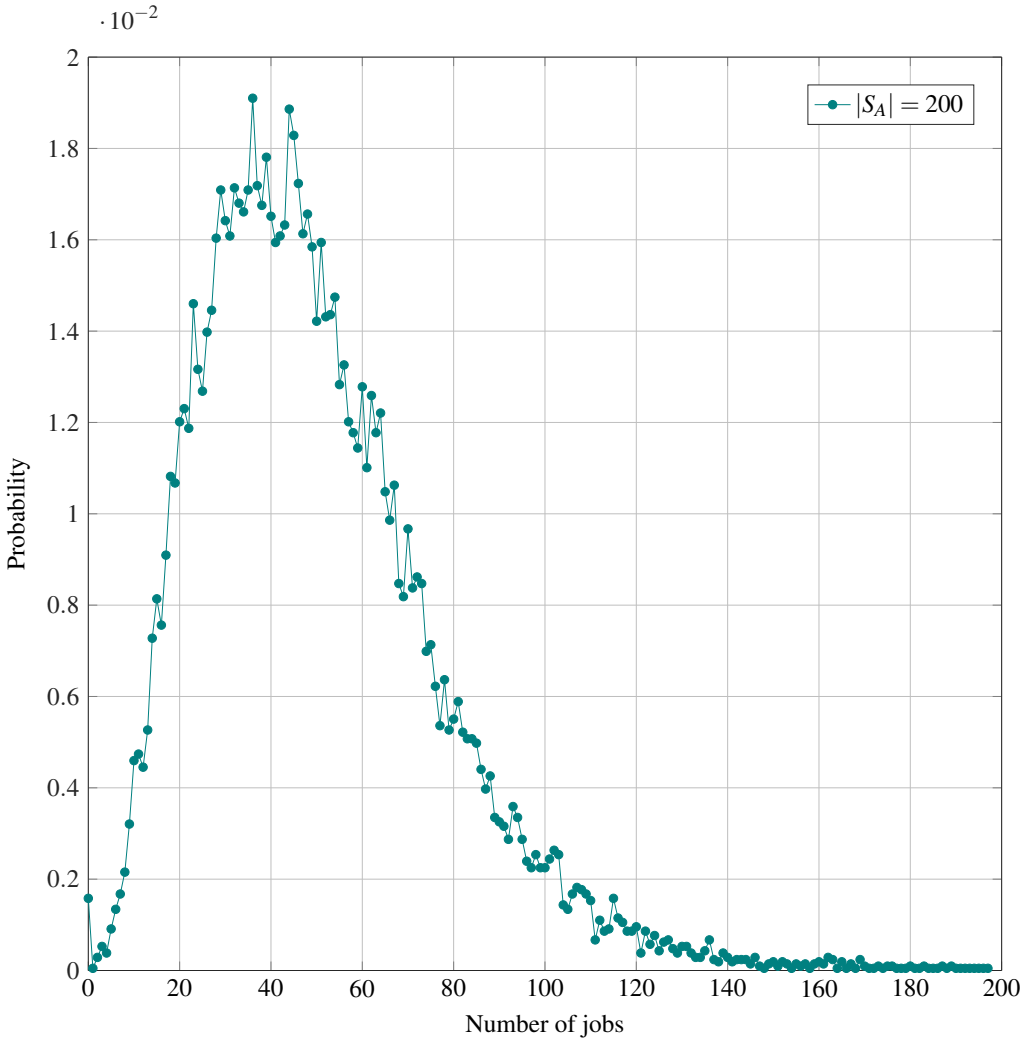


Figure 4.13: Discrete distributions for big model.

Table 4.6: Settings of the numerical analysis for big data center.

Parameters	**Value**	**Unit**	**Description**
$\mathcal{M}$	100	servers	total number of servers
$\mathcal{H}_D$	Δ_1	jobs/server	processing capacity of a server
b	100	jobs	buffer size
$\lvert S_A \rvert$	200	bins	size of the arrival jobs histogram
c_M	10		cost of energy needed by a server
c_N	10		cost of waiting a job
c_L	10		cost of rejecting a job
h	1000	slot	horizon or period of analysis

As we have 100 machines, the number of possible actions is 101. As in the previous sections we ask PRISM to check the following Formula:

Listing 4.8: Formula to check for Big Data Center

```
1 Rmin=?[C<=1000]
```

to get the minimum cumulative cost of the best strategy over all possible strategies. Table 4.7 summarizes the results:

Table 4.7: Result of the numerical analysis for big data center.

Parameters	**Value**	**Unit**
Size of the Prism specification	20000	line
Building model time	1.36	minute
Number of states	39 491	state
Number of transitions	772 399 318	transition
Needed memory space	out of memory	-
Verification time	impossible	-
Best policy cost	impossible	-

PRISM is not able to solve the MDP model to give the best cost strategy for this system. Because of the explosion of the number of states PRISM gives the message ***out of memory*** (see Figure 4.14).

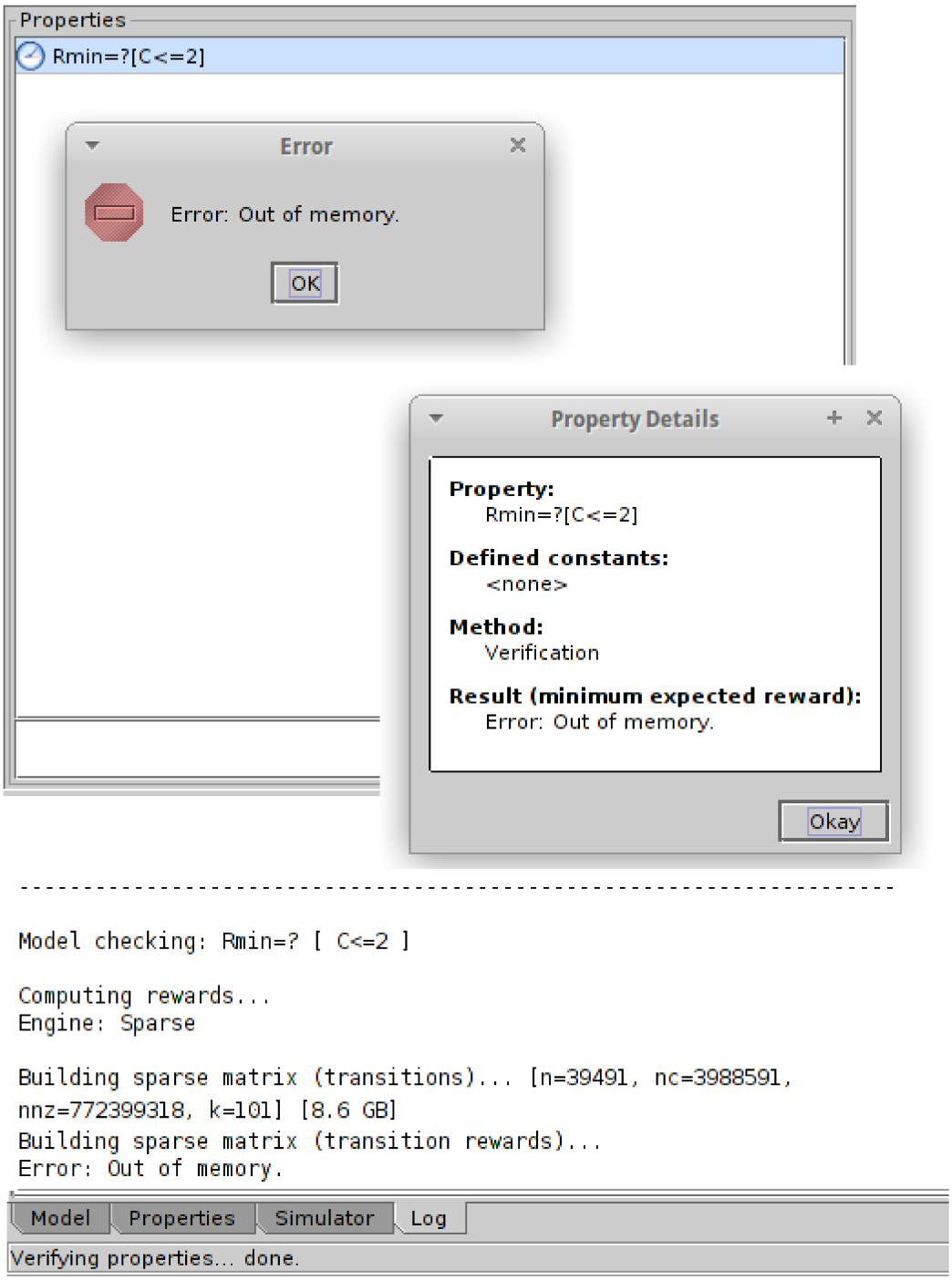


Figure 4.14: PRISM is not able to solve the MDP model to give the best cost strategy for a real data center parameters.

4.1.5.5 Comparison with the threshold strategy

As explained in the previous chapter, we presented an approch that uses real traffic traces then determines the best threshold based policy presented in Section 3.2 to find trade-off between energy consumption and performance evaluation. Under the same parameters of Table 4.8, this section compares the results obtained by the threshold based policy with those obtained when using MDP approach. The results obtained are presented in Table 4.9.

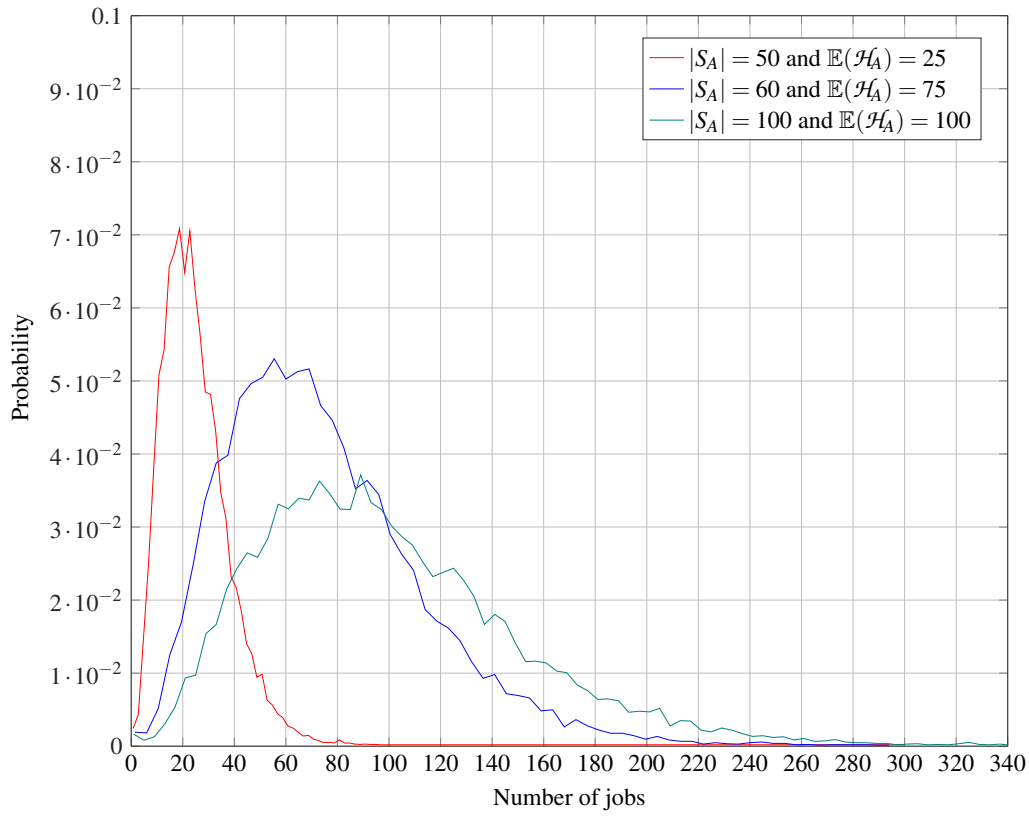


Figure 4.15: Discrete distributions.

Table 4.8: Settings of different data centers parameters.

Setting	Data center	Total number of servers	Buffer size
First	Small	$\mathcal{M} = 21$ server	$b = 31$ job
Second	Medium	$\mathcal{M} = 51$ server	$b = 59$ job
Third	Big	$\mathcal{M} = 103$ server	$b = 111$ job

Table 4.9: Result of the numerical analysis: Note that the gain is computed against the cost generated in the case where all servers are switched on during all the period of time.

Data center	Approach	Total cost gain	Memory	Verification time	Best policy cost
Small	**MDP**	59%	622 MB	34 min	7.3×10^4
	Threshold	55%	100 MB	7 s	7.8×10^4
Medium	**MDP**	61%	3.9 GB	4 h	8×10^4
	Threshold	58%	300 MB	1 min	8.5×10^4
Big	**MDP**	64%	5.8 GB	13 h	9×10^4
	Threshold	60%	400 MB	5 min	9.3×10^4

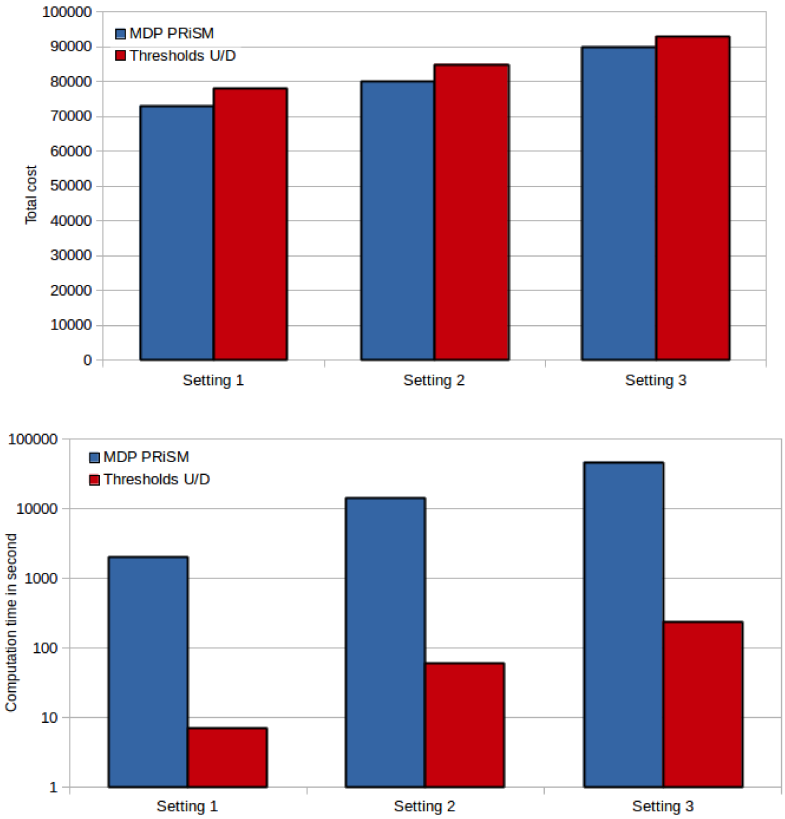


Figure 4.16: Top: total cost of the policy obtained from solving the MDP model under PRISM (blue) and the strategy given by threshold method (red) for a set of different data center settings. Bottom: with logarithmic scale, the time needed to compute the policy when using MDP model under PRISM (blue) and threshold method with the tool of [BDF+16] (red).

Results show that PRISM is able to solve the MDP model to give the best cost strategy that reduces significantly the total cost. The threshold method gives a sub-optimal cost which is close to the MDP one. The threshold approach is more interesting if we need to compute a satisfying strategy in short time when using a small amount of space memory.

In this section we presented a discrete-time Markov decision process model for an optimal management of a data center with homogeneous servers, with the objective of minimizing energy consumption when avoiding the degradation of the QoS. Additionally, some theoretical results are presented, as well as a numerical study on the optimal policy.

4.2 MDP for heterogeneous data center

Our analysis of the Google trace servers [Wil11, RWH11] shows that data centers present a non negligible level of server heterogeneity. Mainly, three levels of heterogeneity were observed based on two criteria: CPU-speed and memory-size of each machine. Those criteria are good indicators for QoS and also energy consumption. Figure 4.17 shows that a very important number of servers runs with a medium CPU-speed and a medium amount of memory. Otherwise a significant number of machines run under a high CPU-speed and memory-size.

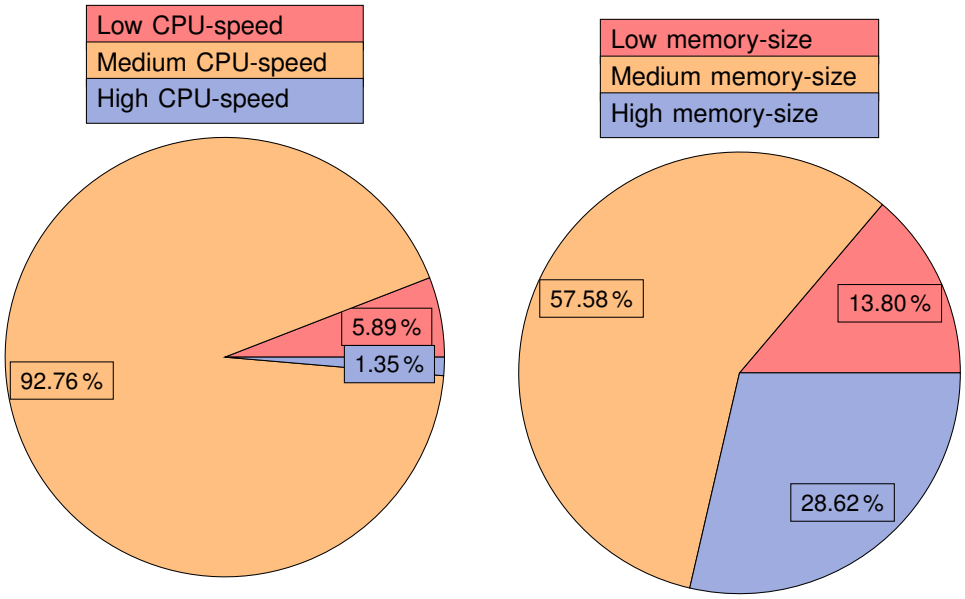


Figure 4.17: Google trace servers repartition according to CPU-speed and memory-size [Wil11, RWH11].

As said before, MDP models for heterogeneous systems were considered in several works as in [Li12] and [NIG07]. In particular [LV02, Efr04, KAR11] used continuous-time MDP's for a heterogeneous data center where jobs arrive in a Poisson fashion and join a single queue served by several non identical servers. Then the authors study structural properties of the optimal policy as the threshold structure property. In fact in this section we will generalize the discrete-time MDP for the homogeneous model used in the previous section to a heterogeneous one.

4.2.1 Generalization of the problem

As previously in this section we consider a discrete-time queue model: a batch arrival queue with a finite capacity buffer b where arrival jobs and service rates are modeled by histograms. A data center is composed of heterogeneous servers grouped essentially into several levels of energy consumption. To simplify we consider the set of the following levels $G = \{high, med, low\}$. The number of operational servers of type g is denoted by m_g and $\mathcal{M}_g$ is the maximal number of servers of level g. During one time unit, the number of jobs that can be served by a server of type g is modeled by a histogram $\mathcal{H}_{D_g}$ where $P_{D_g}(d)$ gives the probability to process d jobs. Thus, the total number of operational servers is $\sum_{g \in G} m_g$ and $\mathcal{M} = \sum_{g \in G} \mathcal{M}_g$ is the maximal number of operational servers.

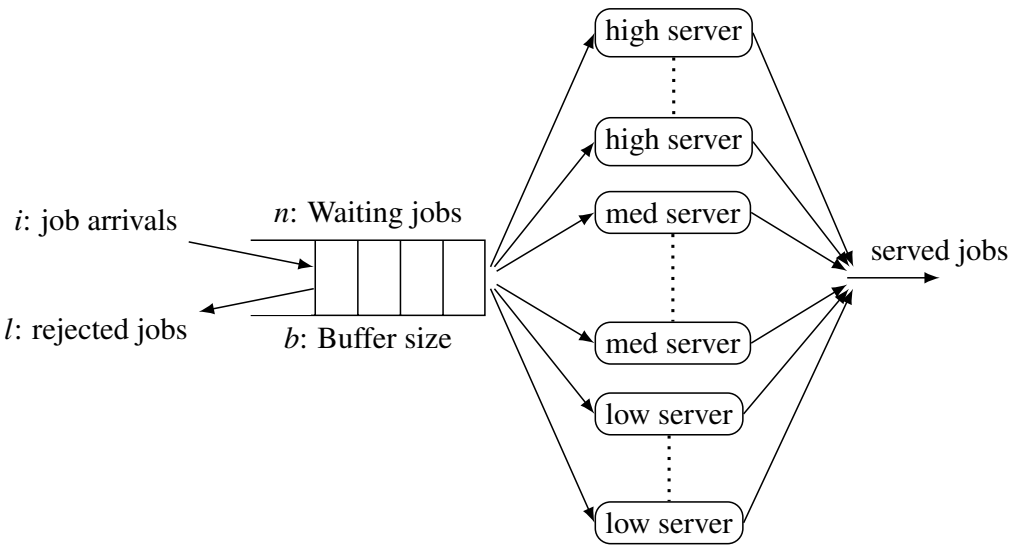


Figure 4.18: Illustration of the queuing model.

The number of waiting jobs in the buffer is denoted by n. The number of

rejected (lost) jobs is denoted by l. We assume that initially the number of operational servers, the number of waiting jobs, and the number of rejected jobs are 0. The number of waiting jobs n can be computed by induction where the exact sequence of events during a slot have to be described. We begin by serving the waiting jobs of the buffer, next we fill the free operational servers by the new jobs, then we fill the buffer. This means that the new jobs are spread over all the free places of the system: in the free operational servers and in the buffer. Notice that this order of events is chosen in this specific way to ensure the FIFO discipline. The job is assigned, at first, to a free server with high running level, if no high server available, the job is attributed to a free server with medium running level, otherwise it will be given to a free server with low running level. The following equations give the number of waiting jobs in the buffer and the lost jobs. For a number of i arrival jobs, we have:

$$\begin{cases} n & \leftarrow & \min\{b, \max\{0, n+i-\sum_{g\in G} m_g \times d_g\}\} \\ l & \leftarrow & \max\{0, n+i-\sum_{g\in G} m_g \times d_g - b\}. \end{cases} \tag{4.15}$$

As i.i.d.-ness assumption of arrivals is assumed (see end of Section 1.3.3), the model of the queue is a time-homogeneous Discrete Time Markov Chain.

Table 4.10 summarizes the energy and QoS metrics.

Table 4.10: Energy and Performance metric for heterogeneous model

Cost	**Meaning**
$c_{M_g} \in \mathbb{R}^+$	energy cost for running one operational server of type g
$c_{On_g} \in \mathbb{R}^+$	energy cost needed to switch-on a server of type g
$c_N \in \mathbb{R}^+$	waiting cost for one job per unit of time
$c_L \in \mathbb{R}^+$	rejection cost of one lost job

In fact, for a given period of time, a dynamic power management system consists in doing at each slot an action (turn on or turn off a specific number of servers) in order to adapt the number of operational servers to incoming job changes. The optimal strategy consists in finding the best sequence of actions to minimize the overall monetary cost (energetic cost plus performance cost). The cost generated for each slot can be presented as:

$$n \times c_N + l \times c_L + \sum_{g\in G} (m_g \times c_{M_g} + \max\{0, m'_g - m_g\} \times c_{On_g}) \tag{4.16}$$

where n is the number of waiting jobs, l is the number of lost jobs, m'_g is the current number of operational servers of level g, and m_g is the number of operational servers before doing an action.

4.2.2 Optimization approach

In order to find the optimal strategy and then analyze the performance and the energy consumption of the data center under this optimal strategy, we will use the concept of *Markov Decision Process* to formulate our optimization problem.

4.2.2.1 Modelization

Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ be an MDP where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition probability, and $\mathcal{C}$ the immediate cost of each action. Let $\mathcal{H}_A = (S_A, P_A)$ be the histogram used to model the arrival of jobs. The state of the system is defined by the couple $((m_{high}, m_{med}, m_{low}), (n, l))$, indeed the state space $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{((m_g : \ g \in G), (n, l)) \mid m_g \in [0..\mathcal{M}_g],\ n \in [0..b] \text{ and } l \in [0..\max(S_A)]\}. \tag{4.17}$$

The action space is defined as:

$$\mathcal{A} = \{\alpha_{(x,y,z)} \mid x \in [0..\mathcal{M}_{high}],\ y \in [0..\mathcal{M}_{med}],\ z \in [0..\mathcal{M}_{low}]\} \tag{4.18}$$

where action $\alpha_{(x,y,z)}$ consist of keeping exactly: x high, y medium, and z low operational servers, during the current slot. Indeed, we have a probability of $\mathcal{P}_{ss'}^{(\alpha_{(x,y,z)})}$ to move from state $s = (m_{high}, m_{med}, m_{low}, (n, l))$ to state $s' = (x, y, z, (n', l'))$ under action $\alpha_{(x,y,z)}$. This probability is defined as:

$$\mathcal{P}_{ss'}^{\alpha_{(x,y,z)}} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_A \text{ and each } d_g \in S_{D_g} \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n + i - x \times d_{high} - y \times d_{med} - z \times d_{low}\}\} \\ l' & = \max\{0, n + a - x \times d_{high} - y \times d_{med} - z \times d_{low} - b\} \end{array}\right.} P_A(i) \times \prod_{g \in G} P_{D_g}(d_g) \tag{4.19}$$

where n' is the new number of waiting jobs, l' is the new number of lost jobs, and x, y, z are the number of operational servers for each level of G after applying action $\alpha_{(x,y,z)}$. In this case, applying action $\alpha_{(x,y,z)}$ induces immediately a cost $\mathcal{C}_s^{\alpha_{(x,y,z)}}$ defined as:

$$\begin{array}{rlll} \mathcal{C}_s^{\alpha_{(x,y,z)}} = & n \times c_N + l \times c_L & & \\ & +x \times c_{M_{high}} & + \ \max\{0, x - m_{high}\} & \times c_{On_{high}} \\ & +y \times c_{M_{med}} & + \ \max\{0, y - m_{med}\} & \times c_{On_{med}} \\ & +z \times c_{M_{low}} & + \ \max\{0, z - m_{low}\} & \times c_{On_{low}}. \end{array} \tag{4.20}$$

The immediate cost $\mathcal{C}_s^{\alpha_{(x,y,z)}}$ includes two parts. The QoS part which includes the cost due to waiting and rejected jobs. The power part is composed of energy consumed for running operational servers and energy used to power-up additional servers.

Every state of the MDP includes three elements:

1. number of operational servers for every level which is between 0 and $\mathcal{M}_g$,
2. number of waiting jobs in the buffer which is bounded above by b, and
3. number of rejected jobs which can be at most equal to the maximum number of arrival jobs given by $\max(S_A)$.

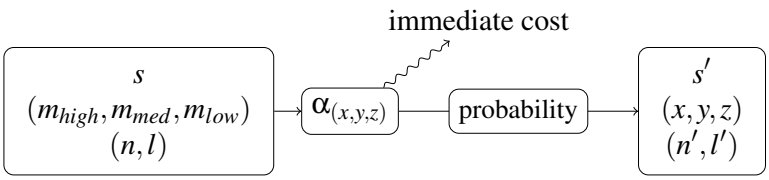


Figure 4.19: Depiction of an MDP action

Table 4.11 summarizes parameters used in our model and our MDP formulation.

4.2.2.2 Memory complexity

In this section we give the spatial complexity of the heterogeneous MDP model in terms of state number and transition number.

Theorem 4.7. *The number of states, and the number of transitions of the heterogeneous MDP model are respectively in* $O\left(b \times \max(S_A) \times \prod_{g \in G} \mathcal{M}_g\right)$ *and in* $O\left(b \times |S_A| \times \max(S_A) \times \prod_{g \in G} \mathcal{M}_g^2\right)$.

Proof. Every state of the MDP includes three element:

1. the number of operational servers for each group which gives a number of combinations of $\prod_{g \in G} (\mathcal{M}_g + 1)$,
2. the number of waiting jobs in the buffer which is bounded above by b, and
3. the number of rejected jobs which can be at most equal to the maximum number of arrival jobs given by $\max(S_A)$.

Table 4.11: Model and MDP Parameters.

Parameters	**Description**
h	duration of analysis
$\mathcal{M}_g$	total number of servers of type g
$\mathcal{H}_{D_g}$	histogram modeling the processing capacity of a server of type g
b	buffer size
m_g	number of operational servers of level g
n	number of waiting jobs
l	number of rejected jobs
$\mathcal{H}_A$	histogram of job arrivals
$\mathcal{S}$	set of all possible states
$\mathcal{A}$	set of all possible actions
s	$s = ((m_g : g \in G), (n, l))$ system state
s_0	$s_0 = ((0,0,0),(0,0))$ starting state
$\alpha_{(x,y,z)}$	action to keep exactly: x high operational servers y medium operational servers z low operational servers
$\mathcal{P}_{ss'}^{\alpha_{(x,y,z)}}$	probability of transition from s to s' under action $\alpha_{(x,y,z)}$
$\mathcal{C}_s^{\alpha_{(x,y,z)}}$	immediate cost from s under action $\alpha_{(x,y,z)}$

So, $|\mathcal{S}|$ is bounded above by $(b+1) \times (\max(S_A)+1) \times \prod_{g \in G}(\mathcal{M}_g+1)$. Additionally, from each state of the MDP we have at most $\prod_{g \in G}(\mathcal{M}_g+1)$ actions, and each action leads to a number of transitions equals to $|S_A|$ (one transition for each bin in the support of the arrival distribution). □

4.2.2.3 Solver algorithm

As we formulate our optimization problem as an MDP, an action consists in turning-on each unit of time a specific number of servers and turning-off the rest of the servers. The optimal strategy is the best sequence of actions to be done in order to minimize the overall cost (accumulated cost) during a finite period of time called *horizon* and noted h. The value function $V : \mathcal{S} \times [0..h] \rightarrow \mathbb{R}^+$ has as objective minimizing the expected sum of costs over time:

$$V_t(s) = \min_{\pi} \mathbb{E}\left[\sum_{k=h-t+1}^{h} \mathcal{C}_{s_k}^{\pi_k(s_k)}\right]. \tag{4.21}$$

The value function can be expressed recursively as:

$$V_t(s) = \min_{\alpha_{(x,y,z)}} \left\{ \mathcal{C}_s^{\alpha_{(x,y,z)}} + \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{\alpha_{(x,y,z)}} V_{t-1}(s') \right\} \tag{4.22}$$

where $\alpha_{(x,y,z)}$ is the action taken by the system, and $\mathcal{P}_{ss'}^{\alpha_{(x,y,z)}}$ is the transition probability from state s to state s'. In this case the optimal policy for each state s is:

$$\pi_t^*(s) \in \operatorname*{argmin}_{\alpha_{(x,y,z)} \in \mathcal{A}} \left\{ \mathcal{C}_s^{\alpha_{(x,y,z)}} + \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^{\alpha_{(x,y,z)}} V_{t-1}(s') \right\}. \tag{4.23}$$

As the value iteration algorithm is an efficient dynamic programming, implementation for solving Bellman equation, we used it to compute the optimal control policy of our MDP.

4.2.3 Example

To illustrate our formalization Let us show a little part of an MDP modeling a very simple data center. Figure 4.20 illustrates a tiny[2] part of an MDP modeling a small data center of four servers (one high server, two medium level servers, and

[2]The number of states of this MDP is around 700.

one low server) with a buffer size equal to four. Which means that $\mathcal{M} = 4$ and $b = 4$. Additionally, to keep the example simple, we set the service rate of each machine type as a histogram with only one bin: $P_{D_{high}}(3) = P_{D_{med}}(2) = P_{D_{low}}(1) = 1$. Job arrivals are modeled by histogram of Example 1.4. For instance, state $s_{82} = ((1,0,1),(3,1))$ means that the system is running with a single high server and a single low server, when three jobs wait in the buffer and one job was rejected. Action $\alpha_{(0,2,1)}$ switches-off the high server and switches-on the two middle servers. This action induces an immediate cost of 1.45 and moves the system to state s_{380} with probability 0.14, to state s_{502} with probability 0.67, and to state s_{121} with probability 0.19.

4.2.4 Optimal strategy structure

As shown previously, the size of the MDP is important, and the computation of the optimal policy can be hard even impossible for a big data center. In fact, it is essential to analyze the structural properties of the optimal policy to make the computation efficient. In particular the double-threshold structure was proved in the case of a homogeneous data center in several models like in [YCNH11].

In the following we will be interested in some properties around the optimal policy, and we show in particular that the property of the double-threshold structure does not hold for our heterogeneous data center model.

Corollary 4.3. *The optimal policy (4.23) is neither isotone nor monotone.*

Proof. The homogeneous model presented in Section is a special case of the heterogeneous model, and as the optimal policy of the homogeneous model is neither isotone (see Corollary 4.1) nor monotone (see Theorem 4.6), the optimal policy of the heterogeneous model is also neither isotone nor monotone. □

4.2.5 Test and results

This section presents a case study which uses real traffic traces based on the open *cluster-data-2011-2* trace [Wil11, RWH11]. As in [BDF+16], we model arrival jobs, and additionally service rates, by discrete distributions build from the job/machine events corresponding to the requests destined to a specific Google data center for the whole month of May 2011. This traffic trace is sampled with a sampling period that ensure the i.i.d.-ness assumption. Thus, we consider frames of 136 second to sample the trace and construct empirical distributions (see Figure 4.21).

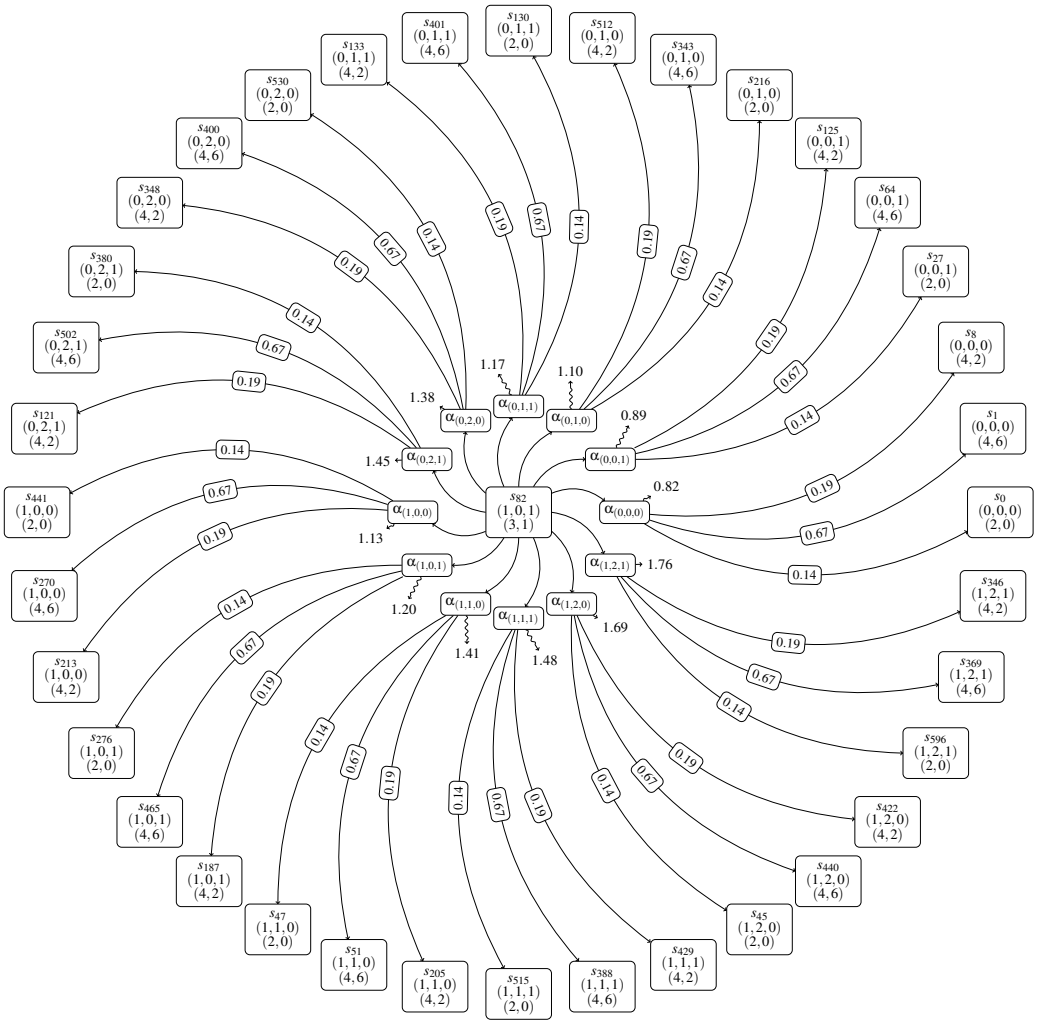


Figure 4.20: MDP example.

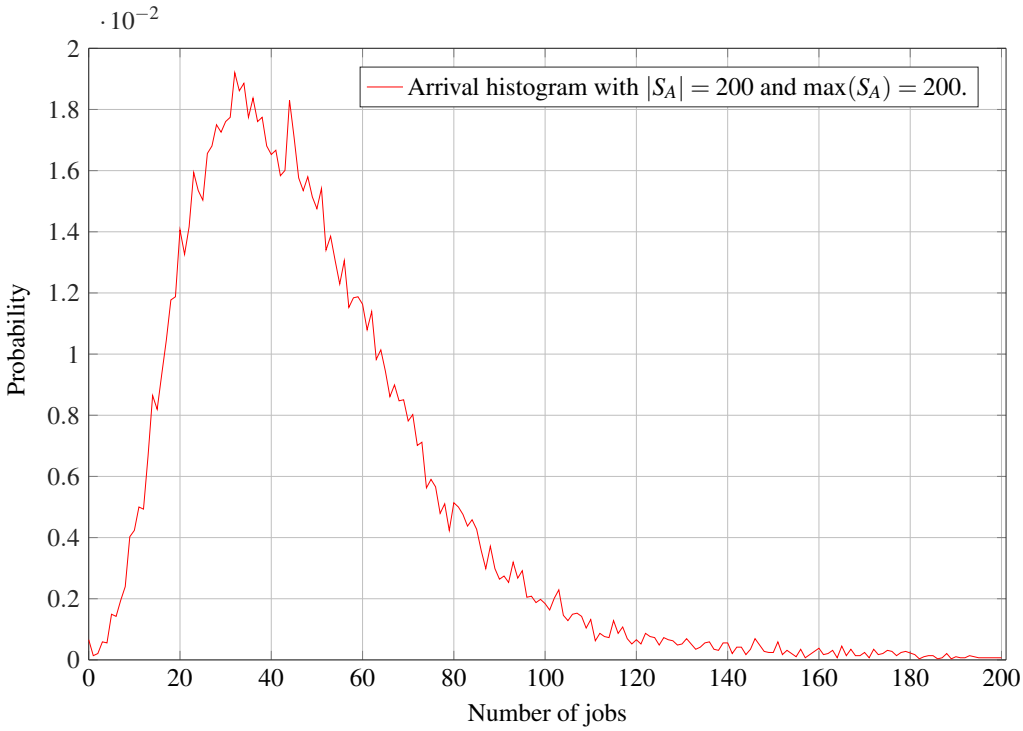


Figure 4.21: Arrival traffic distribution.

In PRISM we need to write a specification for our MDP model which can be written easily if the system parameters were small, but when we consider real systems with big parameters it will be very hard. As an example the PRISM specification associated with a data center with ten heterogeneous servers is a file of several thousands of lines, all lines are different from each other (copy/paste can't help). Trying to write our specification without automated generation is time consuming, gives rise to making easily mistakes, forgetting some cases, losing time for updating or maintaining. Indeed, as the specification of our heterogeneous model is huge and not obvious to write (see complexity given in Section 4.2.2), we have coded a tool that helps us to generate automatically the PRISM specification file.

Table 4.12 summarizes values of the parameters of our system.

Table 4.12: Parameters of the numerical analysis.

b	$\mathcal{M}$	$\mathcal{H}_A$	$\mathcal{H}_{D_{high}}$	$\mathcal{H}_{D_{med}}$	$\mathcal{H}_{D_{low}}$	$c_{M_{high}}$	$c_{M_{med}}$	$c_{M_{low}}$	$c_{On_{high}}$	$c_{On_{med}}$	$c_{On_{low}}$	c_N	c_L
50	10	38	Δ_{10}	Δ_5	Δ_2	6	4	2	3	2	1	1	2

Figure 4.22 shows the minimal expected overall cost (accumulated cost) during

one day ($h = 1$ day$= 635$ slots) when varying the buffer size b and the unitary waiting/losing job cost and keeping the rest of parameters as in Table 4.12.

Table 4.13 summarizes the results obtained for the following configurations:

1. a data center with ten heterogeneous servers,
2. a data center with ten identical high servers,
3. a data center with ten identical medium servers,
4. a data center with ten identical low servers.

Table 4.13: Result of the numerical analysis.

	(i) Heterogeneous servers	**(ii) Homogeneous high servers**	**(iii) Homogeneous medium servers**	**(iv) Homogeneous low servers**
$\mathcal{M}_{\mathbf{high}}$	2	10	0	0
$\mathcal{M}_{\mathbf{med}}$	5	0	10	0
$\mathcal{M}_{\mathbf{low}}$	3	0	0	10
Prism lines	7×10^4	7.5×10^2	7.5×10^2	7.5×10^2
States	1.6×10^4	1.6×10^3	1.6×10^3	1.6×10^3
Transitions	88×10^6	0.81×10^6	0.87×10^6	0.89×10^6
Memory	2 GB	18.8 MB	20.4 MB	20.7 MB
Time	1.6 hour	1 minute	2 minute	2 minute
Expected minimal cost	50342	52251	54175	56381

Results show that the size of the three homogeneous models is relatively small compared to the size of the heterogeneous one which is huge. The size of heterogeneous model is due to the large number of different action combinations. However, heterogeneous model saves more energy. The total cost is 12% (respectively 8%, 4%) better than the low (respectively medium, high) homogeneous model. Figure 4.23 shows the result of experiments in which we are analyzing the total cost over one day when varying the buffer size b for heterogeneous/homogeneous data centers.

We observe that the total cost increases when b is less than some value ($b_0 \simeq 40$). However, when b is bigger than b_0 the total cost seems to be convergent for any configuration.

We can explain this behavior as follows: for a small size of the buffer, the number of waiting jobs is low. This leads to a small number of served jobs. In fact the system switches-on a small number of servers. So the energetic cost and also the waiting jobs are low.

When the buffer size is bigger, the number of waiting jobs is more important. Which leads to a bigger number of served jobs. In fact the system switches-on more servers. So both, the energetic cost and the number of waiting jobs increases.

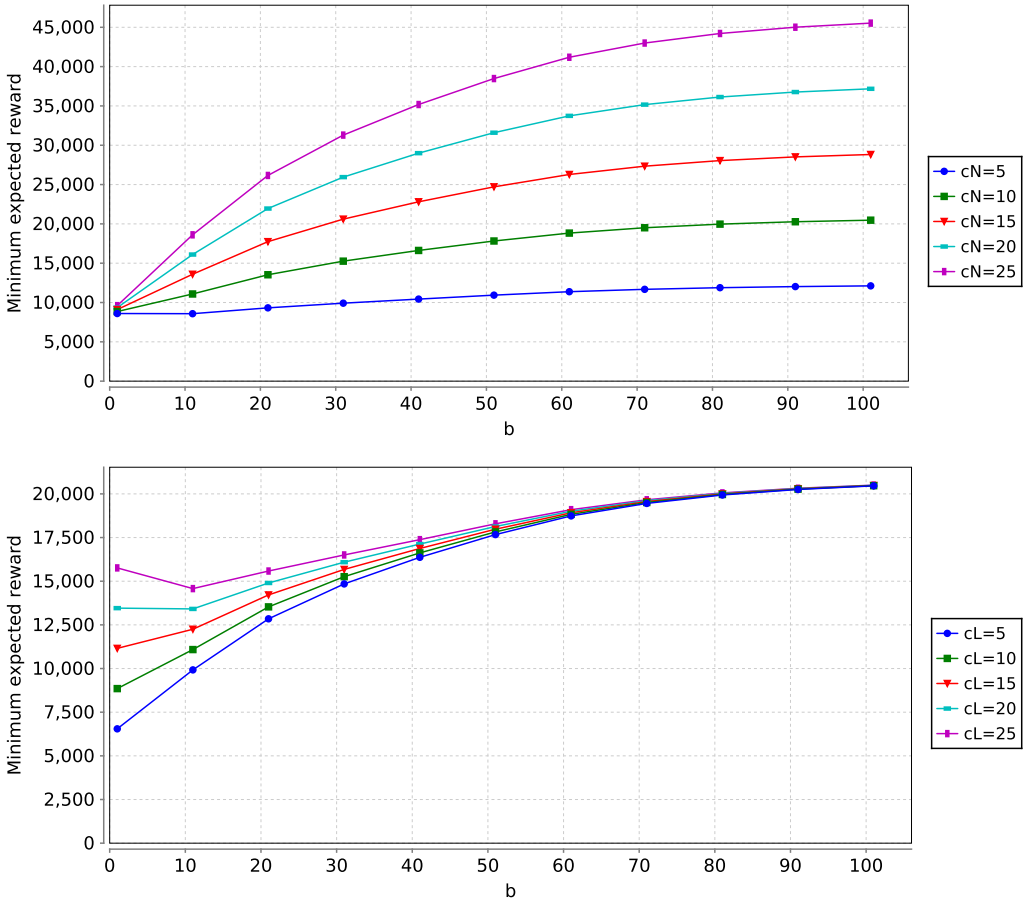


Figure 4.22: Top: varying the unitary waiting job cost c_N while keeping other unitary costs constant. We observe that the bigger c_N is, the greater the total cost is. Bottom: varying the unitary losing job cost c_L while keeping other unitary costs constant. For a value of b sufficiently large, the unitary cost for losing jobs c_L does not affect the total cost of the system.

As the arrival jobs are bounded, when the buffer size is bigger than some value, the number of waiting jobs and the number of served job become relatively stable, which leads to a constant number of running servers and necessarily to a constant total cost.

We can also observe that:

- globally the heterogeneous model leads to a better cost minimization: this can be explained by the fact that the heterogeneous model presents a huge number of ways to combine actions, which leads to a better control policy that may save more energy.
- the homogeneous model with low servers induces an important cost: this can be explained by the fact that low servers have a low service capacity which leads to an important number of waiting jobs ($\mathbb{E}(\mathcal{H}_{D_{low}}) = 2$).

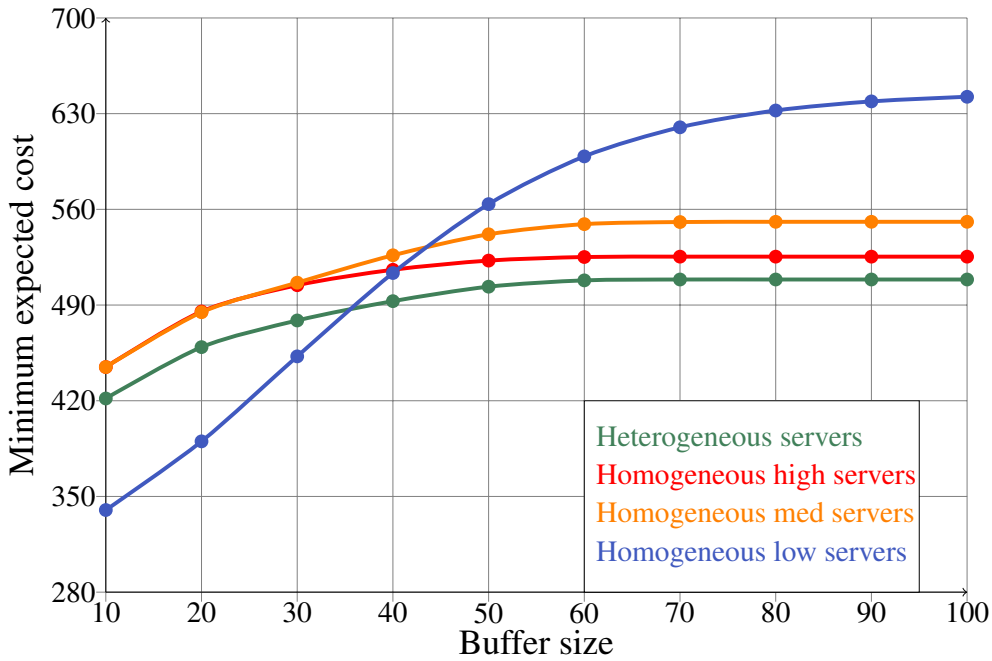


Figure 4.23: Total cost when varying buffer size b for heterogeneous and homogeneous data centers.

In this section we presented a discrete-time Markov decision process model for optimal management of a data center with heterogeneous servers, with the objective of minimizing energy consumption and Quality of Service (QoS) costs. Additionally, some theoretical results are presented, as well as a numerical study on the performance of the optimal policy.

4.3 MDP for data center with latency

It is well known that servers need an additional amount of energy to switch-on (resp. switch-off). This phenomena was handled in the last sections by introducing the c_{on} unitary cost that quantifies this amount. It is also known that the switching-on (resp. switching-off) is not instantaneous and a server needs some units of time to transit from a sleeping/stopping mode to a ready-working mode. This period of time during which the server switches from a mode to another is called *latency*. In the previous models we consider latency as zero. However, in the following we will show how MDP can be modified to include the latency of servers. Notice that modeling our energetic optimization problem with an MDP when including latency is not easy. In fact, and in order to keep the model simple and clear for the reader we will make following assumptions:

1. only latency of switching-on is considered;
2. turning-off a running server is instantaneous which means that switching-off latency is zero;
3. all servers are homogeneous and have the same constant latency period which is of a duration of k units of time.

4.3.1 Problem extension

In this part we also consider a discrete time queue model. Our queuing model is a batch arrival queue with a finite capacity buffer b. We model arrival jobs and service rate by histograms which are discrete distributions formed from real traces. The number of jobs arriving to the data center during a slot is modeled by a histogram $\mathcal{H}_A$ where $P_A(i)$ gives the probability to have i arrival jobs per a slot. A data center is composed of $\mathcal{M}$ homogeneous servers. During one time unit, the number of jobs that can be served by a server is modeled by a histogram $\mathcal{H}_D$ where $P_D(d)$ gives the probability to process d jobs. A server can be in one of the following states:

- stopped: switched-off
- ...
- in latency since i unit of time and needs $k-i$ unit to be ready, for each $i = 0..(k-1)$
- ...
- ready: switched-on

At each slot we need to know the number of servers grouped by states given above (see Figure 4.24). We consider the following set of levels of latency:

$$\mathcal{L} = \{lat_\infty = stop, lat_k, lat_{k-1}, \ldots, lat_i, \ldots, lat_2, lat_1, ready = lat_0\}. \tag{4.24}$$

where lat_i denotes that the server is within the switching-on period and need i slot to be completely ready to work. The number of servers of level $x \in \mathcal{L}$ is denoted by m_x.

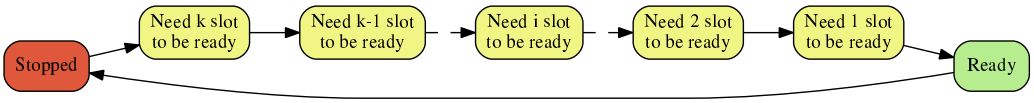


Figure 4.24: Transition between stopping/sleeping mode and ready mode.

Thus, the number of operational servers is $m = m_{ready} = m_{lat_0}$, the number of servers that will be ready within i slot is m_{lat_i}, the total number of servers in latency is given by $m_{lat} = \sum_{i=1}^{k} m_{lat_i}$, and finally $\mathcal{M} = m_{ready} + m_{lat} + m_{stop}$ is the maximal number of operational servers (total number of servers).

The number of waiting jobs in the buffer is denoted by n. The number of rejected (lost) jobs is denoted by l. We assume that initially the number of waiting jobs, the number of rejected jobs are 0, and all servers are in the stopped mode.

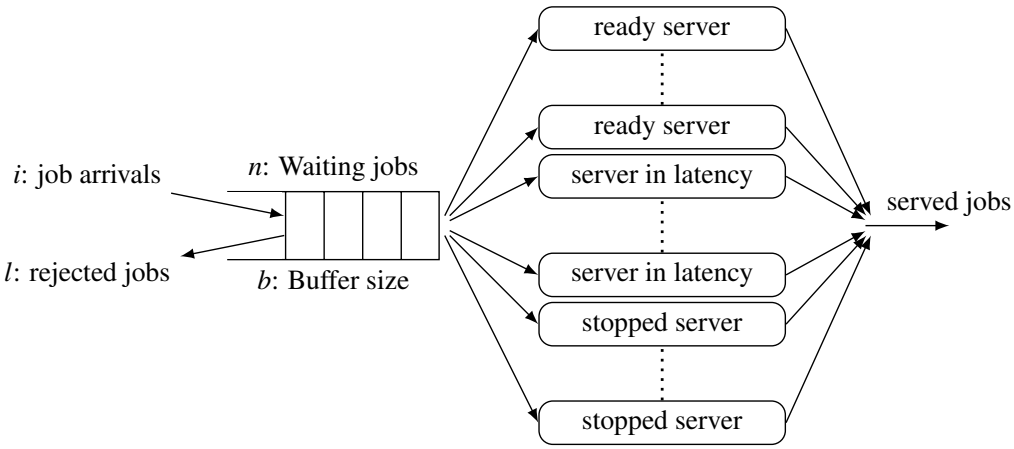


Figure 4.25: Illustration of the queuing model.

The number of waiting jobs n can be computed by induction where the arrival jobs are added to the system (buffer and free operational servers), a maximum of $d \times m_{ready}$ jobs will be on servers to be processed, and the rest of jobs will be

rejected (see Section 1.3.1 for the exact order of events). For a number of i arrival jobs, and a service rate of d we have:

$$\begin{cases} n & \leftarrow & \min\{b, \max\{0, n+i-m_{ready} \times d\}\} \\ l & \leftarrow & \max\{0, n+i-m_{ready} \times d - b\}. \end{cases} \tag{4.25}$$

As i.i.d.-ness assumption of arrivals is assumed (see end of Section 1.3.3), the model of the queue is a time-homogeneous Discrete Time Markov Chain.

The energy consumption takes into account the number of operational servers. A server may consume no energy when it is turned off. But consumes some units of energy per slot when it is operational and it costs some monetary units c_M. As we are considering that a server needs k unit of time to switch on, a server may consume an additional amount of energy that costs also some monetary unit c_{on}. We assume that c_{on} is consumed uniformly during the latency period, let say $\frac{c_{on}}{k}$ every slot. The total energy consumed is the sum of all units of energy consumed during a specific period. The QoS takes into account the number of waiting and lost jobs. Each waiting (resp. rejected) job costs some monetary units c_N (resp. c_L). For a given period of time, a dynamic power management system consists in doing each slot an action (turn on or turn-off a specific number of servers) in order to adapt the number of operational servers to incoming job changes. The optimal strategy consists in finding the best sequence of actions to minimize the overall monetary cost (energetic cost plus performance cost). The cost generated for each slot can be presented as:

$$n \times c_N + l \times c_L + m_{ready} \times c_M + m_{lat} \times \frac{c_{on}}{k} \tag{4.26}$$

where n is the number of waiting jobs, l is the number of lost jobs, m_{ready} is the current number of operational servers, and m_{lat} is the number of servers in latency mode.

4.3.2 Optimization approach

4.3.2.1 Modelization

Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ be an MDP where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition probability, and $\mathcal{C}$ the immediate cost of each action. Let $\mathcal{H}_A = (S_A, P_A)$ be the histogram used to model the arrival of jobs, and let $\mathcal{H}_D = (S_D, P_D)$ be the histogram used to model the service rate. The state of the system is defined by the couple $((m_{stop}, m_{lat_k}, m_{lat_{k-1}}, \ldots, m_{lat_i}, \ldots, m_{lat_2}, m_{lat_1}, m_{ready}), (n, l))$, indeed the state space $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{((m_x : x \in \mathcal{L}), (n, l)) \mid m_x \in [0..\mathcal{M}], n \in [0..b] \text{ and } l \in [0..\max(S_A)]\}. \tag{4.27}$$

The action space is defined as:

$$\mathcal{A} = \{\alpha_{+z} \,|z \in [1..\mathcal{M}]\} \cup \{\alpha_0\} \cup \{\alpha_{-z} \,|z \in [1..\mathcal{M}]\} \tag{4.28}$$

where:

1. action α_0 consists in doing nothing;
2. action α_{+z} consists in switching-on z additional servers if the number of stopped servers is more than z, otherwise switches-on all stopped machines;
3. action α_{-z} consists in switching-off z servers if the number of ready servers is more than z, otherwise switches-off all the ready machines.

If the current state of the system is:

$$s = \begin{cases} (m_{stop}, m_{lat_k}, m_{lat_{k-1}}, \ldots, m_{lat_i}, \ldots, m_{lat_2}, m_{lat_1}, m_{ready}) \\ (n, l) \end{cases} \tag{4.29}$$

then:

1. action α_0 moves the system to the state s^0:

$$s^0 = \begin{cases} (m_{stop}, 0, m_{lat_k}, m_{lat_{k-1}}, \ldots, m_{lat_i}, \ldots, m_{lat_2}, m_{lat_1} + m_{ready}) \\ (n', l') \end{cases} \tag{4.30}$$

so we have a probability of $\mathcal{P}_{ss^0}^{\alpha_0}$ and an immediate cost of $\mathcal{C}_s^{\alpha_0}$ to transit from state s to state s^0 under action α_0:

$$\mathcal{P}_{ss^0}^{\alpha_0} = \sum_{\begin{cases} \text{for} & \text{each } i \in S_A \text{ and each } d \in S_D \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-(m_{ready}+m_{lat_1}) \times d\}\} \\ l' & = \max\{0, n+a-(m_{ready}+m_{lat_1}) \times d - b\} \end{cases}} P_A(i) \times P_D(d) \tag{4.31}$$

$$\mathcal{C}_s^{\alpha_0} = n \times c_N + l \times c_L + (m_{lat_1} + m_{ready}) \times c_M + \sum_{i=2}^{k} m_{lat_i} \times \frac{c_{on}}{k} \tag{4.32}$$

2. action α_{+z} moves the system to the state s^+:

$$s^+ = \begin{cases} (\max\{m_{stop} - z, 0\}, \min\{z, m_{stop}\}, m_{lat_k}, m_{lat_{k-1}}, \ldots \\ \quad \ldots, m_{lat_i}, \ldots, m_{lat_2}, m_{lat_1} + m_{ready}) \\ (n', l'). \end{cases} \tag{4.33}$$

so we have a probability of $\mathcal{P}_{ss^+}^{\alpha_{+z}}$ to transit from state s to state s^+ under action α_{+z}:

$$\mathcal{P}_{ss^+}^{\alpha_{+z}} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_A \text{ and each } d \in S_D \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-(m_{ready}+m_{lat_1})\times d\}\} \\ l' & = \max\{0, n+a-(m_{ready}+m_{lat_1})\times d - b\} \end{array}\right.} P_A(i)\times P_D(d) \tag{4.34}$$

And we have an immediate cost of $\mathcal{C}_s^{\alpha_{+z}}$ to transit from state s to state s^+ under action α_{+z}:

$$\begin{aligned} \mathcal{C}_s^{\alpha_{+z}} &= n\times c_N + l\times c_L + (m_{lat_1}+m_{ready})\times c_M \\ &\quad + \Big(\min\{z, m_{stop}\} + \sum_{i=2}^{k} m_{lat_i}\Big)\times \frac{c_{on}}{k} \end{aligned} \tag{4.35}$$

3. action α_{-z} moves the system to the state s^-:

$$s^- = \left\{\begin{array}{l} (m_{stop}+\min\{z, m_{ready}\}, 0, m_{lat_k}, m_{lat_{k-1}}, \ldots \\ \quad \ldots, m_{lat_i}, \ldots, m_{lat_2}, m_{lat_1}+\max\{m_{ready}-z, 0\}) \\ (n', l') \end{array}\right. \tag{4.36}$$

so we have a probability of $\mathcal{P}_{ss^-}^{\alpha_{-z}}$ to transit from state s to state s^- under action α_{-z}:

$$\mathcal{P}_{ss^-}^{\alpha_{-z}} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_A \text{ and each } d \in S_D \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-(m_{lat_1}+\max\{m_{ready}-z, 0\})\times d\}\} \\ l' & = \max\{0, n+a-(m_{lat_1}+\max\{m_{ready}-z, 0\})\times d - b\} \end{array}\right.} P_A(i)\times P_D(d) \tag{4.37}$$

and we have an immediate cost of $\mathcal{C}_s^{\alpha_{-z}}$ to transit from state s to state s^- under action α_{-z}:

$$\begin{aligned} \mathcal{C}_s^{\alpha_{-z}} &= n\times c_N + l\times c_L + (m_{lat_1}+\max\{m_{ready}-z, 0\})\times c_M \\ &\quad + \sum_{i=2}^{k} m_{lat_i}\times \frac{c_{on}}{k}. \end{aligned} \tag{4.38}$$

Notice that n' (resp. l') is the new number of waiting (resp. rejected) jobs, and the immediate cost includes two parts. The QoS part which includes the cost due to waiting and rejected jobs. The power part composed of energy consumed for running operational servers and energy used by servers which are in the switching-on latency period. Table 4.14 summarizes parameters used in our model and our MDP formulation.

Table 4.14: Model and MDP Parameters.

Parameters	**Description**
h	duration of analysis
k	latency period
b	buffer size
$\mathcal{M}$	total number of servers
$\mathcal{H}_D$	histogram modeling the processing capacity of a server
$\mathcal{H}_A$	histogram of job arrivals
m_{ready}	number of operational servers
m_{lat_i}	number of servers in latency level i
n	number of waiting jobs
l	number of rejected jobs
$\mathcal{S}$	set of all possible states
$\mathcal{A}$	set of all possible actions
s	$s = ((m_x : x \in \mathcal{L}), (n,l))$ system state
s_0	$s_0 = ((\mathcal{M}, 0, ..., 0), (0,0))$ starting state
α_0	action to keep the same number of operational servers
α_{+z}	action to switch-on z additional server
α_{-z}	action to switch-off z server
$\mathcal{P}^{\alpha}_{ss'}$	probability of transition from s to s' under action α
$\mathcal{C}^{\alpha}_{s}$	immediate cost from s under action α

4.3.2.2 Space state complexity

Theorem 4.8. *The number of states of the MDP is in* $O\Big(b \times \max(S_A) \times \mathcal{M}^k\Big)$.

Proof. Every state of the MDP includes three parts:

1. the number of waiting jobs in the buffer which is bounded above by b,
2. the number of rejected jobs which can be at most equals to the maximum number of arrival jobs given by $\max(S_A)$, and
3. as we deal with k level of latency, the set of number of servers for every latency level in $\mathcal{L}$. Each number is between 0 and $\mathcal{M}$.

So, $|\mathcal{S}|$ is bounded above by $(b+1) \times (\max(S_A)+1) \times \mathcal{M}^{k+2}$. In fact the number of states of the MDP is in $O\Big(b \times \max(S_A) \times \mathcal{M}^k\Big)$. □

Theorem 4.9. *The number of transitions of the MDP is in*

$$O\Big(b \times |S_A| \times |S_D| \times \max(S_A) \times \mathcal{M}^k\Big).$$

Proof. From each state of the MDP we have at most $(2 \times \mathcal{M} + 1)$ actions, and each action leads to a number of transitions equals to $|S_A| \times |S_D|$ (one transition for each bin in the support of the arrivals distribution combined with each bin in the support of the service rates distribution). Then, the number of transitions of the MDP is in $O\Big(b \times |S_A| \times |S_D| \times \max(S_A) \times \mathcal{M}^k\Big)$. □

In order to solve the previous optimization problem we use a similar approach as explained in Sections 4.1.2.3 and 4.2.2.3.

4.3.3 Example

To illustrate our formalization Let us show an example of an MDP modeling a very simple data center of two servers $\mathcal{M} = 2$ with a latency period of two units of time $k = 2$ with a buffer size equals two $b = 2$. Additionally, to keep the example simple, we set service rate of each machine as a histogram with only one bin: $P_D(1) = 1$ (one server processes one job every slot). Also job arrivals are modeled by an histogram with only one bin $P_A(1) = 1$ (the system receives one job every slot). Listing 4.9 gives the associated PRISM specification, and Figure 4.26 shows the MDP obtained.

Listing 4.9: Example of PRISM specification for a model with latency.

```
mdp
const int k=2, B=2, m=2, d=2, A0=1;
const double cN=1, cL=1, cM=1, cON=1, p0=1.0;
module system1
off  : [0..m ] init m;
M    : [0..m ] init 0;
set2 : [0..m ] init 0;
set1 : [0..m ] init 0;
N    : [0..B ] init 0;
L    : [0..A0] init 0;
[a_5] M>=5 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1-5))&(set1'=set2)&(set2'=0)&
(off'=min(m,off+5))&(L'=max(0,N+A0-M*d-B));

[a_4] M>=4 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1-4))&(set1'=set2)&(set2'=0)&
(off'=min(m,off+4))&(L'=max(0,N+A0-M*d-B));
```

```
[a_3] M>=3 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1-3))&(set1'=set2)&(set2'=0)&
(off'=min(m,off+3))&(L'=max(0,N+A0-M*d-B));

[a_2] M>=2 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1-2))&(set1'=set2)&(set2'=0)&
(off'=min(m,off+2))&(L'=max(0,N+A0-M*d-B));

[a_1] M>=1 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1-1))&(set1'=set2)&(set2'=0)&
(off'=min(m,off+1))&(L'=max(0,N+A0-M*d-B));

[a0] off>=0 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1))&(set1'=set2)&(set2'=0)&
(off'=off-0)&(L'=max(0,N+A0-M*d-B));

[a1] off>=1 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1))&(set1'=set2)&(set2'=1)&
(off'=off-1)&(L'=max(0,N+A0-M*d-B));

[a2] off>=2 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1))&(set1'=set2)&(set2'=2)&
(off'=off-2)&(L'=max(0,N+A0-M*d-B));

[a3] off>=3 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1))&(set1'=set2)&(set2'=3)&
(off'=off-3)&(L'=max(0,N+A0-M*d-B));

[a4] off>=4 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1))&(set1'=set2)&(set2'=4)&
(off'=off-4)&(L'=max(0,N+A0-M*d-B));

[a5] off>=5 -> p0:(N'=min(B,max(0,N+A0-M*d)))&
(M'=min(m,M+set1))&(set1'=set2)&(set2'=5)&
(off'=off-5)&(L'=max(0,N+A0-M*d-B));
endmodule
rewards "r"
[a_5] true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a_4] true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a_3] true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a_2] true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a_1] true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a0]  true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a1]  true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a2]  true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a3]  true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a4]  true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
[a5]  true : M*cM+N*cN+L*cL+(set1+set2)*cON/k;
endrewards
```

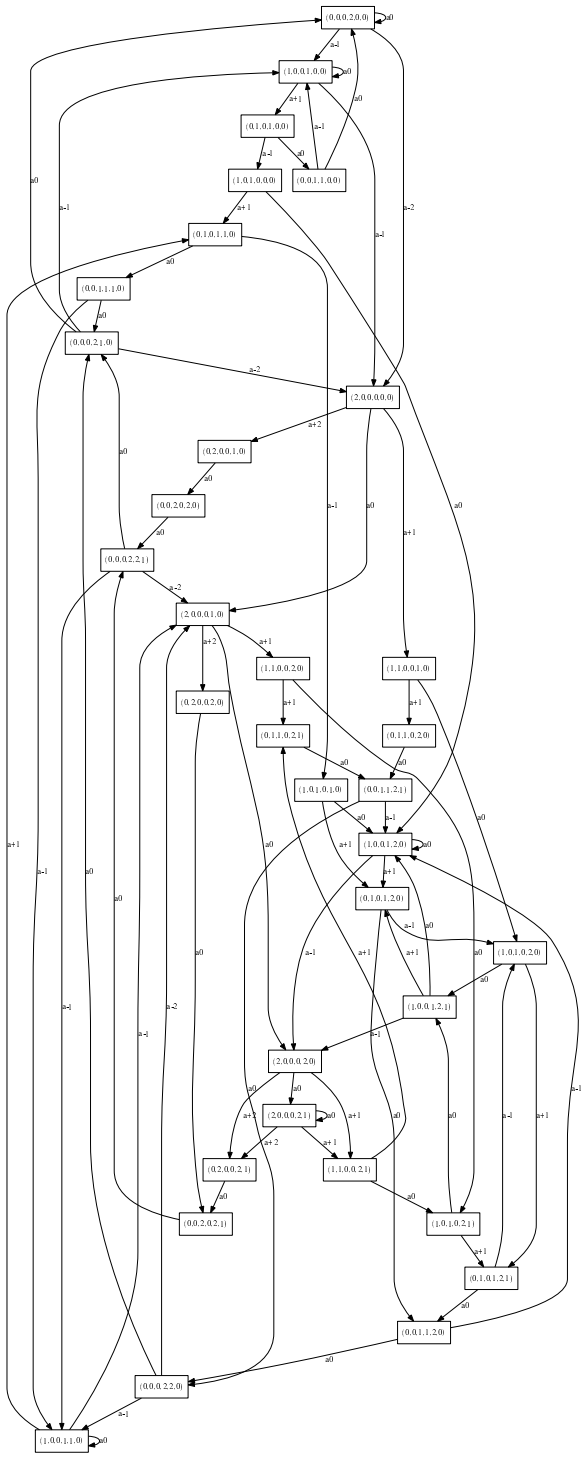


Figure 4.26: MDP example for a data center of only two machine with latency $k = 2$ and $b = 2$.

4.3.4 Experimental application

As previously this section presents a case study that uses real traffic traces based on the open *cluster-data-2011-2* trace [Wil11, RWH11]. Also as already said before, the specification of our latency model is huge and not obvious to write in PRISM (see Theorems 4.8 and 4.9). In fact we have coded a tool that helps us to generate automatically the PRISM specification file.

Figure 4.24 summarizes the results obtained for the following configuration:

Table 4.15: Parameters of the numerical analysis.

b	$\mathcal{M}$	$\mathbb{E}(\mathcal{H}_A)$	$\mathcal{H}_D$	c_M	c_{On}	c_N	c_L	k	h
4-67	5	7	Δ_2	1	1	1	1	2-5	100 slot

Figure 4.27 shows the result of experiments in which we are analyzing the total cost over 100 unit of time when varying the buffer size b and latency k. We observe that in general the total cost increases when b is less than some value b_k which depends on k.

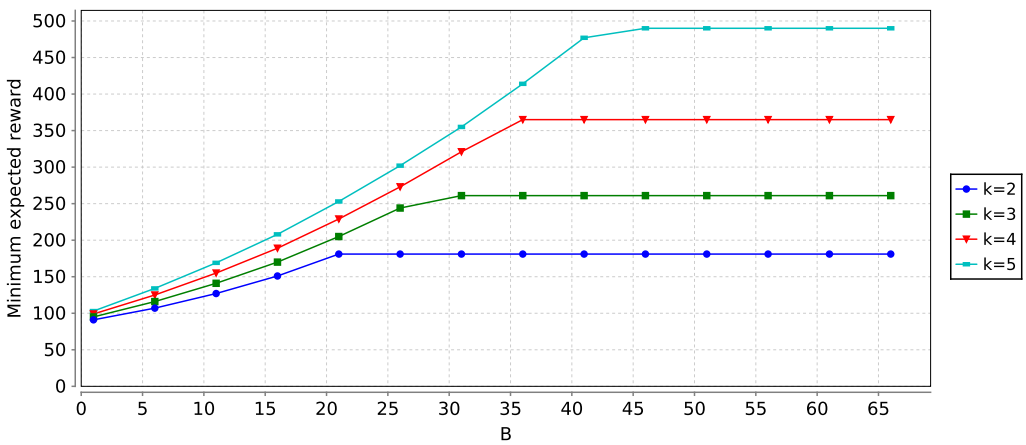


Figure 4.27: Total cost when varying buffer size b for different values of latency k.

However, when b is bigger than b_k the total cost seems to be convergent for any configuration (see Section 4.2.5 for the explanation).

Another related observation is the fact that b_k grows when k increases. For example, for a system with $k = 4$, the total cost becomes independent from the latency period when $b > b_k = b_4 = 36$, however for a system with $k = 2$, the total cost becomes independent from the latency period when only $b > b_k = b_2 = 21$.

It means that a data center with servers with longer period of latency have to be designed with larger buffer size.

A last observation is the fact that the total cost is more important when k increases. We can explain that from the fact that the system, during the latency period, accumulates more waiting jobs before the complete switching-on of the servers.

4.4 MDP for variable arrival distribution

Our analysis of the Google trace arrival jobs [Wil11, RWH11] shows that arrival traffic changes during time and presents a non negligible level of fluctuation daily along the month (see Figure 1.7 of Chapter 1), and hourly as shown in Figure 4.28 which present three samples of the arrival traffic for three different hours. For example the traffic between 6h and 7h for 27 May 2011 (blue curve), and the traffic between 10h and 11h for 19 May 2011 (red curve), are close and may be modeled by the same histogram. However it is clear that the traffic between 14h and 15h for 18 May 2011 (orange curve), must be modeled by another histogram as the traffic during this period is significantly higher than the two first ones. Figure 4.29 shows that even from a minute to another minute the traffic my change significantly. In fact, in contrary of the previous models of MDP where we consider that arrivals are modeled by the same histogram over time (see Figure 4.30), in this section, we will consider that each slot we may receive a number of arrival jobs modeled by a different histogram (see Figure 4.31). In order to model the case of arrival jobs distributions that changes over time, we may consider two kind of solution. The first one is an adaptation of the generic value iteration algorithm for the case of variable distribution arrival jobs. Thus we adapt the computation of the value function V to take into account the appropriate arrival job histogram for each slot t. The second solution is more adapted to be implemented in a model checker in-which the value iteration algorithm is predefined and encoded in the kernel of the model checker as in PRISM or MdpToolBox (see Annex D for more details). Notice that both approaches give rise to the same number of transitions (see Table 4.16).

Table 4.16: Approaches comparison.

	States	**Transitions**	**Advantage**
First approach	$O(\mathcal{M} \times b)$	$O(\mathcal{M}^2 \times b \times h \times \lvert S_A \rvert)$	Less states
Second approach	$O(\mathcal{M} \times b \times h)$	$O(\mathcal{M}^2 \times b \times h \times \lvert S_A \rvert)$	Use of PRISM

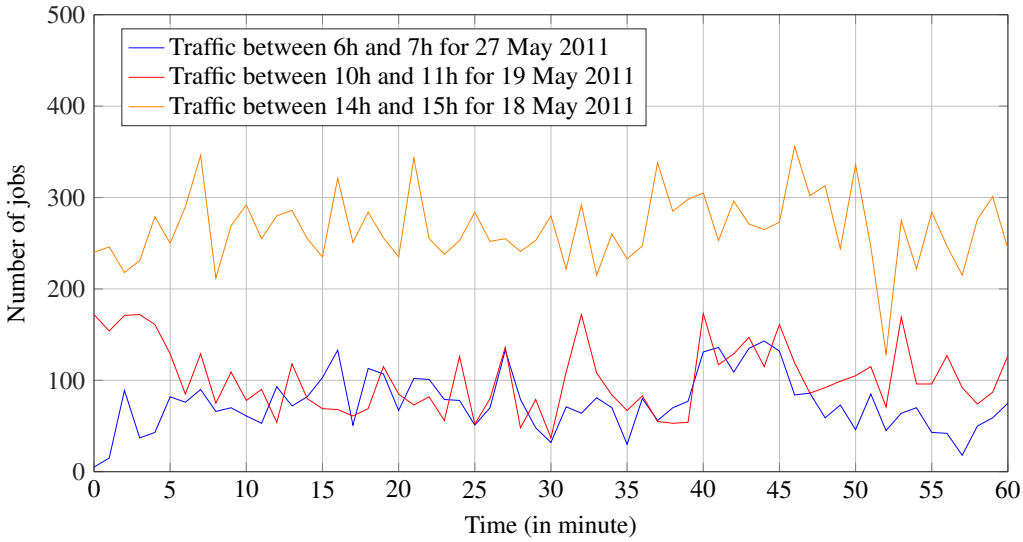


Figure 4.28: Arrival traffic for different hours.

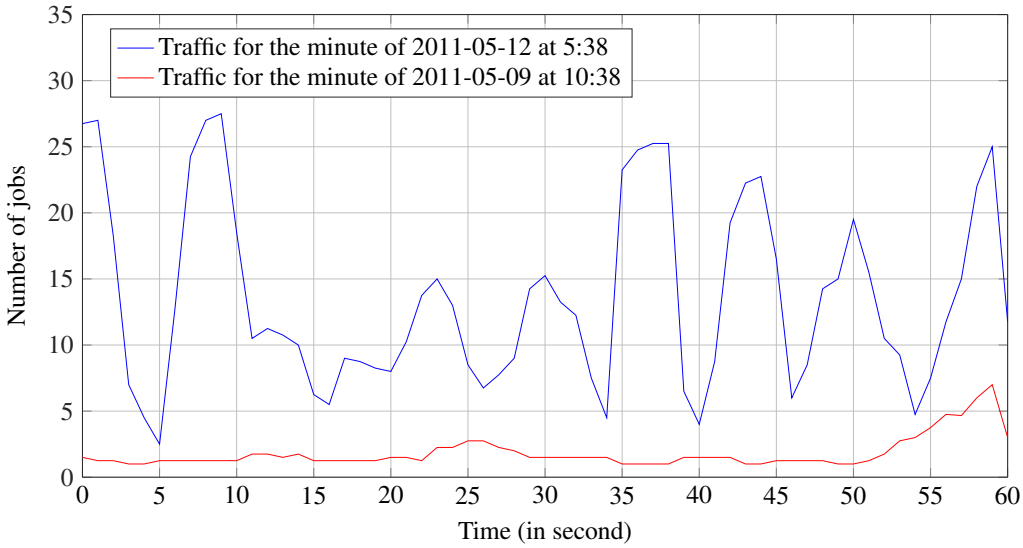


Figure 4.29: Arrival traffic for different minutes.

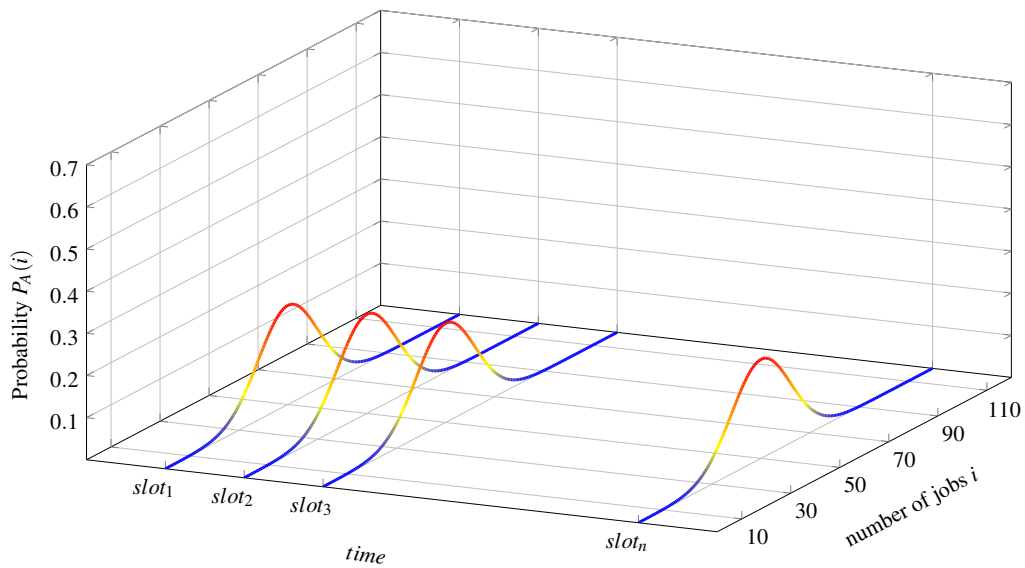


Figure 4.30: Illustration of the generic i.i.d. arrival jobs: each slot we receive a number of jobs with the same distribution over time.

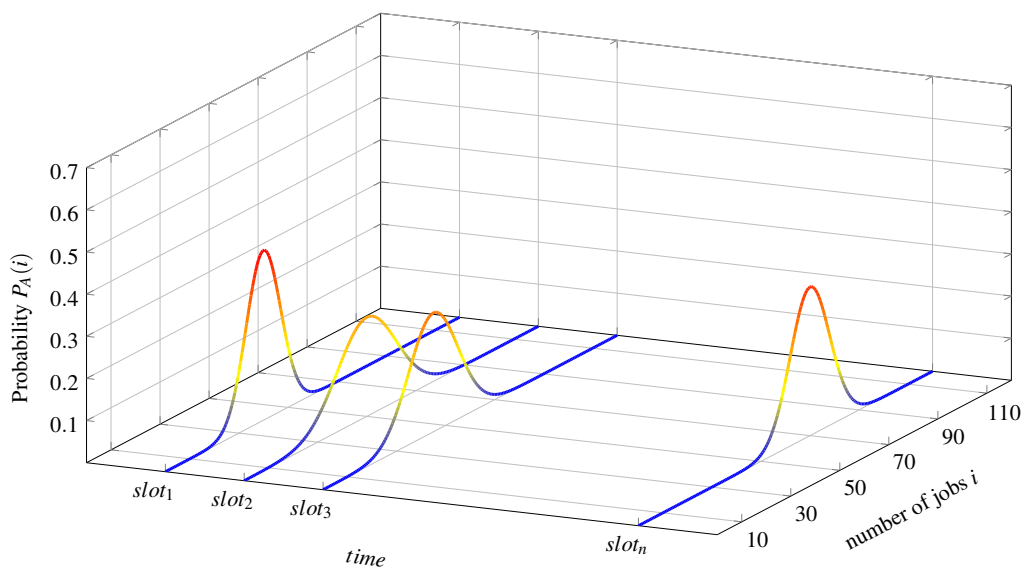


Figure 4.31: Illustration of a generic variable arrival jobs: each slot we may receive a number of jobs with a different distribution. Which means that every slot we may deal with a different distribution.

In the following we give a summary of the modelization that can not be written in a model checker in-which the value iteration algorithm is predefined as in PRISM or MdpToolBox, because it is based on a modified version of the generic value iteration algorithm that is able to take into account the fact that each slot we may have a different distribution for the arrival jobs process.

Let $\mathcal{H}_{A(t)} = (S_{A(t)}, P_{A(t)})$ be the histogram used to model the arrival of jobs during slot t. Assume that the optimization will be done over a period of h slots (the horizon). Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ be an MDP where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P} = \{\mathcal{P}_t \mid t \in [0..h]\}$ is a set of transition probability matrices where $\mathcal{P}_t$ is the transition probability matrix during slot t, and finally $\mathcal{C}$ is the immediate cost of each action. The state of the system is defined by the couple (m,n) where m is the number of operational servers, n is the number of waiting jobs. However t the age of the system, will be associated to the probability function $\mathcal{P}$. Indeed the state space $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{(m,n) \mid m \in [0..\mathcal{M}] \text{ and } n \in [0..b]\}. \tag{4.39}$$

At the beginning of each slot, and based on the current state of the system, an action $\alpha_j \in \mathcal{A}$ will be made to determine how many servers will be operational during the current slot. In fact the action space $\mathcal{A}$ is defined as $\mathcal{A} = \{\alpha_j \mid 0 \leq j \leq \mathcal{M}\}$, where action α_j consists in keeping exactly j operational servers during the current slot. We have a probability of $\mathcal{P}^{\alpha_j}_{t,ss'}$ to move from state $s = (m,n)$ to $s' = (j,n'))$ under action α_j, during slot t. This probability is defined as:

$$\mathcal{P}^{\alpha_j}_{t,ss'} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_{A(t)} \text{ and each } d \in S_D \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-d\times j\}\} \end{array}\right.} P_{A(t)}(i) \times P_D(d). \tag{4.40}$$

Consequently moving from state $s = (m,n)$ to $s' = (j,n')$ under action α_j induces immediately a cost $\mathcal{C}_s^{\alpha_j}$ defined as:

$$\mathcal{C}_s^{\alpha_j} = j \times c_M + \max\{0, j-m\} \times c_{On} + n \times c_N. \tag{4.41}$$

Algorithm 7 is an adaptation of the generic value iteration algorithm to include the fact that arrival jobs distribution may change over time.

Now we will evaluate the space complexity of the data structure behind our MDP formulation.

Theorem 4.10. *The number of states of the MDP is in $O(\mathcal{M} \times b)$.*

Algorithm 7: Value iteration algorithm for MDP.

```
Data: 𝓜, 𝓒, 𝓢, horizon h
Data: 𝓟_0, 𝓟_1, ···, 𝓟_t, ···, 𝓟_h
Result: (V_h, π*_h)
foreach s ∈ 𝓢 do
    V_0(s) = 0
end
for k ← 1 to h do
    foreach s ∈ 𝓢 do
        (m, n) ← s
        for a_j ← a_0 to a_𝓜 do
            C_s^{α_j} ← j × c_M + max{0, j − m} × c_On + n × c_N
            Q_k(s, a_j) ← C_s^{α_j} + Σ_{s'} 𝓟^{α_j}_{h−k,ss'} V_{k−1}(s')
        end
        π*_k(s) ← argmin_a Q_k(s, a)
        V_k(s) ← Q_k(s, π*_k(s))
    end
end
return (V_h, π*_h)
```

Proof. Every state of the MDP includes two element:

1. the number of operational servers which is between 0 and $\mathcal{M}$,
2. the number of waiting jobs in the buffer which is bounded above by b, and

So, $|\mathcal{S}|$ is bounded above by $(\mathcal{M}+1)\times(b+1)$. □

Theorem 4.11. *The number of transitions of the MDP is in* $O(\mathcal{M}^2 \times b \times h \times |S_A|)$.

Proof. From each state of the MDP we have at most $(\mathcal{M}+1)$ actions, and each action leads to a number of transitions equals to $|S_A|$ (one transition for each bin in the support of the arrival distribution). However as we need to consider the set of $(h+1)$ matrices of probability transition the total number of transitions is $|\mathcal{S}| \times |S_A| \times (h+1)$. In fact, as the number of states was already evaluated in Theorem 4.10, we deduce that the number of transitions is bounded above by $\mathcal{M} \times b \times (\mathcal{M}+1) \times |S_A| \times h$ which is in $O(\mathcal{M}^2 \times b \times h \times |S_A|)$. □

Theorem 4.12. *The computation-time to find the optimal strategy with the value iteration algorithm is in* $O(h \times b^2 \times \mathcal{M}^3)$.

Proof. As we know, at each iteration, for each of $|\mathcal{S}|$ states, the value iteration algorithm computes expectation for $|\mathcal{A}|$ actions. In general, each expectation takes $O(|\mathcal{S}|)$ time, however in this modelization each expectation takes only $O(\mathcal{M} \times b)$ time. In fact the total time complexity is $O(h \times |\mathcal{S}| \times |\mathcal{A}| \times (\mathcal{M} \times b))$ which equals $O(h \times b^2 \times \mathcal{M}^3)$ (see proof of Theorem 4.3 for more details). □

In fact, even if the number of servers in the data center is low, if we want an optimization over an important period of time (a big horizon h), the MDP structure will be huge and parsing it to find the optimal solution becomes very important.

4.5 Lessons learn from this work

This chapter presents a discrete-time Markov decision process (MDP) model for optimal management of a data center when considering separately different aspects: homogeneity, monotony, heterogeneity, and latency. The objective was to minimize energy consumption and Quality of Service (QoS) costs. Job arrivals and service rates are modeled by a general discrete probability distribution, which can be estimated from real data through a histogram. The optimal control policy is computed by the value iteration algorithm and used to define the Dynamic Power Management that ensures the trade-off between QoS and energy consumption. We

prove that in general the optimal policy is not monotone. Consequently, the optimal policy cannot be designed as a double-threshold structure.

For the homogeneous model theoretical and experimental results show that depending on the values of the system parameters the optimal policy is monotone or not. Therefore we can claim that strategy based on double-threshold structure leads only to a sub-optimal policy. Results show that MDP leads to the optimal policy when saving a significant amount of energy. But, it needs more computation time and more memory space to analyze and solve the optimization problem. We observe that the threshold based approach computes the optimization strategy faster, and uses a small amount of space memory. Otherwise we show that the size of an MDP model is important and in the following chapter, we consider to apply some heuristic algorithm to approximate efficiently the optimal policy by avoiding state space explosion.

For the heterogeneous model we proved that neither monotony nor isotony properties holds for optimal heterogeneous policy. Consequently, the optimal policy cannot be designed as a simple double-threshold structure. Additionally results show that the size of heterogeneous model is bigger than the size of homogeneous model. However, from experimental results, it seems that increasing server heterogeneity leads to more potential energy savings.

For the model with latency results show that the size of model is huge, more precisely the size is an exponential of k the period of latency. However, from experimental results, it seems that increasing buffer size leads to more potential energy saving when the latency is big.

Thus, in order to deal with the space and time complexity problem, we suggest in the next chapter what we call the greedy-window algorithm that allows to find a sub-optimal strategy better than that produced when considering special mechanism like the threshold approaches. And more important, unlike the MDP approach, this greedy-window algorithm does not require the building of the structure including all possible strategies and it is able to give a strategy very close to the optimal strategy with small space and time complexities.

Chapter 5

Greedy-window optimization algorithm

PREVIOUS chapters show that computing the optimal strategy, which requires passing through an MDP, is often difficult even impossible to achieve for large data centers. Especially if we take into account real aspects like servers heterogeneity or latency. Moreover, when we consider a data center with arrival jobs modeled by a distribution which changes over time, the MDP structure becomes huge because the space state size will depend on the horizon h. Indeed, the problem comes mainly from the exponential size of the MDP structure and then the time needed to go through it to find the optimal strategy. So the MDP complexity is big both in space and in computation-time. In this chapter we propose an algorithm that allows to find a sub-optimal strategy which is better than the strategy produced by the threshold approaches. More important, unlike the MDP approach, this algorithm does not require the entire construction of the structure including all possible strategies. Thus our algorithm gives a strategy close to the optimal strategy with small space and time complexities.

5.1 Problem specification

Let DC be a data center composed of $\mathcal{M}$ identical servers. DC receives jobs requesting the offered service. The maximal number of jobs that can be served by one server in one slot is assumed to be constant and denoted by d. So the service rate is modeled by a Dirac histogram $\mathcal{H}_D = \Delta_d$. We assume that the arrival process changes between periods. This allows us to model for instance hourly or daily variations of the job arrivals. Suppose that the analysis will be done over h a whole

period of time. The number of jobs arriving to the data center per slot during time slot t is modeled by a histogram $\mathcal{H}_{A(t)}$ where $P_{A(t)}(i)$ gives the probability to have i arrival jobs during slot t. Note that we assume that arrivals of jobs are independent, and their distribution is obtained from real traces, empirical data, or incoming traffic measurements. In this manner, the queuing model is a batch arrival queue with constant services and finite capacity buffer b (buffer size). In this chapter we assume that a server needs no time to be switched on or off. We assume also that the switching-on leads instantaneously to an additional energetic cost c_{on} however the switching-off takes place without consuming any additional amount of energy. Otherwise we consider that, in average, one on-server costs c_M unit per slot. During slot t, the number of waiting jobs in the buffer is denoted by $n(t)$ and its distribution by $\mathcal{H}_{N(t)}$. The number of operational servers is denoted by $m(t)$. The number of rejected (lost) jobs is denoted by $l(t)$ and its distribution by $\mathcal{H}_{L(t)}$. We assume that initially the number of operational servers and the number of waiting jobs are 0. The maximal number of servers that can be operational is $\mathcal{M}$.

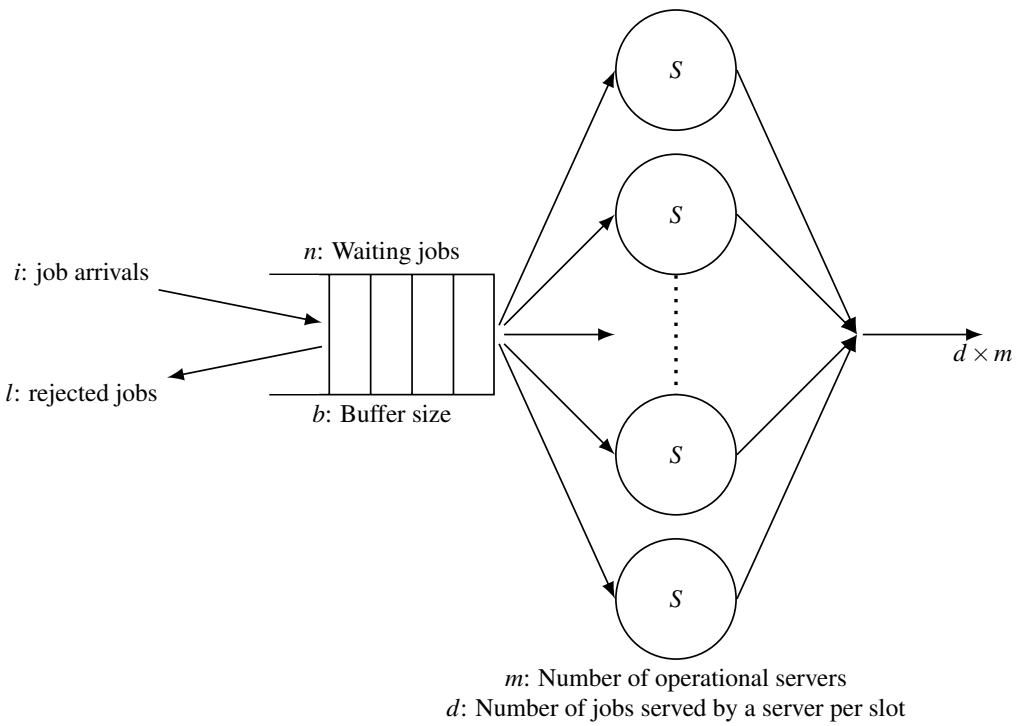


Figure 5.1: Illustration of the queuing model.

The histogram operators given in Chapter 1.3, are used to compute inductively

the evolution of distributions $\mathcal{H}_N$:

$$\mathcal{H}_{N(t+1)} \leftarrow MIN_b\big((\mathcal{H}_{N(t)} \oplus \mathcal{H}_{A(t)}) - (d \times m)\big) \tag{5.1}$$

where the arrival jobs are added to the system (buffer and free operational servers), a maximum of $d \times m$ jobs will be on servers to be processed, and the rest of jobs will be rejected (see Section 1.3.1 for the exact order of events). In fact, the distribution of the number of lost jobs is computed as:

$$\mathcal{H}_{L(t+1)} \leftarrow \big(\mathcal{H}_{N(t)} \oplus \mathcal{H}_{A(t)} - (d \times m + b)\big). \tag{5.2}$$

5.1.1 Energy and Performance metric

Energy consumption takes into account the state of each server and its transitions:

- Each server consumes some units of energy per slot when a server is operational and it costs an average of $c_M \in \mathbb{R}^+$ monetary unit.
- when switching a server on, the server consumes an additional amount of energy that costs an average of $c_{On} \in \mathbb{R}^+$ monetary unit.
- A server consumes a very low amount of energy when it is turned off in fact it is assumed that a non operational server does not consume energy $c_{off} = 0$.

The total consumed energy is the sum of all units of energy consumed among a specific period.

As QoS metric, we take into account the number of rejected jobs. Each rejected job causes a penalty that costs c_L. Additionally, we take into account the number of waiting job. Each waiting job causes a penalty that costs c_N for every slot. In fact the cost of QoS is computed as:

$$\sum_{i \in S_{L(t)}} P_{L(t)}(i) \times i \times c_L + \sum_{i \in S_{N(t)}} P_{N(t)}(i) \times i \times c_N. \tag{5.3}$$

5.1.2 Objective function

A dynamic power management system consists in doing each slot an action (turn on or turn-off a specific number of servers) in order to adapt the number of operational servers to incoming job changes. A good strategy consists in finding a sequence of actions to reduce the overall accumulated cost (energetic cost plus performance

cost). The expected cost generated for each slot is a linear combination of energetic and QoS costs called objective function:

$$c(t) = \sum_{i \in S_{L(t)}} P_{L(t)}(i) \times i \times c_L \tag{5.4}$$

$$+ \sum_{i \in S_{N(t)}} P_{N(t)}(i) \times i \times c_N \tag{5.5}$$

$$+ m(t) \times c_M + \max\{0, m(t) - m(t+1)\} \times c_{On} \tag{5.6}$$

where $m(t)$ is the current number of operational servers, and $m(t+1)$ is the number of operational servers after doing an action.

5.2 Optimization approach

Here we explain our greedy-window approach to optimize energy and QoS. Based on some elements of the system like the number of job arrivals, the number of waiting jobs, the number of rejected jobs and the number of servers, the method consists in turning on or off, each slot, an optimal number of servers to minimize energy consumption and maximize the QoS. So we will need to evaluate every slot the number of waiting and rejected jobs. Notice that parameters $n(t)$, $m(t)$ and $l(t)$ are correlated, gain on energy consumption leads to a degradation on the QoS and vice versa. In the following we will give the details of our greedy-window optimization algorithm. Before that, let us show other solutions:

1. The first one based on the concept of *rolling horizon procedure* which leads to an algorithm similar to the truncated value iteration algorithm that returns a sub-optimal policy. Unfortunately this solution still suffering from the state space explosion problem, and consumes a considerable amount of computation time.
2. The second solution based on a greedy algorithm which is very fast, need a small amount of memory to find a strategy, however the computed strategy is not close enough to the optimal one.
3. Finally we explain the greedy-window algorithm which allows us to compute a sub-optimal strategy very close to the optimal one in a small amount of space memory and computation time.

5.2.1 Truncated value iteration

The MDP optimization consists in exploring all the possible paths of length h from the initial state of the system until the end of the period of analysis (See the

pink paths in Figure 5.2). As seen before, the use of the value iteration algorithm reduce the time and space used for exploring the paths to a polynomial complexity, unfortunately we still consume a considerable amount of computation time and memory space. The idea of the rolling horizon procedure [WL11, IRCS13] is to not go through all the possible paths for the entire horizon h as MDP does. But instead, for each slot, we explore only the paths of length w, in fact the search is limited each slot to a depth called window of length w (see Figure 5.3). Notice that this concept is called sometime *sliding look-ahead window* [GOA18] or *model predictive control* [AA11]. The use of the window concept in the MDP optimization gives the truncated MDP. In Algorithm 8, for each slot t, we look for the sequence of optimal actions $m_{t+0}, m_{t+1}, \ldots, m_{t+w}$ performed during the duration $[t..t+w]$ which minimizes the total accumulated cost over the same period $[t..t+w]$. The first action in this sequence m_{t+0} is the action chosen to be applied for the slot t. After this we go to the next slot $t+1$ and we redo the same thing: which means find the optimal sequence of action between $t+1$ and $t+1+w$ to find the best action to perform for the slot $t+1$. And so on for the rest of the slots.

Algorithm 8: Truncated value iteration algorithm.

```
Data: M, d, b, c_M, c_N, c_L, c_On, m(0) = 0                    1
Data: H_A(0), H_A(1), ···, H_A(h)                               2
Result: m(1), m(2), ···, m(h)                                   3
n(0) ← 0;                                                       4
l(0) ← 0;                                                       5
for t ← 0 to h do                                               6
    /* Use a value iteration algorithm with a horizon w to compute the   7
       optimal strategy over period [t..t+w]                    */
    (m_{t+0}, m_{t+1}, ..., m_{t+w}) ←                          8
      value_iteration(n(t−1), l(t−1), m(t−1), [t..t+w]);
    m(t) ← m_{t+0};                                             9
    Compute n(t);                                               10
    Compute l(t);                                               11
end                                                             12
```

Theorem 5.1. *The computation-time to find the strategy by the truncated value iteration Algorithm 8 is in* $O(h \times w \times \mathcal{M}^2 \times b \times |S_A|)$.

Proof. As we consider a window of w slots, m_k is the number of operational servers at slot k. For each slot form 0 to w we will try all possible combinations of operational servers for each slot. And then we evaluate the total cost over the window period for each possible combination. Finally we choose the combination

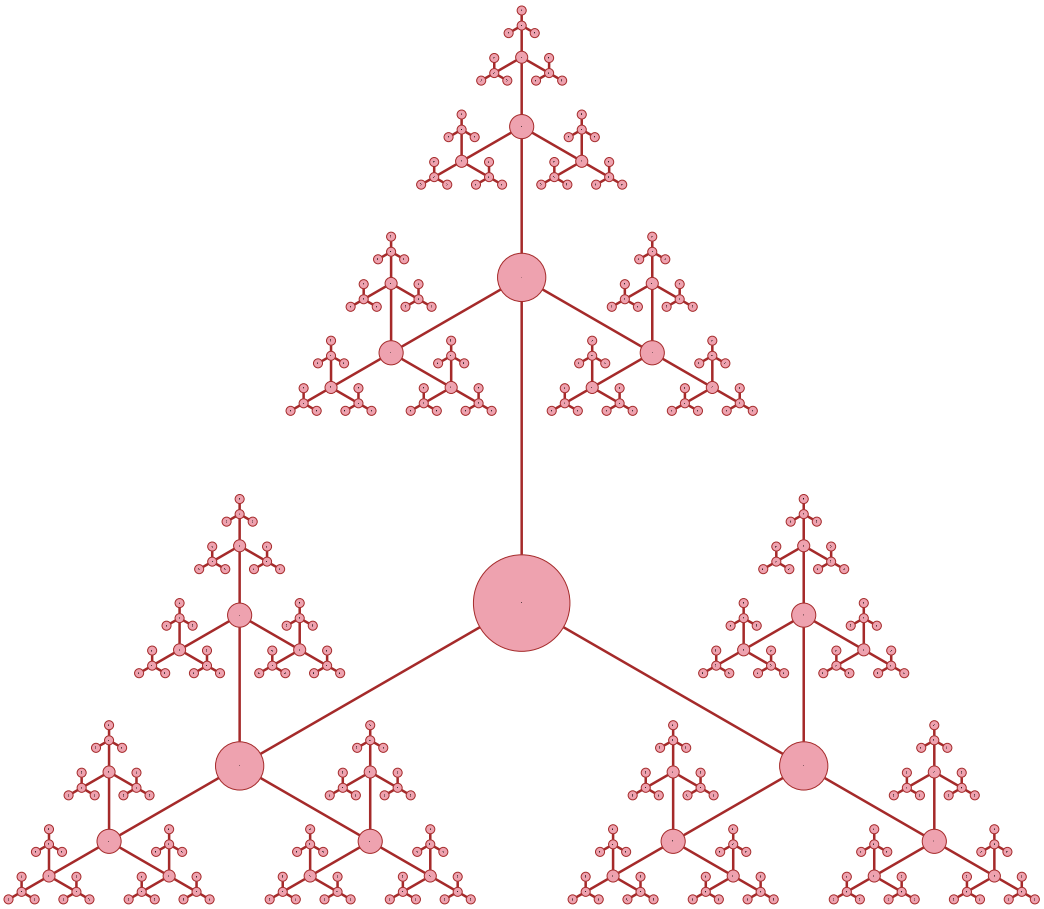


Figure 5.2: In the MDP optimization we have to visit all possible paths.

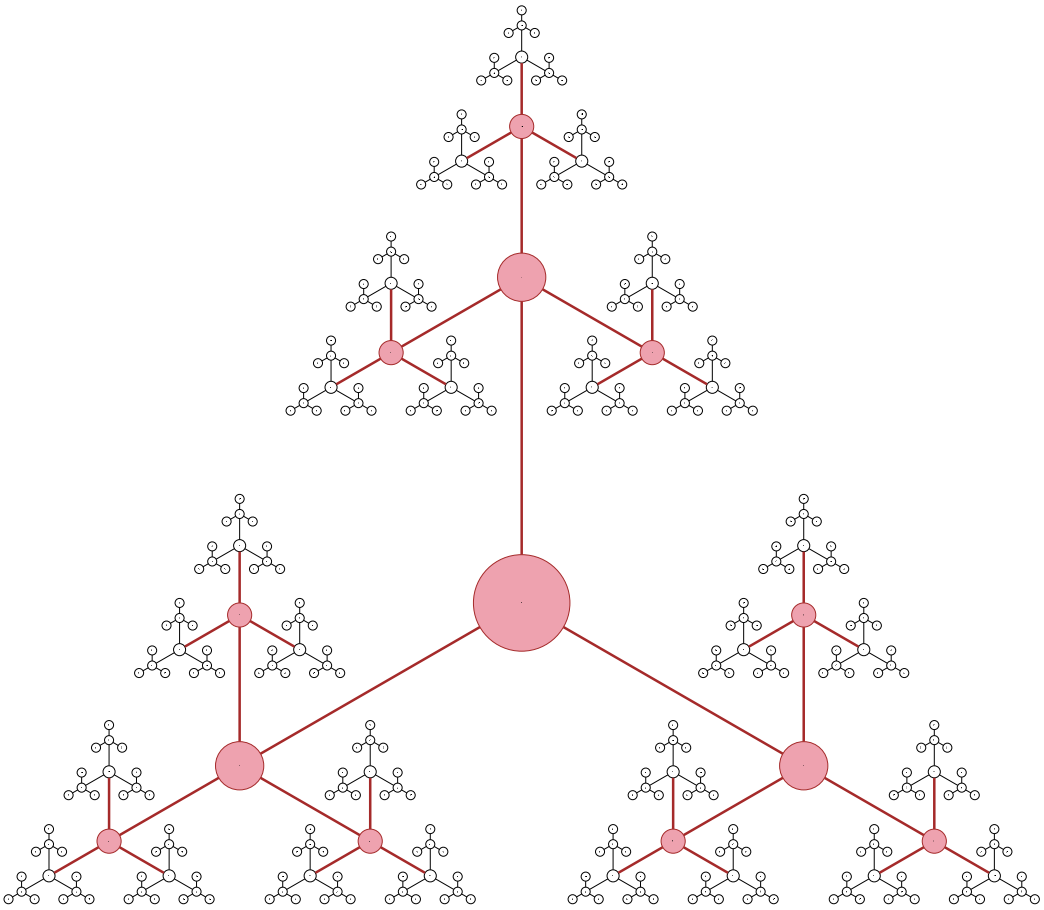


Figure 5.3: In the Truncated MDP optimization we have to visit all possible paths restricted to a window (depth) $w < h$. For this example the number of actions is 3, $w = 3$, $h = 5$, and the pink paths present the visited paths.

with the minimal cost and switch-on, during slot t, the m_0 servers associated with this best combination. In fact finding the sub-optimal action during slot t can be seen as an execution of a value iteration instance over the system for a horizon equals only w. And the space of states is restricted to a size of $|\mathcal{S}| = \mathcal{M} \times b$. So, for each slot (we have h slot), for all iterations of w, the value iteration algorithm is in $O(w \times \mathcal{M}^2 \times b \times |S_A|)$ (see Theorem 4.3 for more details). In fact the total time complexity is $O(h \times w \times \mathcal{M}^2 \times b \times |S_A|)$. □

5.2.2 Greedy optimization

As we have seen in previous chapters and previous section, MDP has the advantage of allowing us to get the optimal optimization strategy by browsing then evaluating the expected accumulated cost of each possible strategy. In fact we can choose, at the end, the strategy that has the minimum cost.

So as a first attempt we will consider Algorithm 9 that builds the optimization strategy by choosing at each slot the action that minimizes the immediate cost, ignore the rest of the actions and move to the next slot (see Figure 5.4). This method is in the class of greedy algorithms [Bla04].

Algorithm 9: Greedy optimization algorithm for one slot.

```
Data: M, d, b                                                    1
Data: m(t−1), H_N(t−1), H_A(t)                                   2
Data: c_M, c_N, c_L, c_On                                        3
Result: m(t), c(t)                                               4
cost_min ← +∞;                                                   5
for m ← 0 to M do                                                6
    N ← MIN_b((H_N(t−1) ⊕ H_A(t)) − d × m);                      7
    L ← (H_N(t−1) ⊕ H_A(t)) − (d × m + b);                       8
    m̃ ← max{0, m − m(t−1)};                                      9
    c ← c_M × m + Σ_{i∈S_N} P_N(i) × i × c_N + Σ_{i∈S_L} P_L(i) × i × c_L + c_On × m̃;   10
    if c < cost_min then                                         11
        cost_min ← c;                                            12
        servers_min ← m;                                         13
    end                                                          14
end                                                              15
c(t) ← cost_min;                                                 16
m(t) ← servers_min;                                              17
```

Knowing the parameters of the system at slot $(t-1)$, our strategy consists in determining the best number of servers to be switched on, in slot t, in order to minimize the cost. To do so, every slot t: for each possible value of $m(t) \in \{0, 1, \ldots, \mathcal{M}\}$, we compute $c(t)$ and then we return the value of $m(t)$ that minimizes the expected cost $c(t)$.

Table 5.1: Example of optimization. Suppose $\mathcal{M} = 10$, $b = 15$, $d = 3$, $c_M = 11$, $c_N = 5$, $c_L = 0$ and for $t = 1$ we have: $\mathcal{H}_N(t-1) = \Delta_5$ and $\mathcal{H}_A(t) = \Delta_7$. In this case our algorithm chooses $m(t) = 4$ this value leads to the minimal cost.

$m(t)$	0	1	2	3	4	5	6	7	8	9	10
$c(t)$	60	56	52	48	44	55	66	77	88	99	110

It is clear that this method is fast and does not cause the explosion of the number of states. However, accumulating the immediate minimum costs does not guarantee us to obtain a strategy close enough to the optimal strategy. So we introduced the notion of window.

Theorem 5.2. *The greedy Algorithm 9 is in $O(q\ \log(q))$ space complexity with $q = \max(S_A) + b$.*

Proof. At each step of the calculation, Algorithm 9 needs to keep track of the evolution of the distribution of the number of waiting jobs in the buffer. As the buffer is finite, the number of waiting jobs is bounded above by b and the algorithm can use only an array of length $O(b)$ to store the distribution probability of waiting jobs, and a space memory of $O(q\ \log(q))$ to compute the convolution, with $q = \max(S_A) + b$. The number of lost jobs is bounded above by $\max(S_A)$ and the algorithm can use only an array of length $O(\max(S_A))$ to store the distribution probability of lost jobs. So we need a total space of $q\ \log(q) + b + \max(S_A)$ which is in $O(q\ \log(q))$. □

Theorem 5.3. *The computation-time to find the strategy by Algorithm 9 is in $O(h \times \mathcal{M} \times q \times \log(q))$ where q is the sum of b and $\max(S_A)$.*

Proof. Algorithm 9 needs to be executed for h iterations. The for loop is of $\mathcal{M}$ iterations, and at each step of the calculation Algorithm 9 needs to compute the distribution of the number of waiting jobs in the buffer. Otherwise, for each slot we need to compute the distribution of the number of waiting jobs in the buffer by computing a convolution between $\mathcal{H}_A$ and $\mathcal{H}_N$. This task needs at most $b \times |S_A|$ operations, however, as being said before in Remark 1.1, by using the fast Fourier transform the convolution can be performed in a reduced time of $O(q \times \log(q))$ with $q = \max(\max(S_A), \max(S_N))$ where $\max(S_N) = b$. □

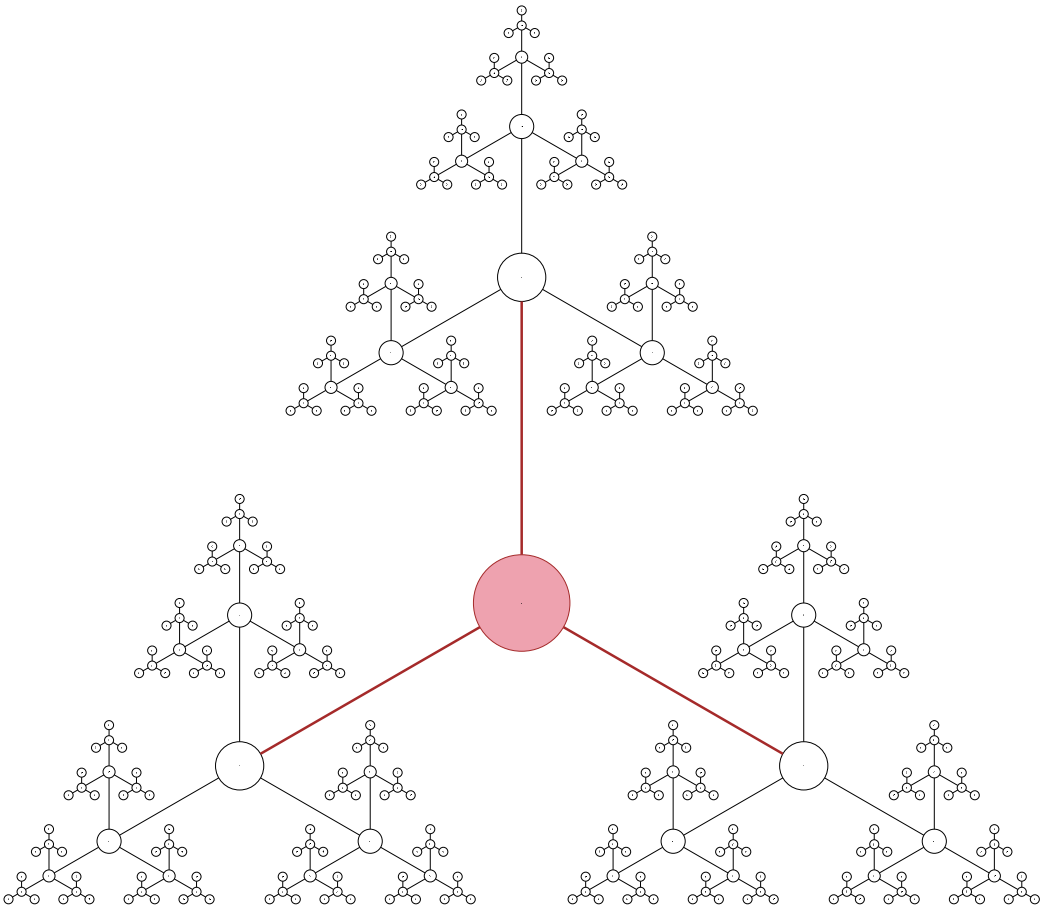


Figure 5.4: In the greedy optimization we have to visit all possible paths of length one. For this example the number of actions is 3, $h = 5$, and the pink paths present the visited paths.

5.2.3 Greedy-window optimization

It is clear that the decision is based on the analysis of one slot in the greedy algorithm. We can generalize our idea to do a decision after analyzing w successive slots as shown in Section 5.2.1.

This generalization will necessarily give a better decision. The bigger w is, the better optimization is, but the time of computation will be more important. Now, in order to avoid executing an entire instance of a like value iteration algorithm over w, we suggest our greedy-window algorithm that computes only a sub-optimal strategy over the window $[t..t+w]$ as follows.

Algorithm 10: Greedy-window optimization algorithm.

```
Data: M, d, b, c_M, c_N, c_L, c_On                                   1
Data: H_A(0), H_A(1), ···, H_A(h)                                    2
Result: m(1), ···, m(h)                                              3
m(0) ← 0, H_N(0) ← Δ_0, H_L(0) ← Δ_0;                                4
for t ← 1 to h do                                                    5
    cost_min ← +∞, servers_min ← 0;                                  6
    for m ← 0 to M do                                                7
        N ← H_N(t−1);                                                8
        L ← H_L(t−1);                                                9
        m̃ ← max{0, m − m(t − 1)};                                    10
        accumulated_cost ← m̃ × c_On;                                 11
        for k ← t to t + w do                                        12
            N ← MIN_b((N ⊕ H_A(t)) − d × m);                         13
            L ← (N ⊕ H_A(t)) − (d × m + b);                          14
            c ← c_M × m + Σ_{i∈S_N} P_N(i) × i × c_N + Σ_{i∈S_L} P_L(i) × i × c_L;   15
            accumulated_cost ← accumulated_cost + c;                 16
        end                                                          17
        if accumulated_cost < cost_min then                          18
            cost_min ← accumulated_cost;                             19
            servers_min ← m;                                         20
        end                                                          21
    end                                                              22
    m(t) ← servers_min;                                              23
    H_N(t) ← MIN_b((H_N(t−1) ⊕ H_A(t)) − d × m(t));                  24
    H_L(t) ← (H_N(t−1) ⊕ H_A(t)) − (d × m(t) + b);                   25
end                                                                  26
```

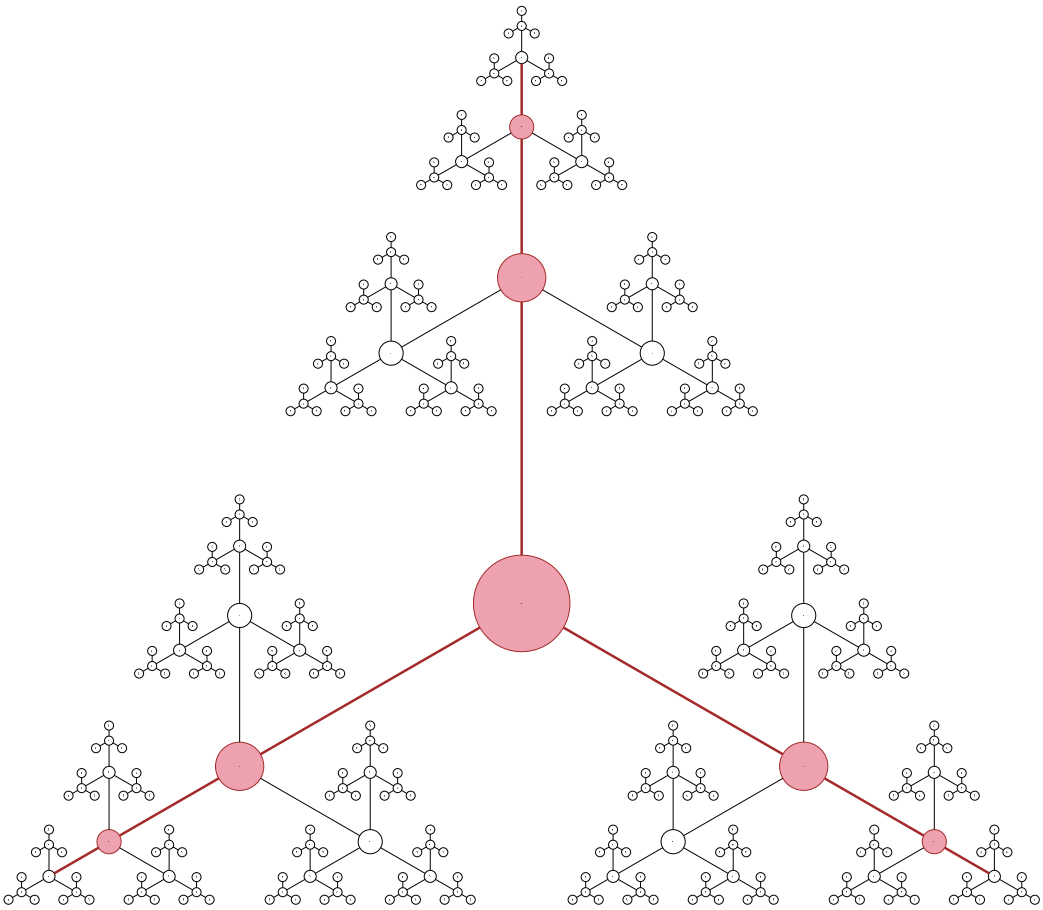


Figure 5.5: In the Greedy-window optimization we have to visit all directions restricted to a window (depth) $w < h$. For this example the number of actions is 3, $w = 3$, $h = 5$, and the pink paths present the visited paths.

This sub-optimal strategy is defined as the best strategy from the set of restricted strategies that **maintains**, over the window period $[t..t+w]$, the **same action** done at slot t. In other words, at slot t the algorithm tries to evaluate the impact of maintaining an action over a period of w, then chooses the action that minimizes the accumulated cost over the window.

Notice that the algorithm repeats this operation each slot and not each w. In other words the greedy-window algorithm, explores only directions (a direction is a path with the same action each slot) of length w (see Figure 5.5). Now, for the choice of the value of w, as shown in Figure 5.6, the cost is less optimized for a small value of w ($w < 5$), however it is more optimized for bigger values ($w \geq 5$). In fact we need to set w to a big value ($w \geq 5$) to guarantee that the strategy is

close enough to the optimal strategy, but w must not be too much big ($w \leq 10$) to maintain a fast computation.

For example Figure 5.6 shows the daily accumulated cost induced by the greedy-window strategy depending on w for various traffic arrivals issued from the Google trace [Wil11, RWH11] under the same settings as in Table 5.4. We observe that even for a small window size $w \ll h$ the cost is significantly improved.

Theorem 5.4. *The greedy-window Algorithm 10 is in $O(q \log(q))$ space complexity with $q = \max(S_A) + b$.*

Proof. See proof of Theorem 5.2. □

Theorem 5.5. *The computation-time to find the strategy by the greedy-window Algorithm 10 is in $O(w \times h \times \mathcal{M} \times q \times \log(q))$ where q is the sum of b and* $\max(S_A)$.

Proof. The first for loop is of h iterations, the second loop is of $\mathcal{M}$ iterations, and at each step of the calculation in the third for loop, Algorithm 10 needs to compute the distribution of the number of waiting/rejected jobs by computing a convolution between $\mathcal{H}_A$ and $\mathcal{H}_N$. □

This way of doing the optimization based on the greedy-window algorithm, is a very interesting compromise, because it avoids the phenomenon of explosion of memory space and computing time (the greedy-window algorithm time-complexity is linear in $\mathcal{M}$ compared to the MDP based algorithm which is quadratic in $\mathcal{M}$), while computing a strategy sufficiently close to the optimal strategy. In addition of all this, it gives the algorithm the ability to be executed in real time (online). Because for each slot it calculates immediately the action to be done (of course based on a projection of w slots) and it does not need to know the horizon h. Algorithm 11 is an example of a possible online version of the greedy-window algorithm. It uses a list called *arrival_history* that stores the number of arrivals for past slots. This list is used to recompute the histogram of arrivals every slot. Notice that this list should be restricted to a limited size by dropping the old arrivals in order to:

1. prevent the list from becoming large which can slow down the execution of the algorithm,

2. compute a histogram that is more representative for new arrivals.

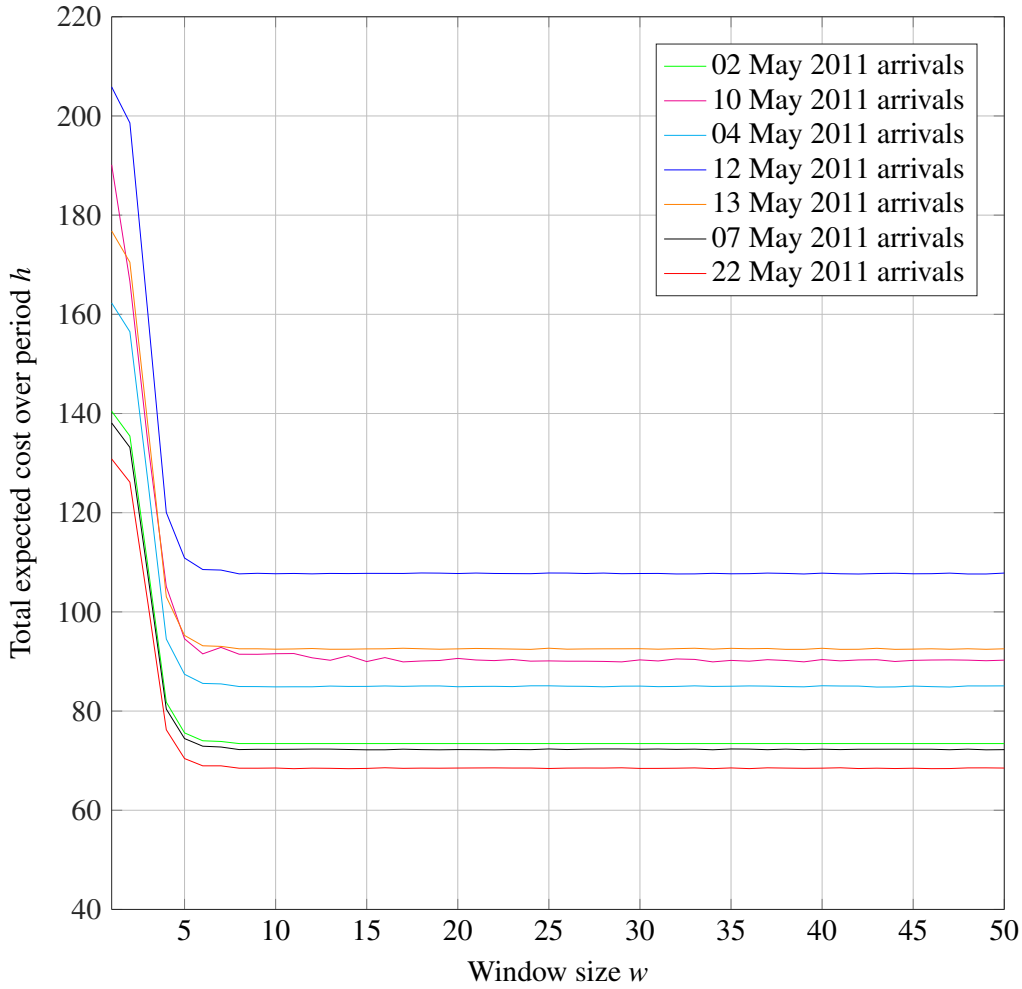


Figure 5.6: Relationship between the window size and the optimization efficiency of the greedy-window algorithm. For example, for the arrivals of the 2nd May 2011, the analysis of only some successive slots improves the cost significantly by about 50% compared to an analysis done with $w = 1$.

Algorithm 11: The online version of the greedy-window algorithm.

```
Data: 𝓜, d, b                                                          1
Data: c_M, c_N, c_L, c_On                                              2
arrival_history ← [];                                                  3
𝓗_A ← Δ_0;                                                             4
foreach beginning of slot t do                                         5
    cost_min ← +∞;                                                     6
    servers_min ← 0;                                                   7
    /* Request the state of the system                                 */ 8
    a ← the number of arrival jobs received in the last slot;          9
    n ← the number of waiting jobs;                                    10
    m ← the number of operational servers;                             11
    /* Add a to the arrival history                                    */ 12
    arrival_history ← arrival_history + [a];                           13
    /* Use the arrival history to update the histogram of arrivals     */ 14
    𝓗_A ← compute(arrival_history);                                    15
    for m ← 0 to 𝓜 do                                                  16
        N ← Δ_n;                                                       17
        m̃ ← max{0, m − m(t − 1)};                                      18
        accumulated_cost ← m̃ × c_On;                                   19
        for k ← 1 to w do                                              20
            N ← MIN_b((N ⊕ 𝓗_A) − d × m);                              21
            L ← (N ⊕ 𝓗_A) − (d × m + b);                               22
            c ← c_M × m + Σ_{i∈S_N} P_N(i) × i × c_N + Σ_{i∈S_L} P_L(i) × i × c_L;   23
            accumulated_cost ← accumulated_cost + c;                   24
        end                                                            25
        if accumulated_cost < cost_min then                            26
            cost_min ← accumulated_cost;                               27
            servers_min ← m;                                           28
        end                                                            29
    end                                                                30
    m ← servers_min;                                                   31
    set the number of operational servers to m;                        32
end                                                                    33
```

We notice that the greedy-window algorithm is designed to support an arrival process modeled by a histogram that may change at each slot, but it still works for the case where the arrivals are i.i.d. and even for this special case, our algorithm has a lower complexity compared to the MDP model (See theorems 4.1 and 4.3).

5.3 Experimental results

In the next sub-sections, we will use our greedy-window optimization algorithm to test, analyze and compare the evolution of the cost (energy consumption and QoS) for different configurations of the system. In order to reduce the number of parameters to explore in our experimentation we define the load factor of the system as follows:

$$\rho = \frac{\sum_{i \in S_A} P_A(i) \times i}{\mathcal{M} \times d}. \tag{5.7}$$

Table 5.2: Workload of system according to ρ.

ρ	[0;0.25[	[0.25;0.5[	[0.5;0.75[	[0.75;1[	≥ 1
	relaxed	moderate	comfortable	high	excessive

It is clear that ρ can be higher than 1 if the number of arrivals is more important than the number of processed jobs (for example if $\mathcal{M} = 10$, $d = 1$, and $\mathcal{H} = \Delta_{20}$ then $\rho = 2 \geq 1$).

5.3.1 Experiments for Dirac arrivals job

In order to show the different behaviors of the system under our greedy-window algorithm, let us first study the case where arrivals are modeled by a constant rate of job arrivals (it can be seen as a deterministic case):

$$a \in \mathbb{R} : \forall t \ : \ a(t) = a. \tag{5.8}$$

In this case the histogram of $a(t)$ is defined as:

$$a \in \mathbb{R} : \forall t \ : \ \mathcal{H}_{A(t)} = \Delta_a. \tag{5.9}$$

In this case the load factor is equal to:

$$\rho = \frac{\sum_{i \in S_A} P_A(i) \times i}{\mathcal{M} \times d} = \frac{a}{\mathcal{M} \times d}. \tag{5.10}$$

5.3.1.1 Optimization and load factor

Tests show that our algorithm turns on, eventually, a number of servers proportional to the system load factor (see Figure 5.7).

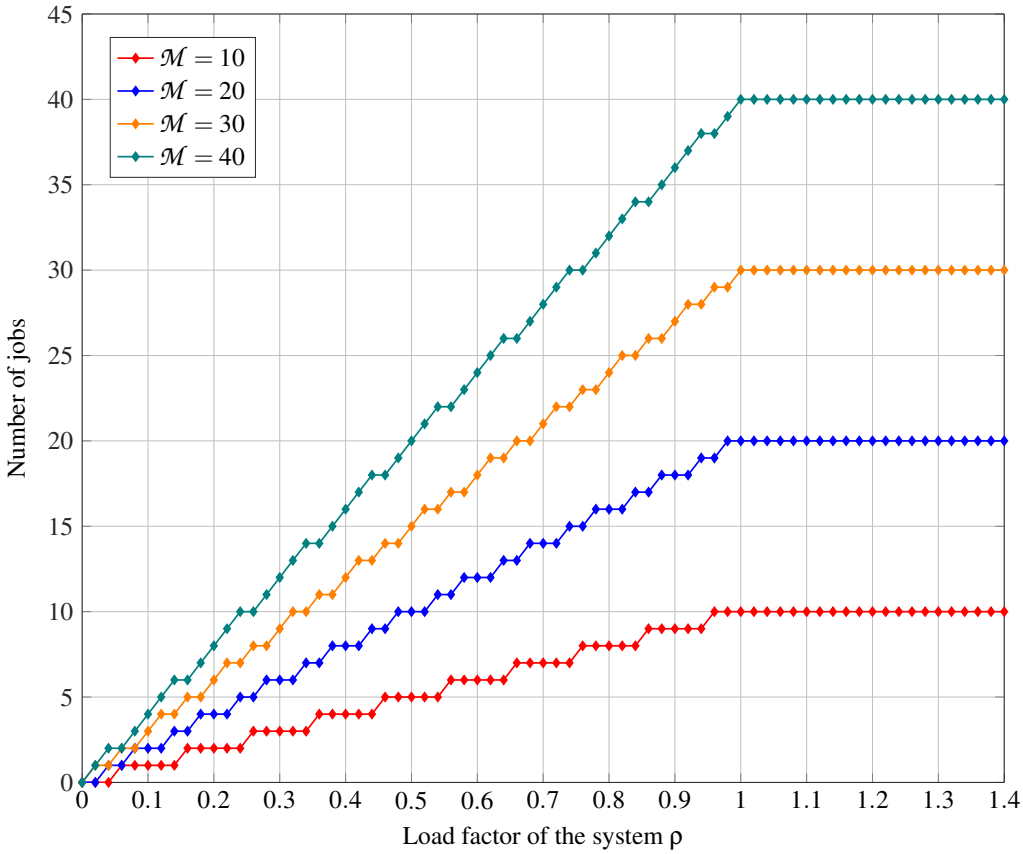


Figure 5.7: Average of operational servers number and load factor.

Although our algorithm calculates from the beginning the best number of servers to be switched on, we observed that this number is kept the same during the whole observation period: $\forall t \;\; m(t) = \lceil \rho \times \mathcal{M} \rceil$. This formula is only valid if the load factor of the system ρ is smaller than 1. Otherwise, if the load factor is greater than 1, the best number of servers to be turned on will exceed the number of servers available, so the system turns on all available servers to be as close as possible to the optimal number of servers. Finally, the number of servers to be switched on by our algorithm in the case of a constant arrival rate is given by:

$$\forall t \;:\; m(t) = \min(\lceil \rho \times \mathcal{M} \rceil, \mathcal{M}). \tag{5.11}$$

Note that the heavier the load factor is, the bigger number of operational servers is.

Note that the results of this subsection are true for costs c_M, c_N and c_L of the same order of magnitude. In the next subsection, we will show the impact of varying the order of magnitude between these costs.

5.3.1.2 Order of magnitude of costs

Let us define c_e as the cost of energy paid in one slot for the treatment of one single job by one server:

$$c_e = \frac{c_M}{d}. \tag{5.12}$$

Table 5.3: Impact of costs on the number of operational servers.

Condition	**# operational servers**	**Behavior**
$c_e \gg c_N, c_L$	0	Lazy
$c_e \ll c_N, c_L$	$\mathcal{M}$	Fully active
$c_e \approx c_N, c_L$	$\min(\lceil \rho \times \mathcal{M} \rceil, \mathcal{M})$	Proportional

The various tests we have done show that the behavior of our strategy against energy and QoS optimization depends on the values c_e, c_N, and c_L. We distinguish mainly three types of behavior:

1. If cost c_e is higher than c_N and c_L, the system prefers to turn off all servers because the cost of switching on a server to serve d jobs is more expensive than the cost of rejecting and/or keeping d jobs in the buffer. We say that the system is *lazy*.
2. If cost c_e is smaller than c_N and c_L, then the system prefers to turn on all the servers because the total cost of waiting and rejection is much more important than the cost of the switching on more servers. We say that the system is *fully active*.
3. If cost c_e, c_N and c_L are close to each other, the system immediately turns on a number of servers proportional to the load factor: $\min(\lceil \rho \times \mathcal{M} \rceil, \mathcal{M})$.

To better clarify the results reported in Table 5.3, a closer analysis of the relationship between costs, the load factor and the optimal number of servers is discussed in the following.

Theorem 5.6. *Assume that the buffer size is infinite or sufficiently large. For any slot t the number of servers computed by the greedy algorithm 9 is:*

$$m(t) = \begin{cases} \mathcal{M} & \text{if } c_e < c_N \\ 0 & \text{if } c_e > c_N. \end{cases}$$

Proof. Assume that b is $+\infty$. This implies that all jobs will be accepted and no job will be rejected, which means that the number of loss jobs $l(t)$ is always zero:

$$\forall t \ : \ \lim_{b \to \infty} l(t) = 0.$$

Additionally, as the arrival process is modeled in this section by a constant, we have: $\forall t \ : \ \mathcal{H}_{A(t)} = \Delta_a$, then we deduce that the number of waiting jobs will be also deterministic:

$$\begin{aligned} n(t) &= \min(b, (n(t-1) + a(t) - d \times m(t))^+) \\ &= \min(+\infty, (n(t-1) + a(t) - d \times m(t))^+) \\ &= (n(t-1) + a(t) - d \times m(t))^+ \\ &= \max(0, n(t-1) + a(t) - d \times m(t)) \\ &= \max(0, n(t-1) + a - d \times m(t)) \end{aligned}$$

and the total cost $c(t)$ will be:

$$\begin{aligned} c(t) &= c_M \times m(t) + \sum_{i \in S_{N(t)}} P_{N(t)}(i) \times i \times c_N + \sum_{i \in S_{L(t)}} P_{L(t)}(i) \times i \times c_L \\ &= c_M \times m(t) + n(t) \times c_N + l(t) \times c_L \\ &= c_M \times m(t) + (\max(0, n(t-1) + a(t) - d \times m(t))) \times c_N + 0 \times c_L \\ &= c_M \times m(t) + c_N \times \max\{0, n(t-1) + a - d \times m(t)\}. \end{aligned}$$

We have two cases:

1. if $(n(t-1) + a - d \times m(t)) \leq 0$ then $c(t) = c_M \times m(t)$. In this case we must always choose $m(t) = 0$ to ensure a minimum total cost.

2. if $(n(t-1) + a - d \times m(t)) > 0$ then

   $$\begin{aligned} c(t) &= c_M \times m(t) + c_N \times (n(t-1) + a - d \times m(t)) \\ &= m(t) \times (c_M - d \times c_N) + c_N \times (n(t-1) + a). \end{aligned}$$

   It is clear that the right-hand side term $c_N \times (n(t-1) + a)$ is always a positive value. Thus minimizing the total cost requires the minimization of the left-hand side term $m(t) \times (c_M - d \times c_N)$. Thus, we are mainly dealing with two sub-cases:

   (a) If $c_e < c_N$ then $m(t) \times (c_M - d \times c_N)$ is negative, and choosing a maximum value of $m(t) = \mathcal{M}$ ensures a minimum total cost.

   (b) If $c_e > c_N$ then $m(t) \times (c_M - d \times c_N)$ is positive, and choosing a zero value of $m(t) = 0$ provides a minimum total cost. □

5.3.2 Experiments for i.i.d. arrival distribution

In this section we will study the case where arrivals are modeled by a single distribution that does not change over time: $\forall t \ : \ \mathcal{H}_{A(t)} = \mathcal{H}_A$, it is the case of i.i.d. job arrivals.

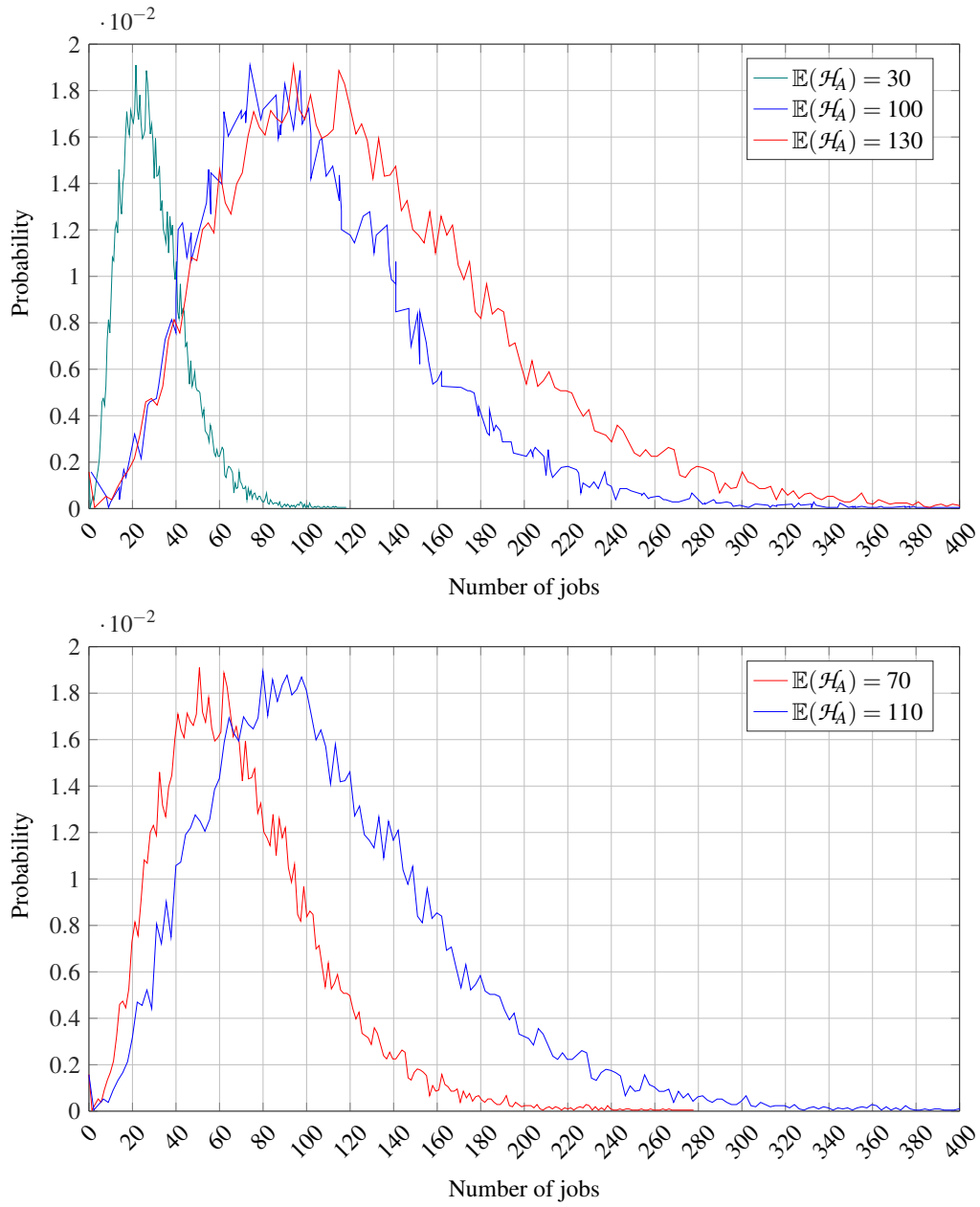


Figure 5.8: Example of different arrival distributions inducing different load factors ρ.

We will consider arrival distributions which induced different load factors as presented in Figure 5.8 for this part of experimentation which is computed to be i.i.d. based on the turning point approach presented in the end of Section 1.3.3, when using the real open trace of Google [Wil11, RWH11].

5.3.2.1 Load factor variation

In this section we will use the three types of arrivals we have previously introduced to analyze numerically the system whose parameters are described in Table 5.4.

Table 5.4: Settings of the first numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	150	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	1000	jobs	buffer size
h	1000	slot	study period
c_M	10		cost of energy needed by a server
c_N	10		cost of waiting a job
c_L	10		cost of rejecting a job

Figure 5.9 shows the average number of operational servers over the period of analysis h for several values of ρ. The bigger ρ is, the more important the average number of operational servers is.

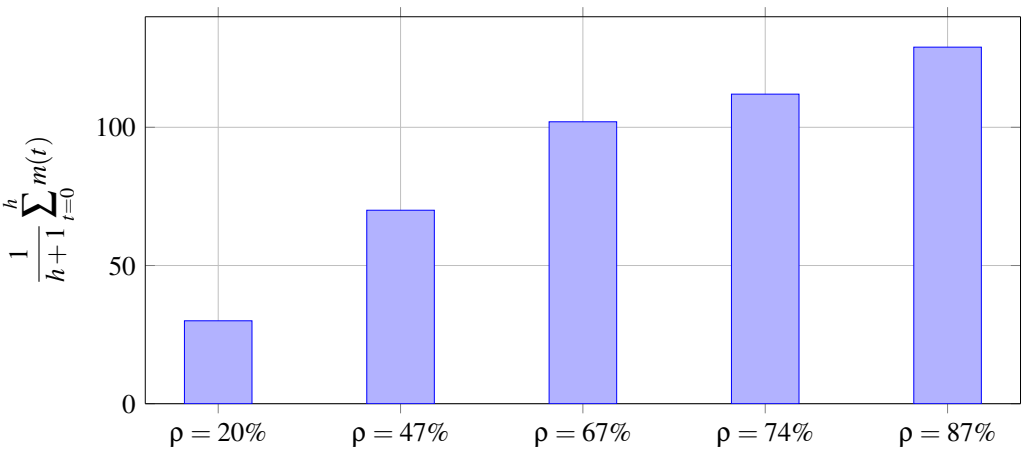


Figure 5.9: The average number of operational servers over the period of analysis depending on load factor values ρ.

Figure 5.9 shows the evolution, of the average number of operational servers according to several load factor values of ρ. Clearly, we observe that the heavier load factor is, the bigger needed operational servers is. If the system is overloaded the system eventually turn on all available servers. Thus, in general, the ultimate number of operational servers in the long term converges to: $\min\left\{\mathcal{M}, \lceil \rho \times \mathcal{M} \rceil\right\}$.

Figure 5.10 shows the evolution of the real number of operational servers over the period of analysis h for several values of ρ. This figure shows the transient aspect of the switching off/on of the servers.

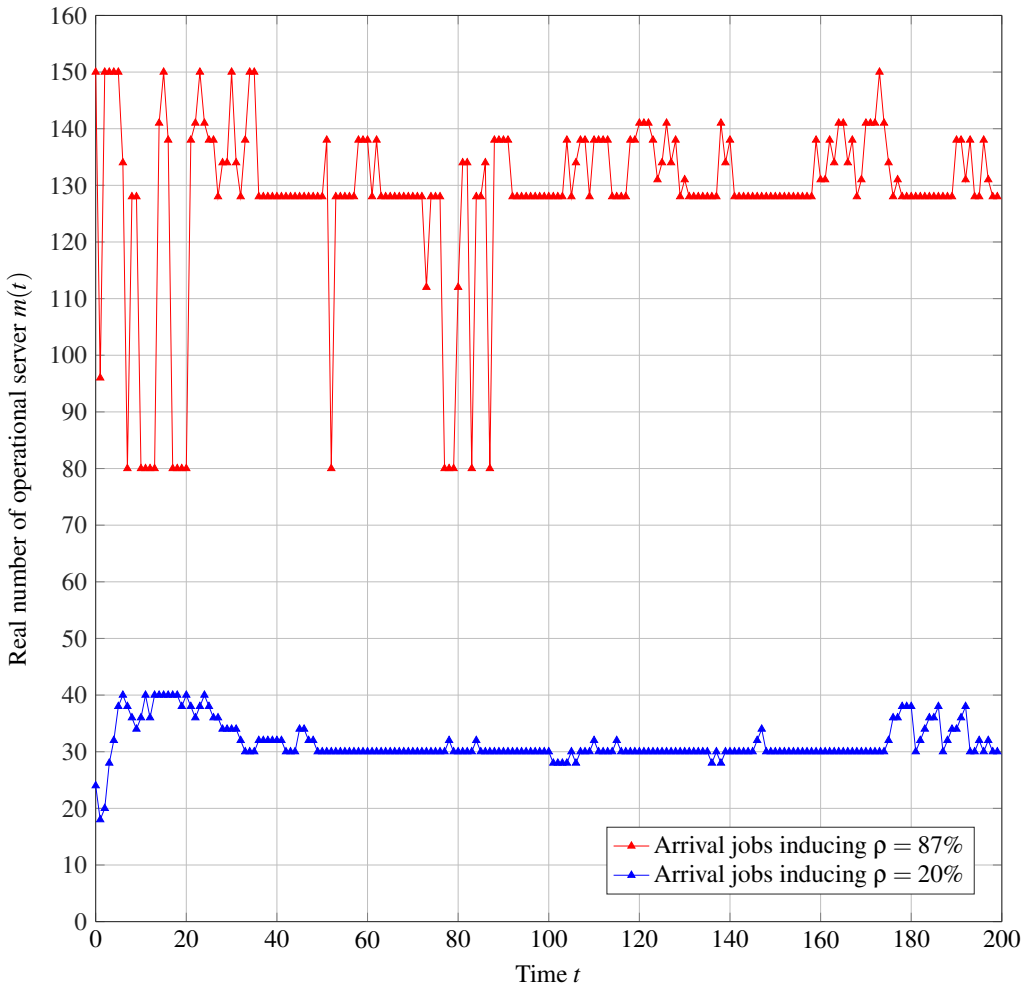


Figure 5.10: Evolution of the number operational server depending on the load factor.

5.3.2.2 Cost variation

In this subsection we will consider an analysis with the same job arrivals distribution $\mathcal{H}_A$, where varying the costs c_M, c_N and c_L. We analyze numerically the system whose parameters are described in Table 5.5.

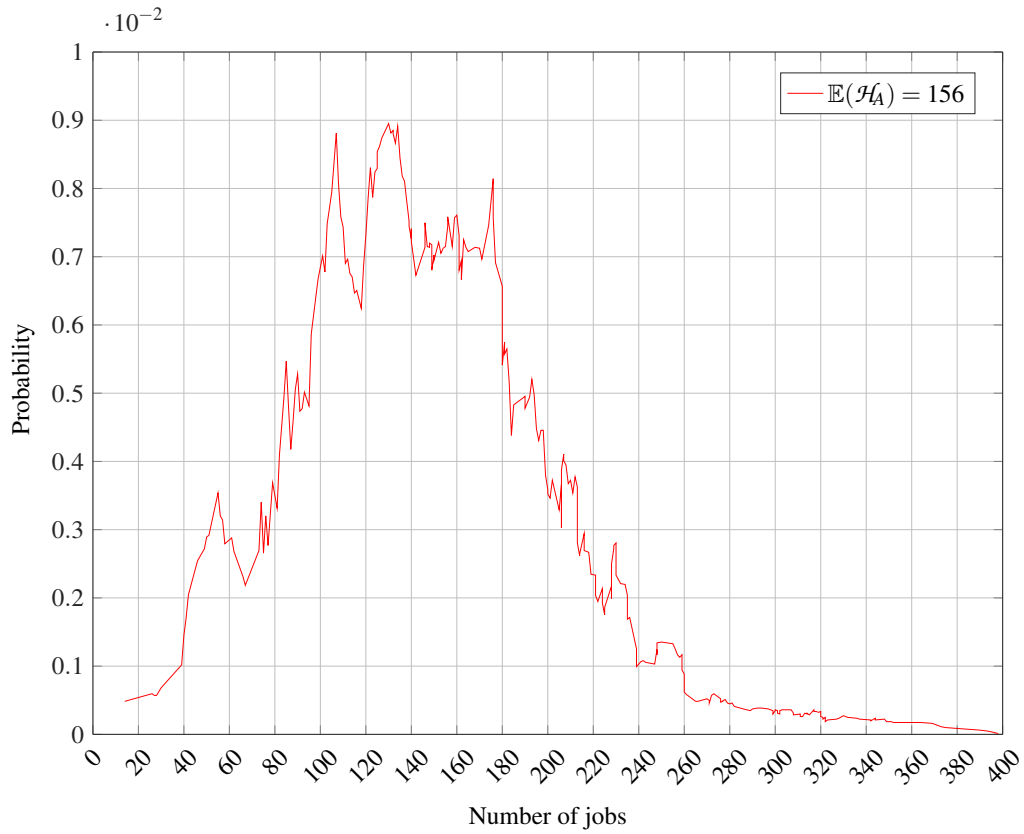


Figure 5.11: Example of arrival distributions.

Table 5.5: Settings of the second numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	300	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	1000-9000	jobs	buffer size
h	1000	slot	study period
ρ	52%	-	load factor (see Figure 5.11)

Waiting cost In this subsection we have set a small value for the cost of energy consumption c_e and a high value for the cost of waiting $c_N > c_e$ with any value for the loss cost. Figure 5.12 illustrates the result of this configuration.

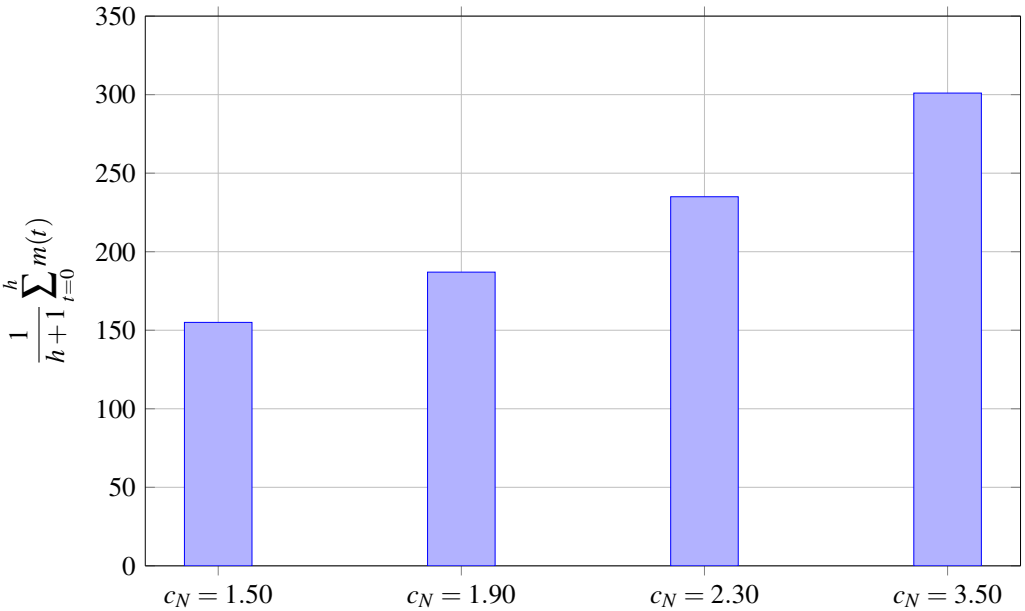


Figure 5.12: Evolution of the average number of operational servers over the period of analysis h for different values of waiting cost c_N ($b = 1000$, $c_L = c_M = 0.5$).

We observed that the system switches on a significant number of servers if the cost of energy is lower than the one of waiting. Thus the system prefers to turn more servers on to serve more jobs and avoid long waiting time.

Loss cost In this numerical analysis we have set a small value for c_e (the cost of energy paid in one slot for the treatment of one single job by one server) and even smaller value for the cost of waiting c_N but a high value for the cost of rejection c_L: $c_L > c_e > c_N$.

When fixing b the size of the buffer and varying c_L (while keeping $c_L > c_e > c_N$), we observe that the higher the cost c_L is, the higher the system turns on servers. It is a completely natural reaction because the system tries to minimize the total cost, and as the loss cost c_L is highest, the system will turn more servers on to serve more jobs and emptying further the queue which allows a low loss rate. Figure 5.13 illustrates this phenomenon.

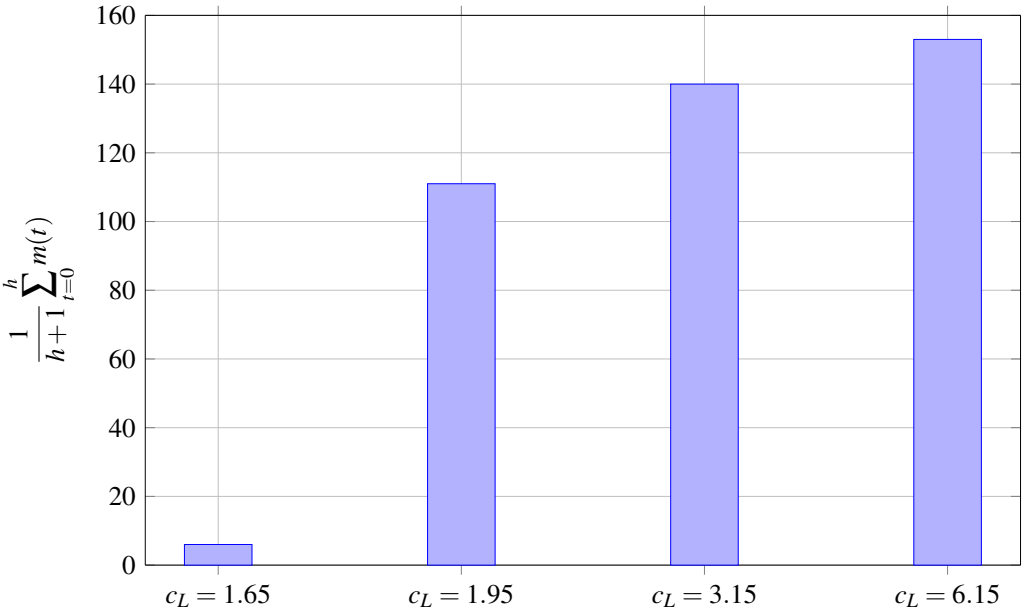


Figure 5.13: Evolution of the average number of operational servers for different values of loss cost c_L ($b = 1000$, $c_N = c_M = 0.5$).

Figures 5.14, 5.15, 5.16, and 5.17 show the evolution of: $\frac{m(t)}{\mathcal{M}}$ the operational servers rate, and $\frac{n(t)}{b}$, the buffer fill rate, over the period of analysis h for several values of b. These figures show the transient aspect of the switching off/on of the servers and the number of waiting jobs. This Figures illustrate the system-inertia phenomenon. We observe that at the beginning and during a certain period, the system keeps all the servers switched off and holds jobs on waiting, because the cost of turning on a server is higher than keeping a job waiting. Thus the system prefers to put jobs on buffer. That said, after a certain inertia period the buffer b becomes full and the system begins to reject jobs, and as the loss cost is too high compared to other costs, the system starts to turn on servers in order to reduce the loss rate.

Let us define the *inertia period* as the initial period during which all the servers of system are turned off.

Curves of Figures 5.14, 5.15, 5.16, and 5.17 show that the larger the buffer b is, the longer inertia period is. The inertia period before the response of the system can be approximated by:

$$\frac{b}{\mathbb{E}(\mathcal{H}_A)}. \tag{5.13}$$

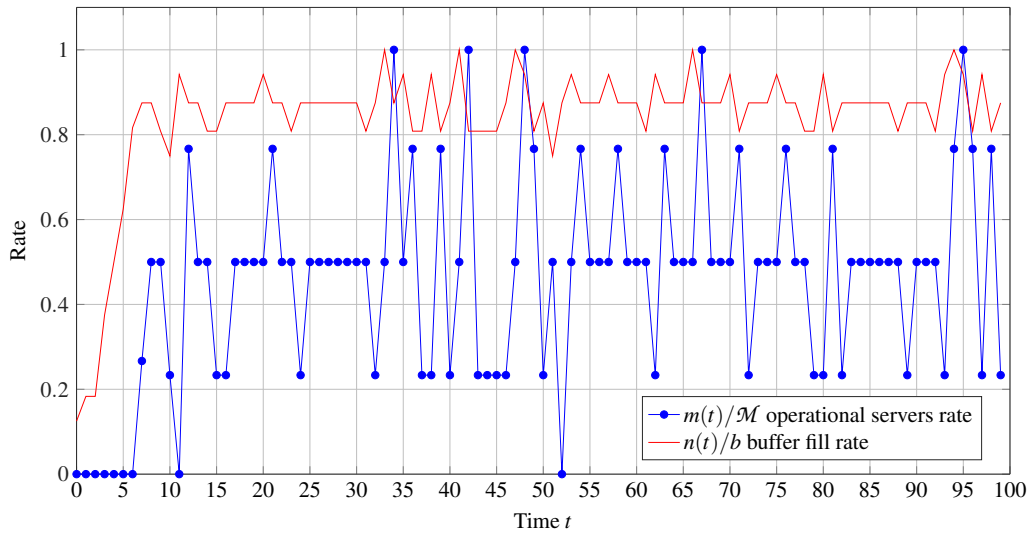


Figure 5.14: Inertia phenomenon: Evolution, over time, of operational servers rate and buffer fill rate for $b = 1000$ ($c_N = c_M = 1$ and $c_L = 5$).

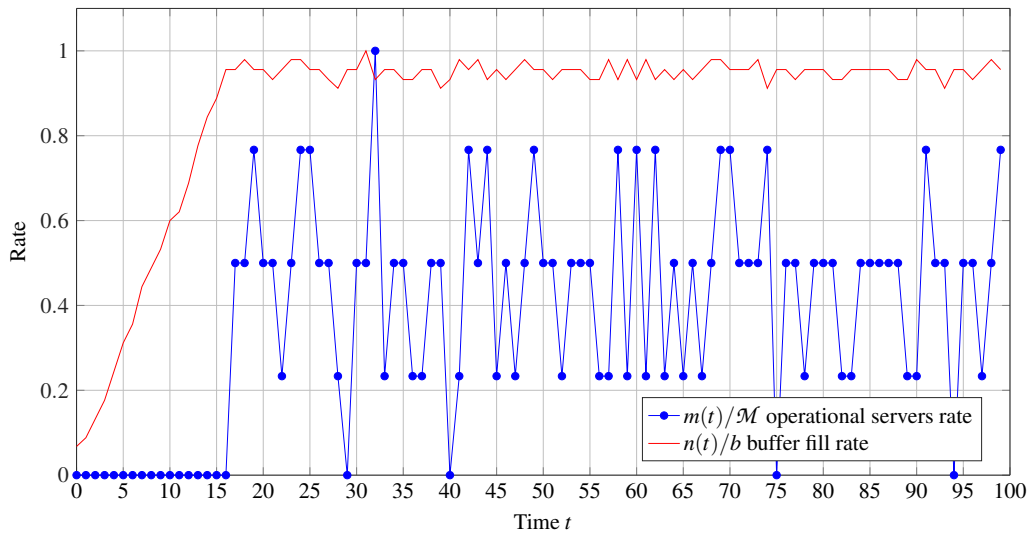


Figure 5.15: Inertia phenomenon: Evolution, over time, of operational servers rate and buffer fill rate for $b = 3000$ ($c_N = c_M = 1$ and $c_L = 5$).

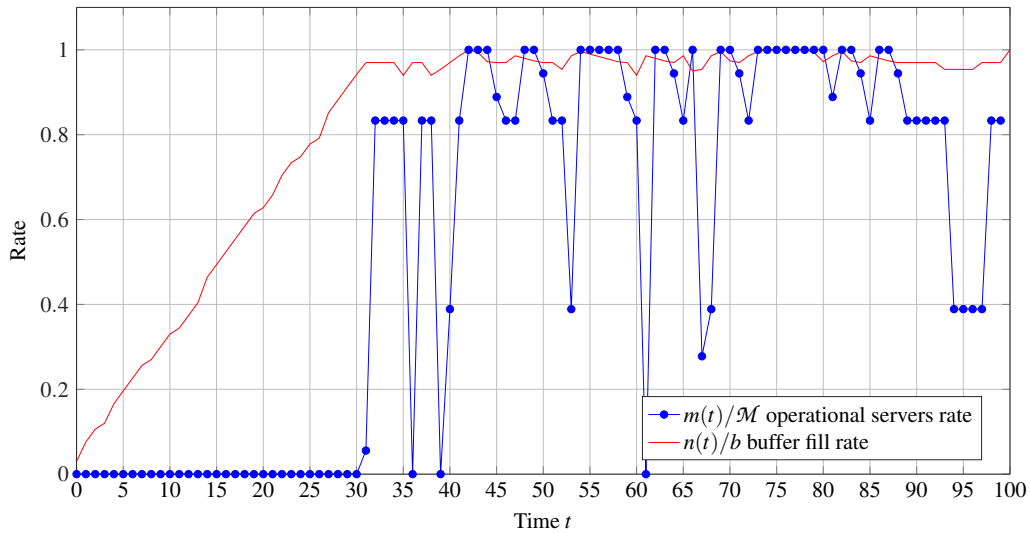


Figure 5.16: Inertia phenomenon: Evolution, over time, of operational servers rate and buffer fill rate for $b = 5000$ ($c_N = c_M = 1$ and $c_L = 5$).

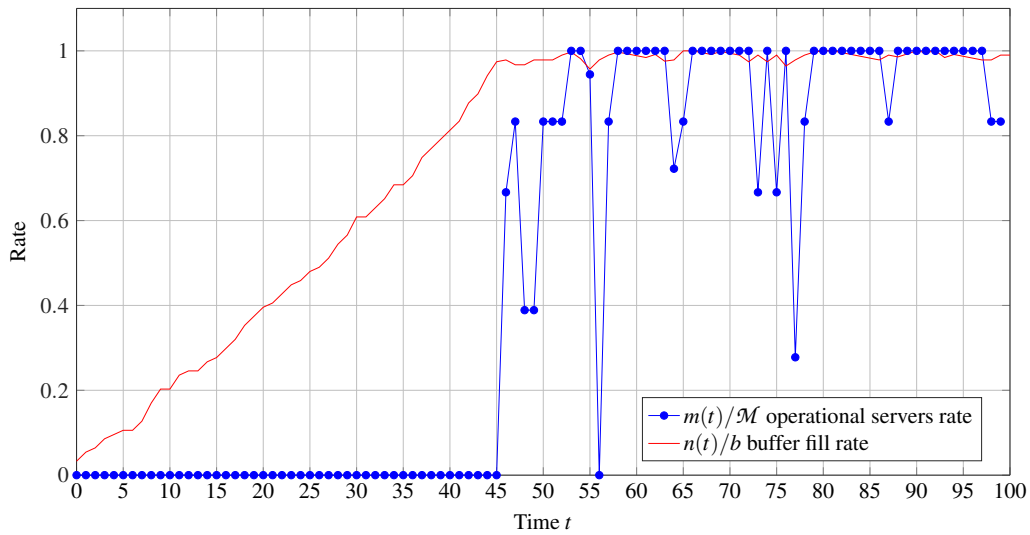


Figure 5.17: Inertia phenomenon: Evolution, over time, of operational servers rate and buffer fill rate for $b = 7000$ ($c_N = c_M = 1$ and $c_L = 5$).

5.3.3 Experiments for variable arrival distribution

In this section we will generalize our study extending it for arrivals modeled by a distribution that changes over time: $\exists t_1 \; \exists t_2 \; : \; \mathcal{H}_{A(t_1)} \neq \mathcal{H}_{A(t_2)}$.

5.3.3.1 Hourly arrival variation

In a real data center the arrivals jobs vary over the day. For example high rate arrivals between 8 a.m. and 4 p.m., low arrivals between 4 p.m. and midnight, and medium arrivals between midnight and 8 a.m. (see Figure 5.18).

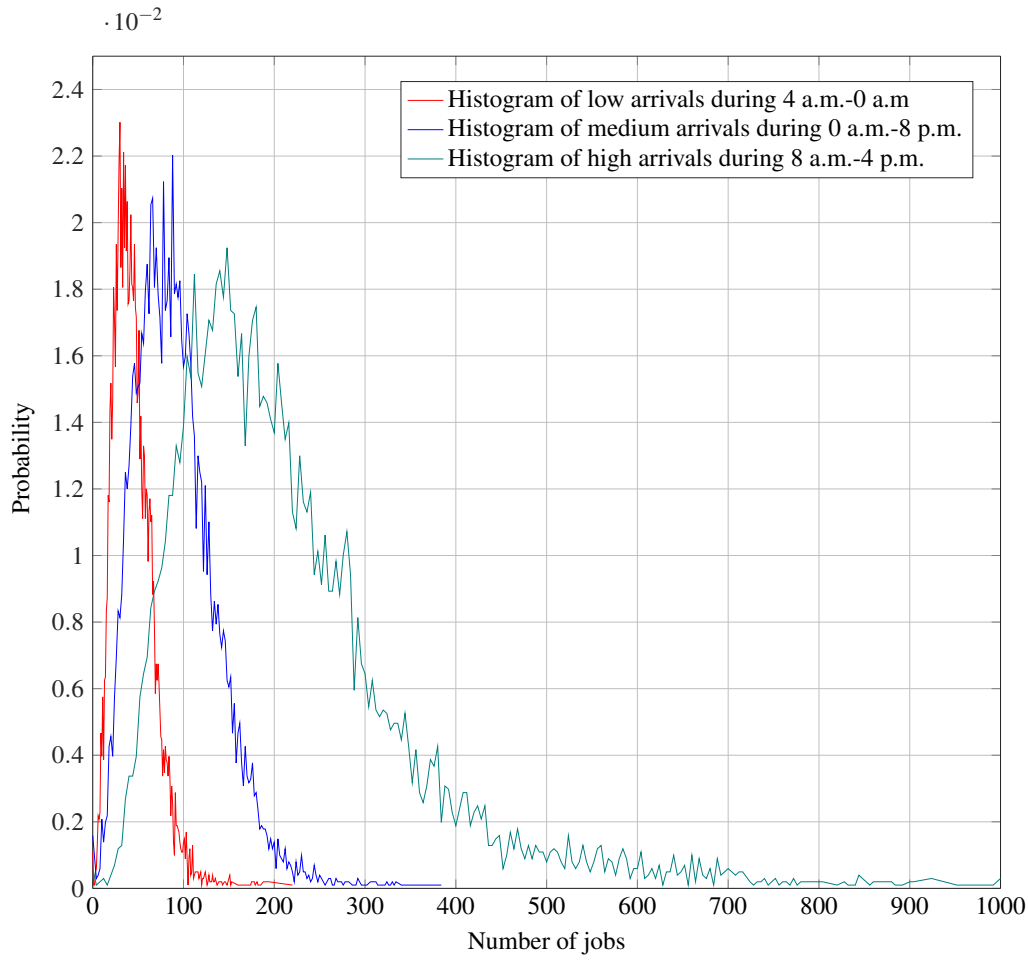


Figure 5.18: Discrete distributions.

Figure 5.19 shows the results of analyzing numerically the system whose parameters are described in Table 5.6.

Table 5.6: Settings of the third numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	400	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	3000	jobs	buffer size
c_M	7		cost of energy needed by a server
c_N	9		cost of waiting a job
c_L	8		cost of rejecting a job

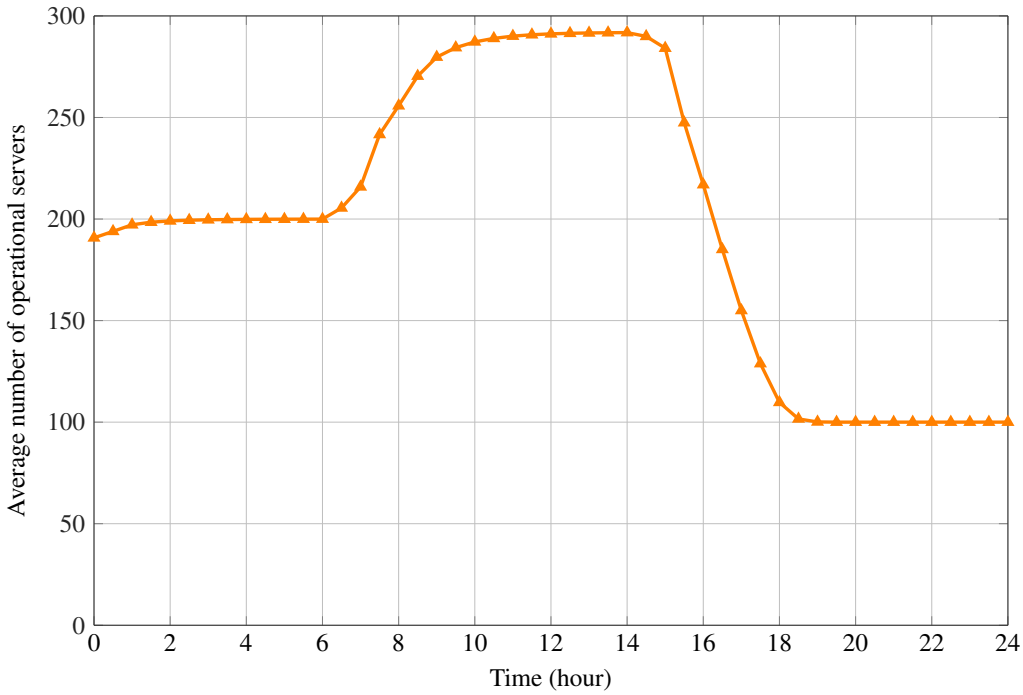


Figure 5.19: Evolution of the average number of operational servers during one day (each triangle represents the average number of operational servers during half an hour).

We observe that the system turns on a number of servers at the beginning of the day to treat arriving jobs. Then, it gradually increases the number of operational servers to treat the high arrival rate between 8 a.m. and 4 p.m. Then from 4 p.m. it begins to turn off the servers and keeping only reduced number of operational servers to serve the low arrival rate until midnight. We clearly note the dynamic adaptation of the energy optimization system to the traffic variation.

5.3.3.2 Daily arrival variation

This section uses real traffic traces to model arrivals. We use the open *clusterdata-2011-2* trace [Wil11, RWH11]. This traffic trace is sampled with a sampling period equal to the slot duration. We consider frames of two minutes to sample the trace and construct seven empirical distributions corresponding to arrivals during each day of the week. For example to compute the distribution of *sunday*, as the trace contains the timestamp of each arrival jobs during a whole month, we were able to group the arrivals of the four Sundays of the month, after that we sampled those arrivals to compute the histogram.

Table 5.7: Example of daily variation of arrivals obtained from Google trace where $\mathbb{E}(\mathcal{H}_A)$ represents the average number of new jobs and $\sigma(\mathcal{H}_A)$ its standard deviation.

Day of week	**Arrivals rate**	$\mathbb{E}(\mathcal{H}_A)$	$\sigma(\mathcal{H}_A)$
Monday	Low	43 jobs	21
Tuesday	Medium	51 jobs	25
Wednesday	Medium	49 jobs	23
Thursday	High	58 jobs	38
Friday	Medium	53 jobs	25
Saturday	Low	41 jobs	24
Sunday	Low	39 jobs	22

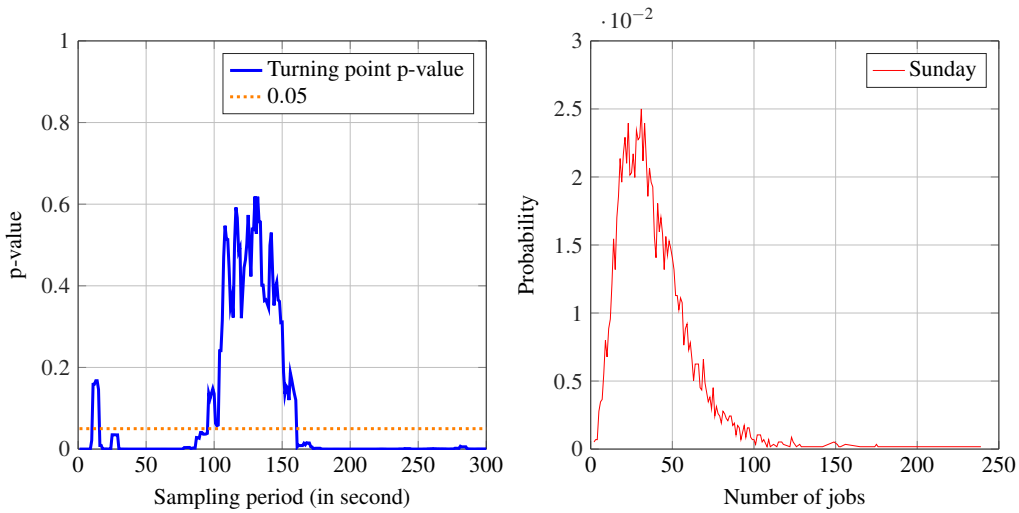


Figure 5.20: Google trace arrivals and i.i.d.-ness test for the Sunday days.

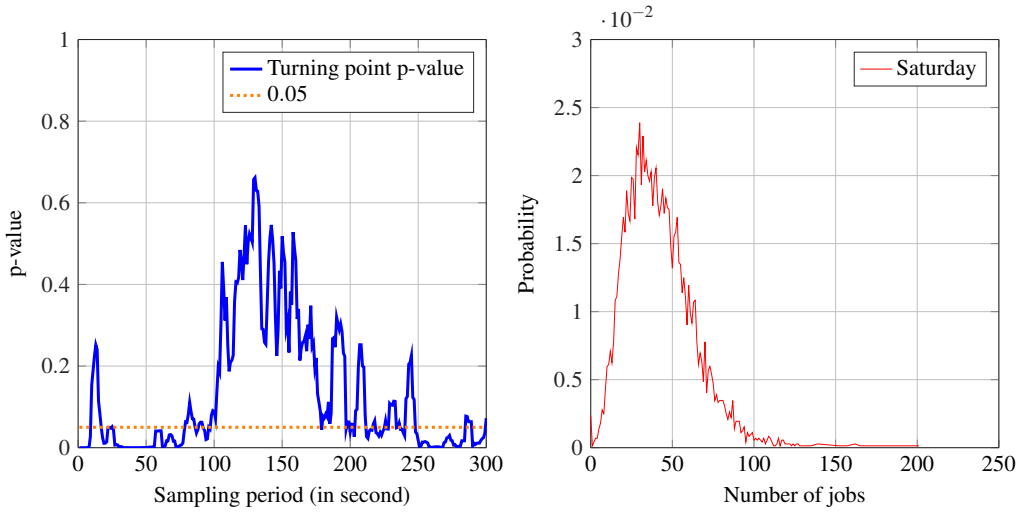


Figure 5.21: Google trace arrivals and i.i.d.-ness test for the Saturday days.

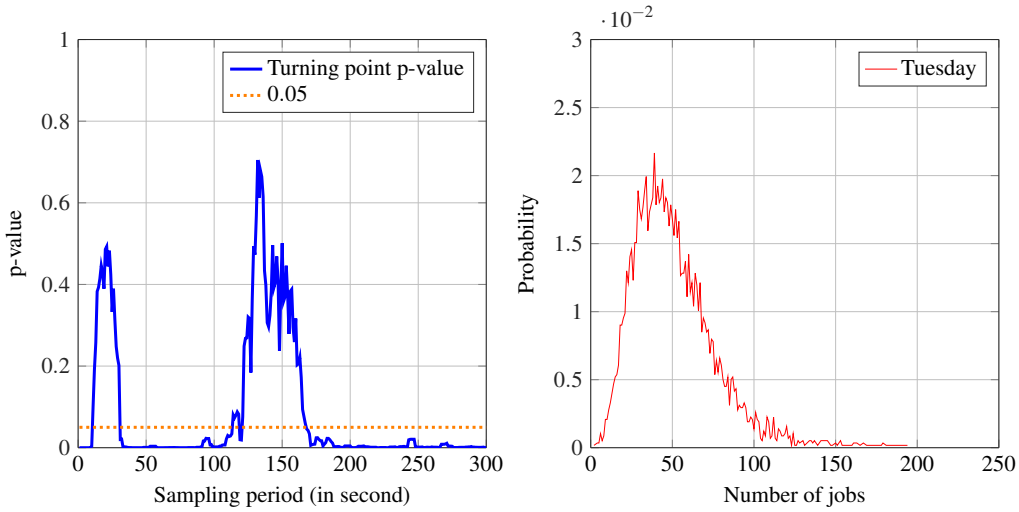


Figure 5.22: Google trace arrivals and i.i.d.-ness test for the Tuesday days.

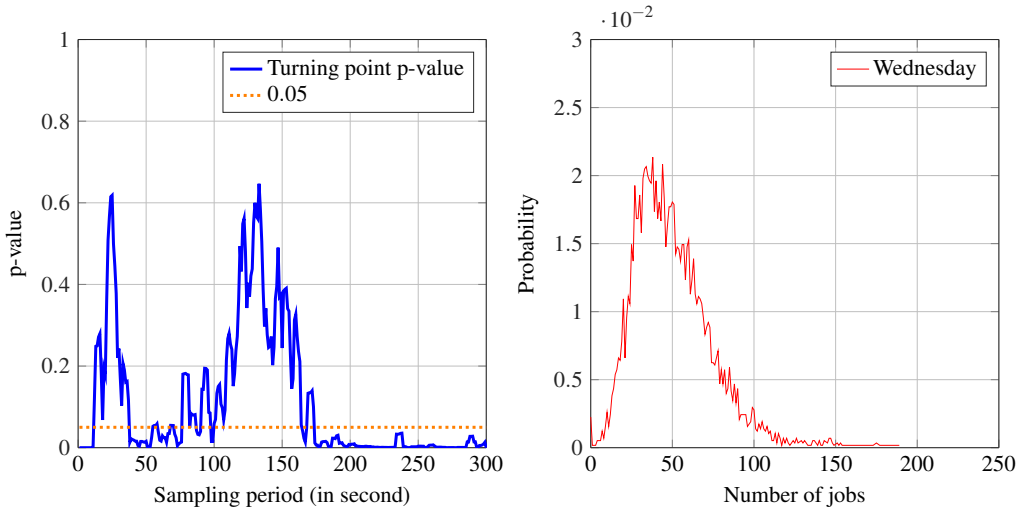


Figure 5.23: Google trace arrivals and i.i.d.-ness test for the Wednesday days.

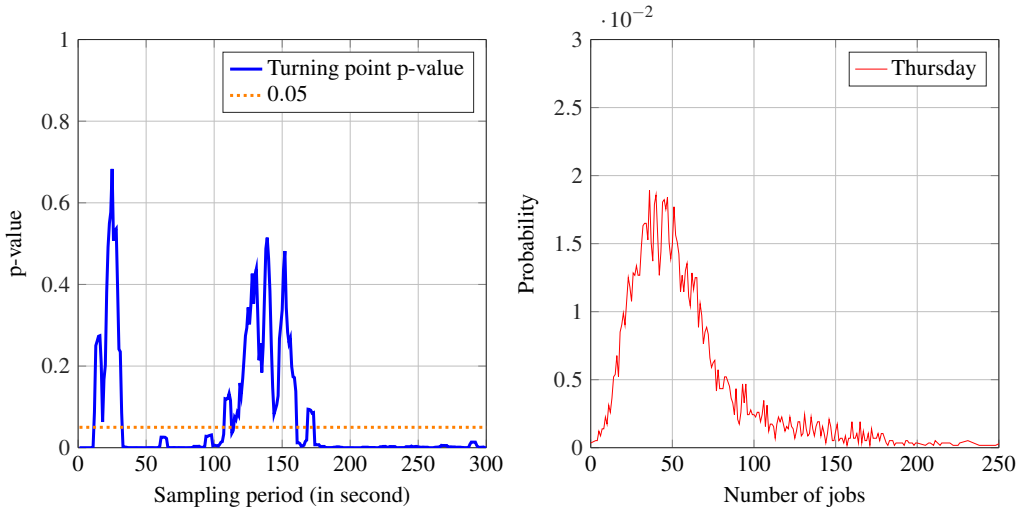


Figure 5.24: Google trace arrivals and i.i.d.-ness test for the Thursday days.

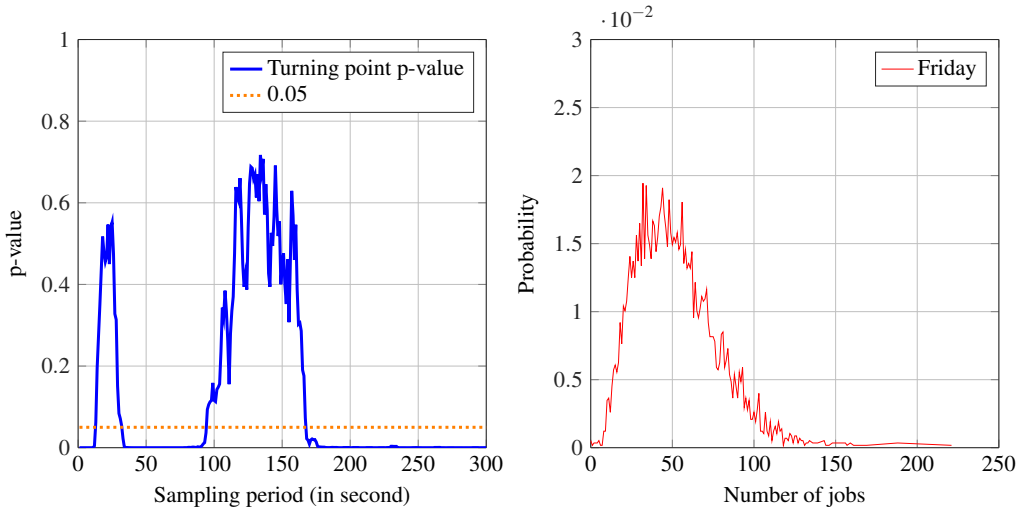


Figure 5.25: Google trace arrivals and i.i.d.-ness test for Friday days.

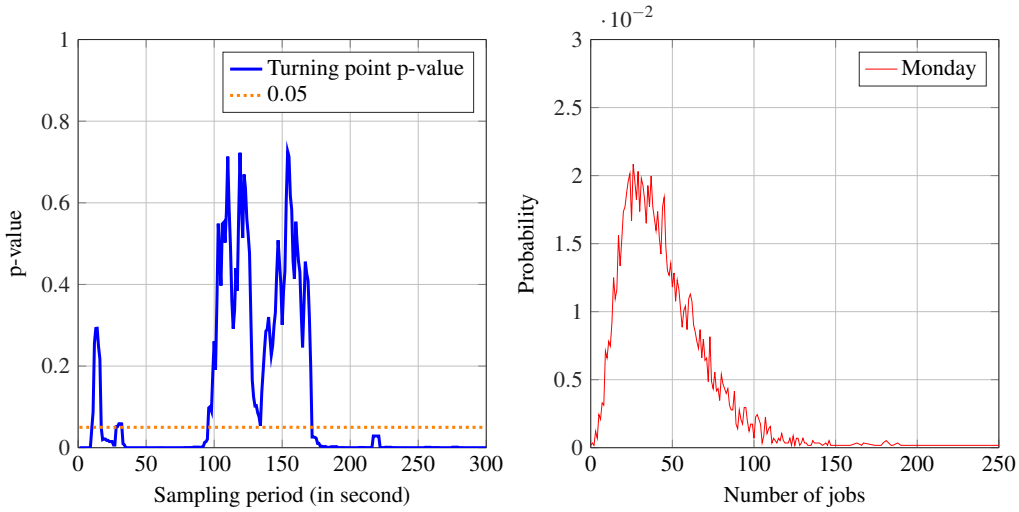


Figure 5.26: Google trace arrivals and i.i.d.-ness test for the Monday days.

The turning point test was applied for each day at confidence level 95% (means 5% of error level). Let us define I_k as the interval for which the sampled data of day k is i.i.d. In fact we choose a sampling period which is in the intersection of all intervals I_k which includes the value of 136 second:

$$sampling_period \in \bigcap_{k=1}^{7} I_k. \tag{5.14}$$

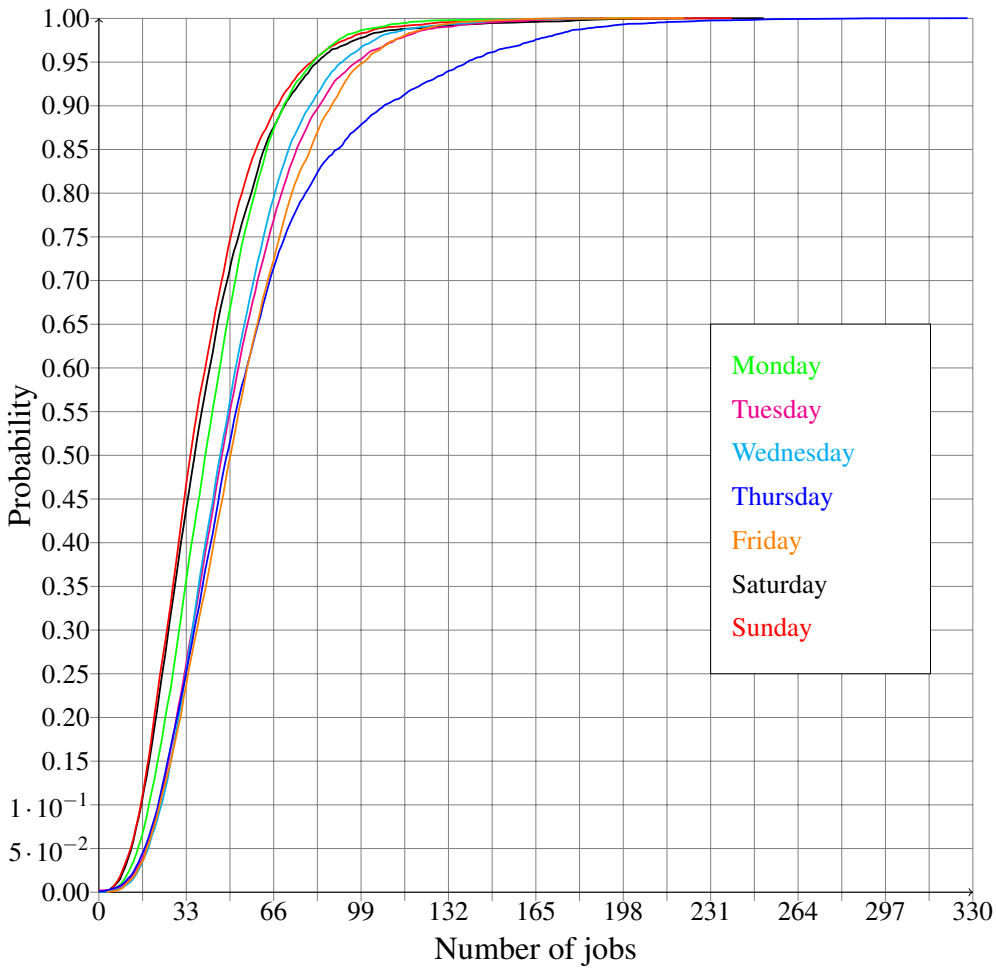


Figure 5.27: Cumulative distribution of $\mathcal{H}_{A(t)}$ for days of the week.

A high arrivals rate is observed on Thursday, low arrivals rate on Saturday, Sunday and Monday, and medium arrivals rate during the rest of the week (see Table 5.7). These distributions have different statistical properties reflecting the fluctuation of traffic over the week (see Figure 5.27). For instance, we observe an average of 39 (resp. 58) jobs per minute during Sunday (resp. Thursday) with a standard deviation of 22 (resp. 38). Figure 5.28 shows the results of analyzing numerically the system whose parameters are described in Table 5.8. We note that the work presented in this section was published in [Bay16].

Table 5.8: Settings of the last numerical analysis.

Parameters	**Value**	**Unit**	**Description**
max	100	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	300	jobs	buffer size
c_M	7		cost of energy needed by a server
c_N	23		cost of a waiting job
c_L	29		cost of rejecting a job

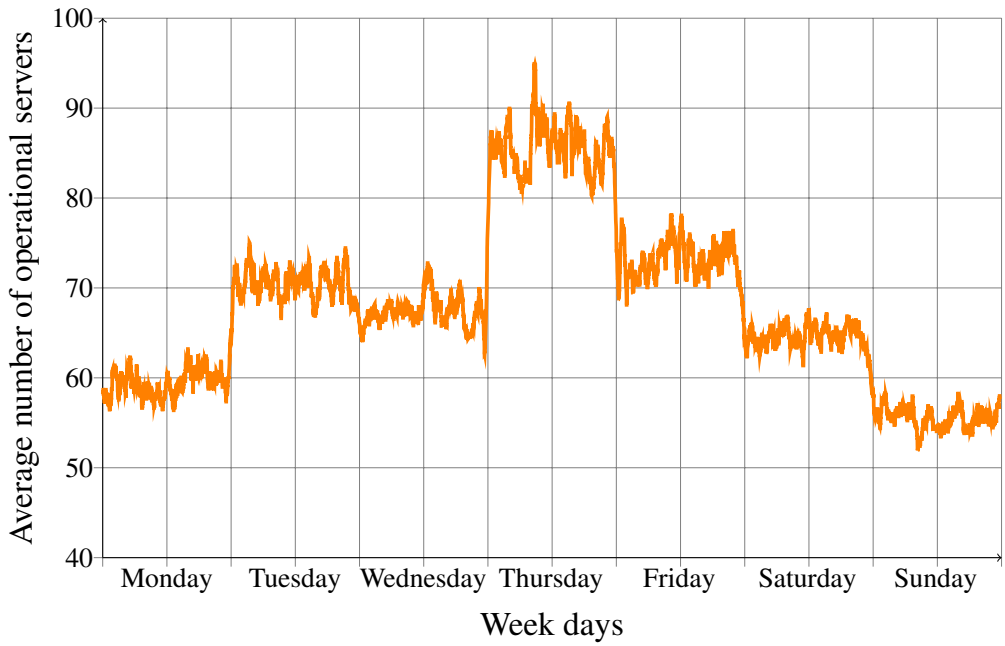


Figure 5.28: Evolution, over a week, of the average number of operational servers (each point of the curve is the average number of operational servers during half an hour). Our algorithm adapts the number of operational server according to the traffic variation.

The methodology adopted in this chapter was built in a tool based on the greedy-window Algorithm 10. The tool takes as input a sequence of instants $0, 1, \ldots, h$ and the corresponding arrival traffic distribution $\mathcal{H}_{A_0}, \mathcal{H}_{A_1}, \ldots, \mathcal{H}_{A_h}$. $\mathcal{H}_{A_k}, k \in 0..h$ refers to the traffic during slot t. The data center is described by its parameters: b, c_M, c_N, c_L, c_{on}, $\mathcal{M}$.

The tool's main function is called for each slot, the function computes the QoS and the energy accumulated cost over a period of w slot while considering all possible actions. Then choose the action that minimize this cost. Algorithm 10 shows the core control function of the tool in a simplified way. Note that the tool is written in Python in a schema that is easily parallelized.

Chapter 6

Comparison of methods and Conclusion

THIS thesis focuses the problem of saving energy consumption in data centers while guaranteeing a high quality of the services. We use the concept of histograms and the set of its associated operations in order to better model the job arrivals and the service rates with general discrete distributions obtained from real traces, empirical data, or incoming traffic measurements. This way of modeling arrivals avoids classical assumptions like Poisson arrivals assumption or exponential services time assumption.

Then we presented a queuing model used to formalize the problem of managing energy and performance in data centers.

After that we introduced the use of DTMC to model and solve a thresholds optimization strategy. We presented also the use of histogram operators to analyze numerically the trade-off between energy consumption and performance evaluation to determine the best threshold based policy among a set of parameters proposed by the modeler.

Next we used the concept of *Markov Decision Process* to find the optimal strategy and then analyze the performance and the energy consumption of the data center under optimal strategy and using PRISM to specify the problem and to analyze the experiments. Because of the computational limitation of the MDP method, we proposed and explained our greedy-window approach to optimize energy and QoS in the data center. Our greedy-window approach is based on observing every slot the state of the system (the number of job arrivals, the number of waiting jobs, the number of rejected jobs and the number of operational servers) then turning on or off an optimal number of servers to minimize energy consumption and maximize the QoS (see Algorithm 11).

6.1 Final result

In this section we compare the performance and the energetic gain induced by the several methods proposed while using real traffic traces to model arrivals.

This traffic trace is sampled with a sampling period equal to the slot duration. We consider frames of 136 second to sample the trace and construct one empirical distribution corresponding to arrivals during all the month (see the end of Section 1.3.3 to understand how the value 136 was chosen).

We do not choose to consider the rejected jobs, neither the latency nor the heterogeneity of the system, because as we observed in Chapter 4 those aspects lead to the explosion of the space state of the MDP based method. Otherwise, we set the same conditions, parameters, and assumptions for the different methods. We found that the use of MDP gives an optimal strategy but need a huge amount of memory and time to find the best strategy. Additionally, when the parameters of the model are big (for example when the size of buffer b or the maximum number of servers $\mathcal{M}$ exceeds 200), the current implementations of the value iteration algorithm (as in PRISM) may not be able to complete the computation to find the best policy. However our greedy-window algorithm gives a strategy close to the optimal strategy (2% of difference) and need a little amount of memory and time to find the best strategy. This greedy-window algorithm is fast compared to the other methods (see the experimental results of Sections 6.1.1, 6.1.2, 6.1.3, 6.1.4, and 6.1.5 in which we show the difference between the cost of the best strategy given by our greedy-window algorithm, MDP, truncated MDP, greedy, and threshold method, we show also the difference between the time of computation of each strategy and the memory usage).

Because of the explosion of the number of states PRISM gives the message ***out of memory***. Our algorithm does not have this problem (see experimentation of Section 6.1.6). The following Tables 6.1 and 6.2 summarizes the theoretical time-space complexity of MDP and our greedy-window algorithm.

Table 6.1: Time complexity comparison: Notice that the complexities are for the homogeneous model without considering latency nor rejected jobs.

	Time	**Theorem**
Thresholds U/D	$O(h \times b^2 \times (b + \max(S_A)) \times \log(b + \max(S_A)))$	3.2
MDP	$O(h \times \mathcal{M}^2 \times b \times \lvert S_A \rvert)$	4.1
Greedy-window	$O(w \times h \times \mathcal{M} \times (b + \max(S_A)) \times \log(b + \max(S_A)))$	5.5

Table 6.2: Space complexity comparison: Notice that the complexities are for the homogeneous model without considering latency nor rejected jobs.

	Space	**Theorem**
Thresholds U/D	$O(b^2)$	3.3
MDP	$O(h \times \mathcal{M} \times b)$	4.4
Greedy-window	$O((b + \max(S_A)) \times \log(b + \max(S_A)))$	5.4

Note that the number of states in MDP is bounded above by $b \times \mathcal{M}$, and the number of transitions is bounded above by $\mathcal{M}^2 \times b \times |S_A|$, and the space complexity of MDP may take into account the two bounds: $O(\mathcal{M} \times b)$ gives the size of the space of states and $O(\mathcal{M}^2 \times b \times |S_A|)$ represents the size of the sparse matrix of MDP which stores the transitions. Additionally, the value iteration algorithm need to use a space of $O(h \times \mathcal{M} \times b)$ to solve the MDP model and trace-back the actions sequence of optimal strategy (see Theorem 4.4).

As noticed in Remark 1.4 both the support of $\mathcal{H}_A$ and its extent are of the same order, thus in terms of complexity $|S_A|$ and $\max(S_A)$ have the same order. If additionally we consider that $|S_A|$, $\max(S_A)$, and b are of the same order of magnitude, lets say q we will have:

Table 6.3: Time complexity comparison.

	Time	**Space**	**Theorem**
Thresholds U/D	$O(h \times b^3 \times \log(b))$	$O(b^2)$	3.2, 3.3
MDP	$O(h \times \mathcal{M}^2 \times b^2)$	$O(h \times \mathcal{M} \times b)$	4.1, 4.4
Greedy-window	$O(h \times \mathcal{M} \times b \times \log(b) \times w)$	$O(b \times \log(b))$	5.4, 5.5

In comparison with MDP, our greedy-window algorithm gives a lower time complexity as it is linear in $\mathcal{M}$ and quasi linear in b, however MDP is quadratic in $\mathcal{M}$ and b. Additionally, as seen at the end of Section 5.2.3, w is negligible compared to the other factors as b, h, and $\mathcal{M}$.

In the following we give all details of six different experimentation we have done to compare the methods presented in this work. Notice that, in the first, the second, the third, and the fourth experimentation we use the same arrival jobs distribution presented in Figure 6.1.

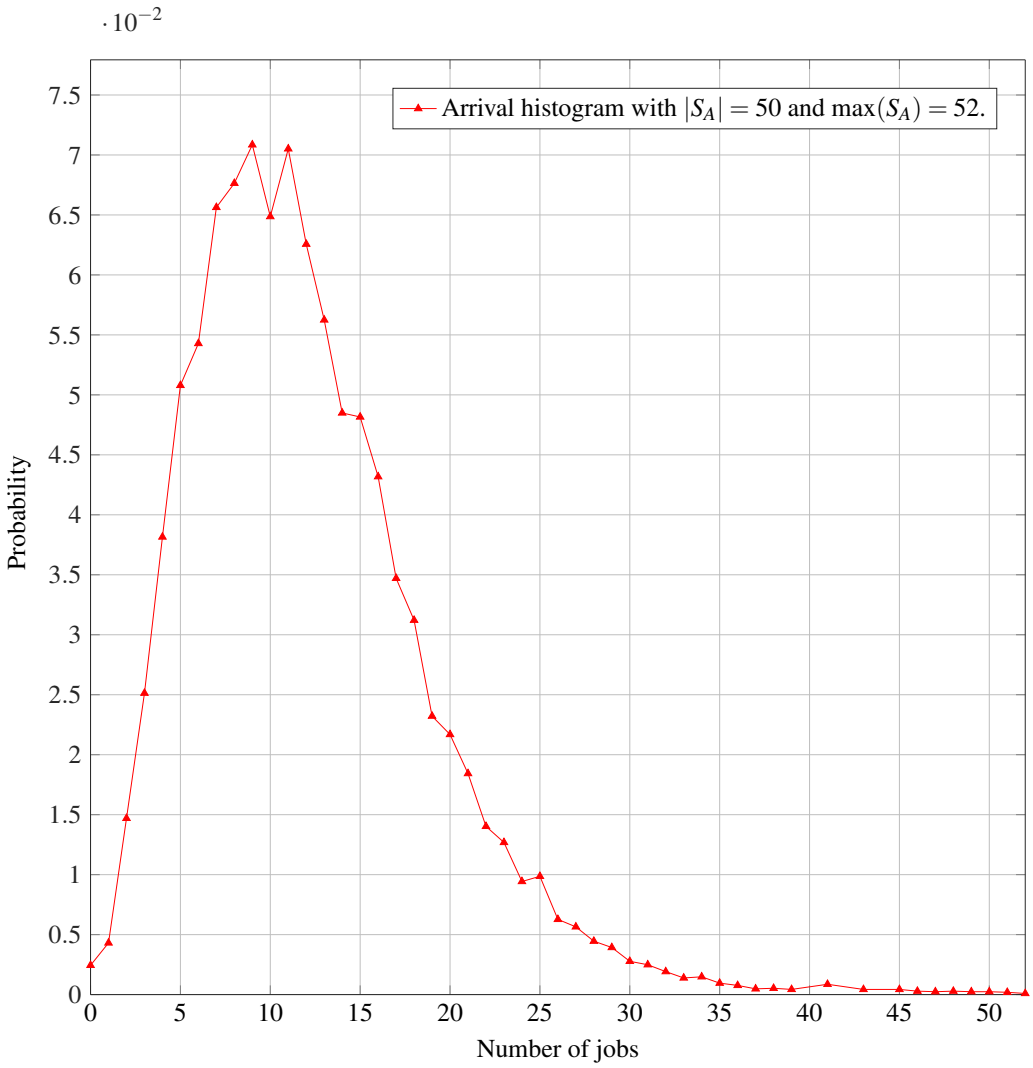


Figure 6.1: Arrival traffic distribution.

6.1.1 First Experimentation

Table 6.4: Settings of the numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	50	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	50	jobs	buffer size
h	100	slots	horizon
w	5	slots	window
$\mathbb{E}(\mathcal{H}_A)$	12.1	jobs	expected arrival jobs per slot
$\lvert S_A \rvert$	50	bins	arrival jobs histogram size
$\max(S_A)$	52	jobs	max arrival jobs per slot
c_M	0.3		cost of energy needed by a server
c_N	0.2		cost of waiting a job
c_{On}	0.1		cost of switching on a server

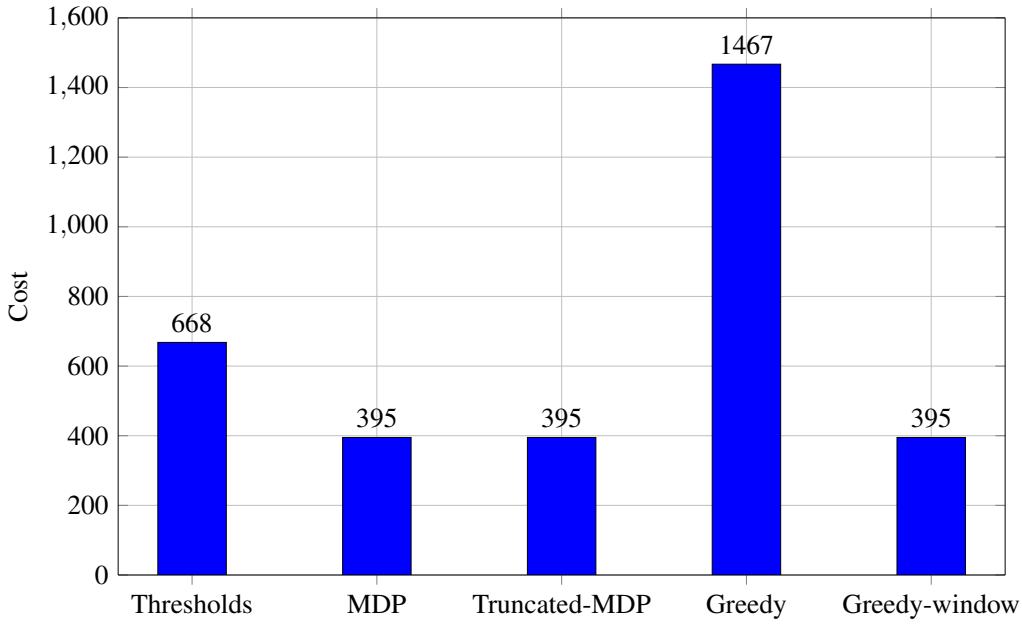


Figure 6.2: Cost comparison: In the first position is shared between MDP, Truncated-MDP and Greedy-window algorithm. After that at the fourth position it's the threshold algorithm with a cost quite far from the optimal one. And the last position is held by the greedy algorithm.

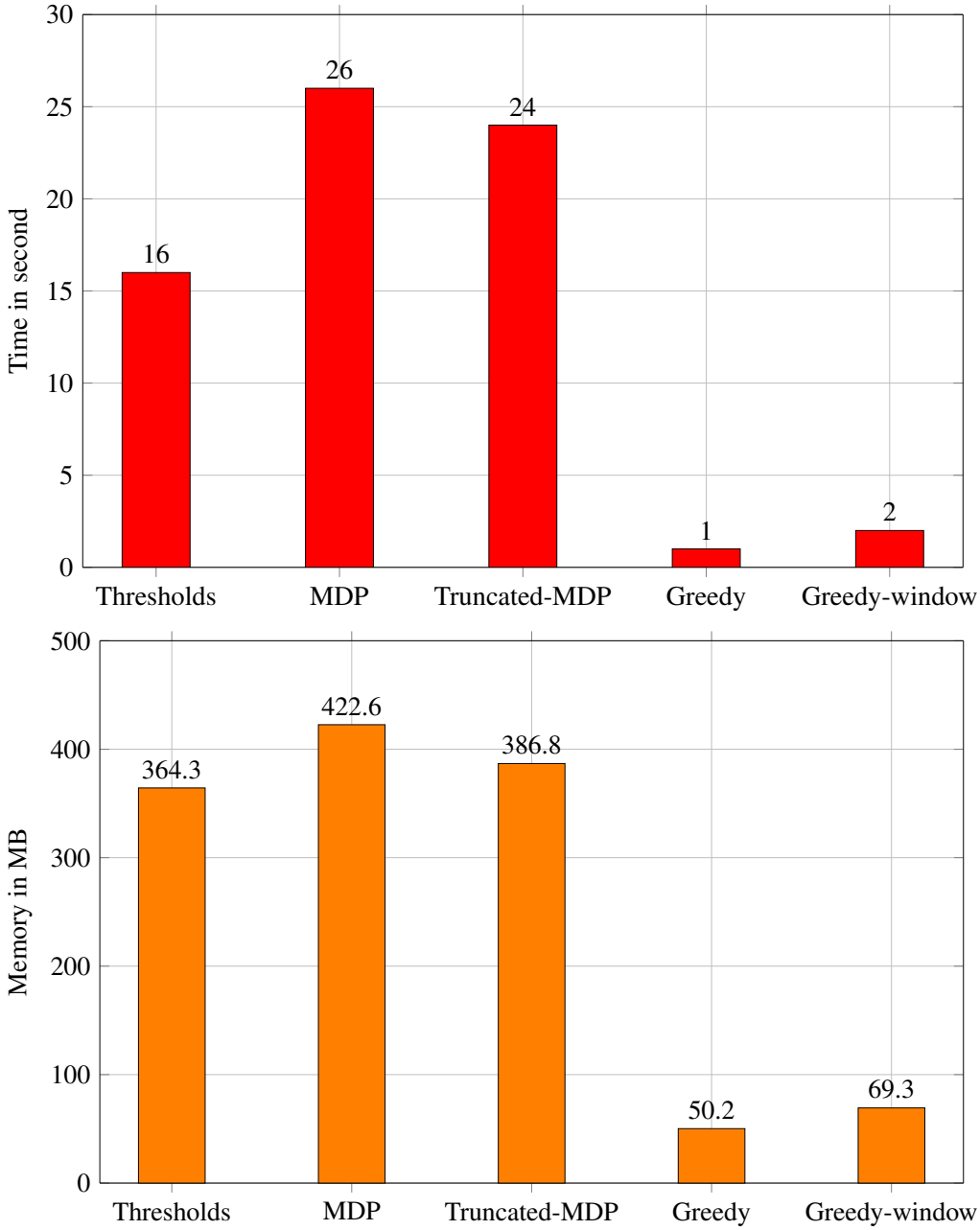


Figure 6.3: Space and computation-time comparison: In the first position, the greedy and the greedy window algorithms consume a negligible amount of space and take a small period of time to compute the strategy. At the second position the threshold method use a medium amount of space and time. And at the last position MDP and Truncated-MDP need a huge amount of memory and time.

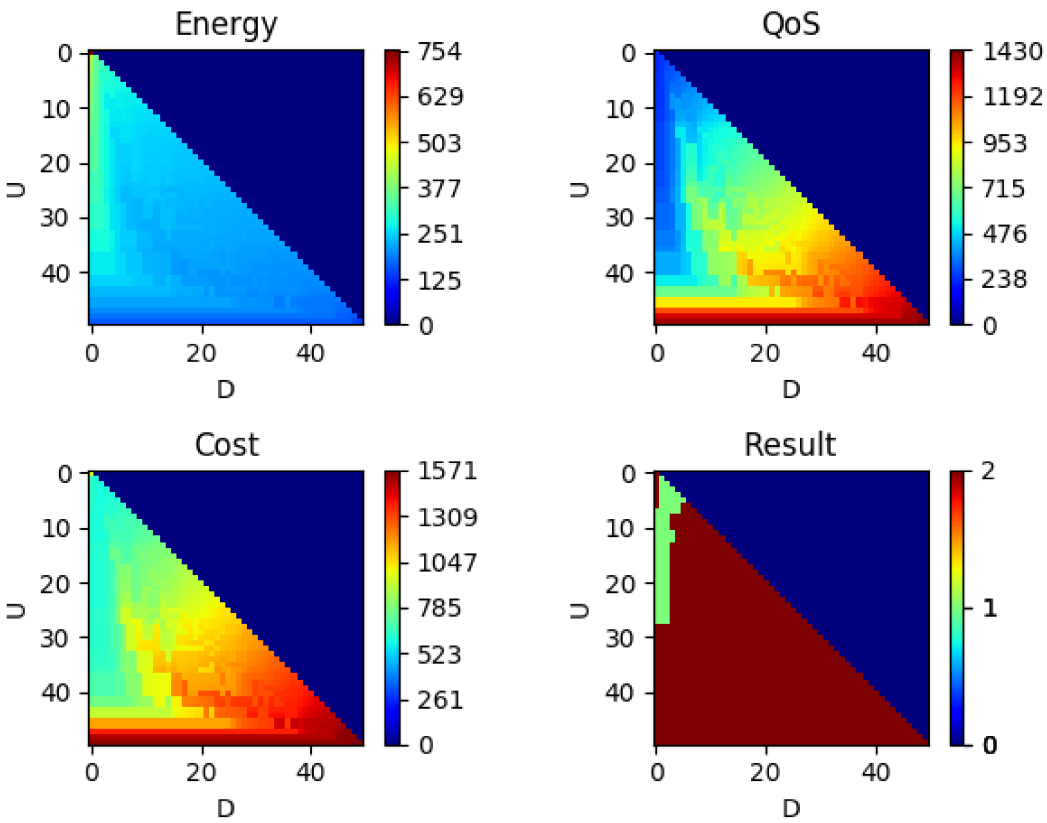


Figure 6.4: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible ($D > U$).

6.1.2 Second Experimentation

In this experimentation we double the buffer size b and we reduce the period of analysis h.

Table 6.5: Settings of the numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	50	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	100	jobs	buffer size
h	50	slots	horizon
w	5	slots	window
$\mathbb{E}(\mathcal{H}_A)$	12.1	jobs	expected arrival jobs per slot
$\lvert S_A \rvert$	50	bins	arrival jobs histogram size
$\max(S_A)$	52	jobs	max arrival jobs per slot
c_M	0.3		cost of energy needed by a server
c_N	0.2		cost of waiting a job
c_{On}	0.1		cost of switching on a server

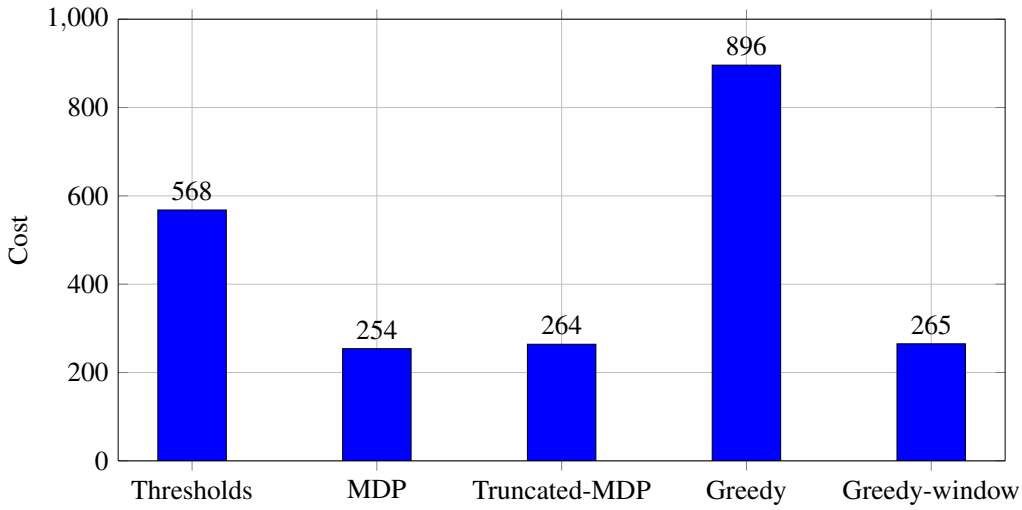


Figure 6.5: Cost comparison: In the first position, the MDP cost is the best one, because this method computes the best policy. The second position is the Truncated-MDP as it computes all possible paths within a window. Then at the third position we found the Greedy-window algorithm which is close to the optimal cost. After that at the fourth position it's the threshold algorithm with a cost quite far from the optimal one. And the last position is held by the greedy algorithm.

6.1.3 Third Experimentation

In this experimentation we increase the total number of servers $\mathcal{M}$.

Table 6.6: Settings of the numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	80	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	100	jobs	buffer size
h	50	slots	horizon
w	5	slots	window
$\mathbb{E}(\mathcal{H}_A)$	12.1	jobs	expected arrival jobs per slot
$\lvert S_A \rvert$	50	bins	arrival jobs histogram size
$\max(S_A)$	52	jobs	max arrival jobs per slot
c_M	0.3		cost of energy needed by a server
c_N	0.2		cost of waiting a job
c_{On}	0.1		cost of switching on a server

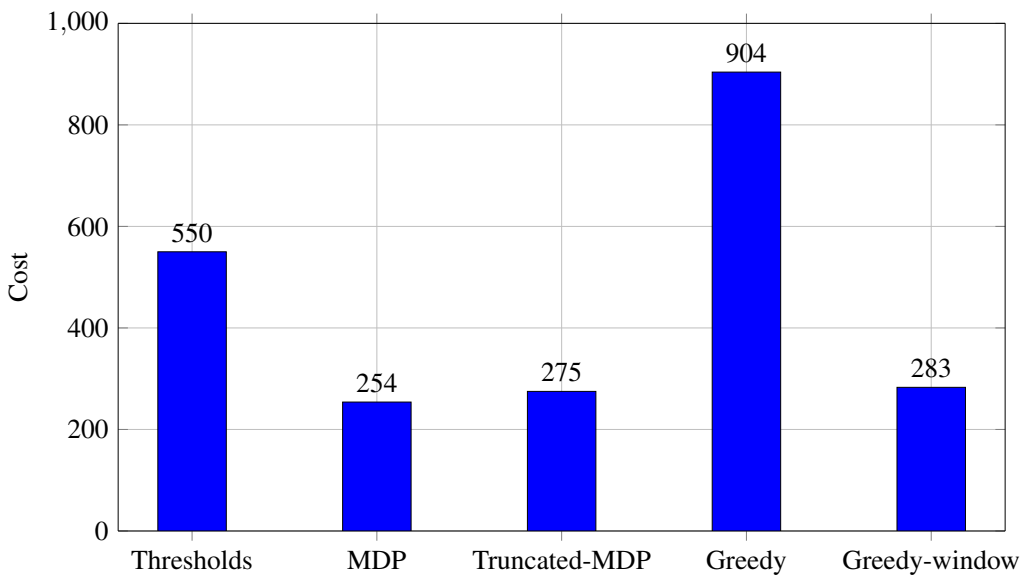


Figure 6.8: Cost comparison: In the first position, the MDP cost is the best one, because this method computes the best policy. The second position is the Truncated-MDP. Then at the third position we found the Greedy-window algorithm which is close to the optimal cost. After that at the fourth position it's the threshold algorithm with a cost quite far from the optimal one. And the last position is held by the greedy algorithm.

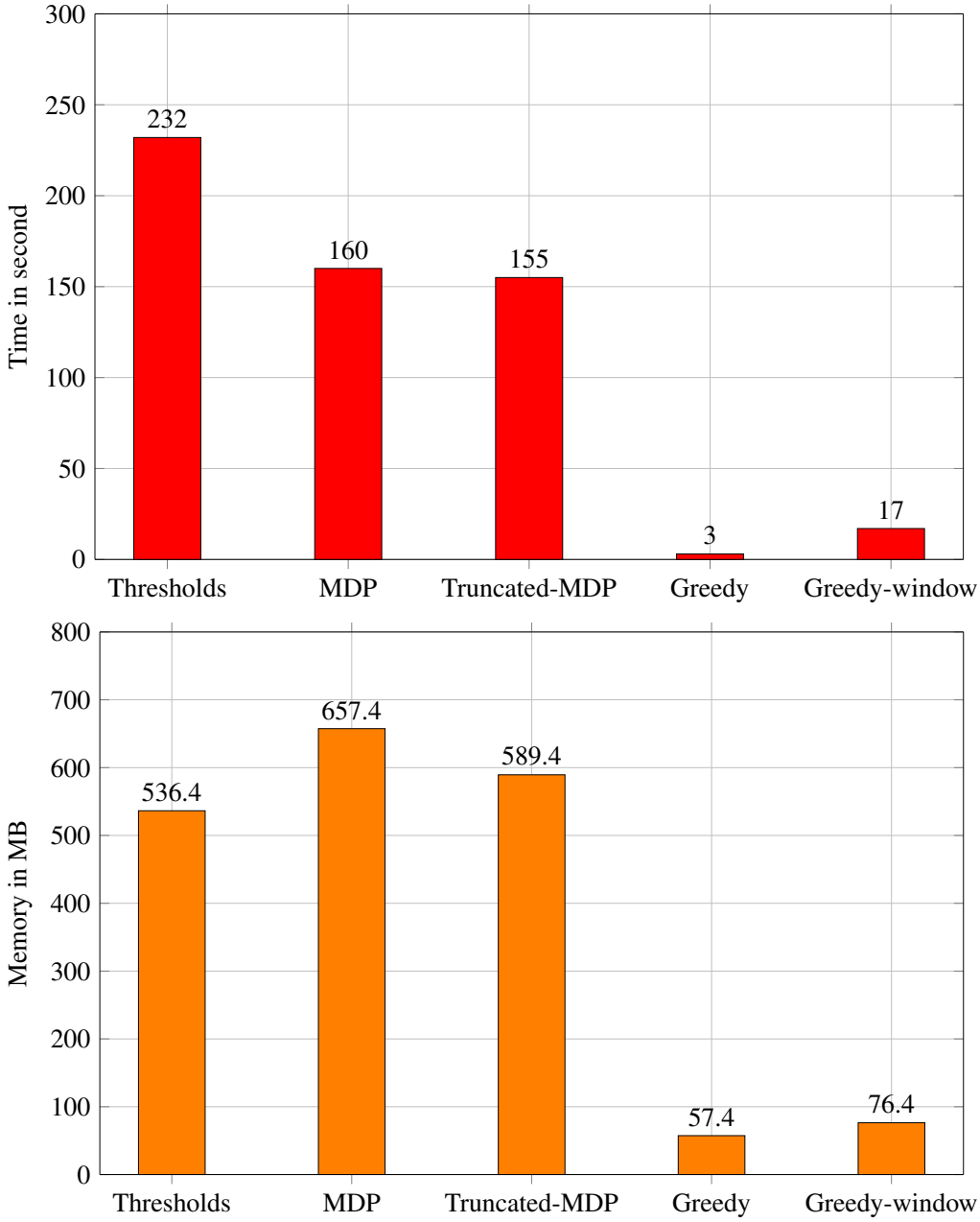


Figure 6.6: Space and computation-time comparison: In the first position, the greedy and the greedy window algorithms consume a negligible amount of space and take a small period of time to compute the strategy. At the second position the threshold method use a medium amount of space and time. And at the last position MDP and Truncated-MDP need a huge amount of memory and time.

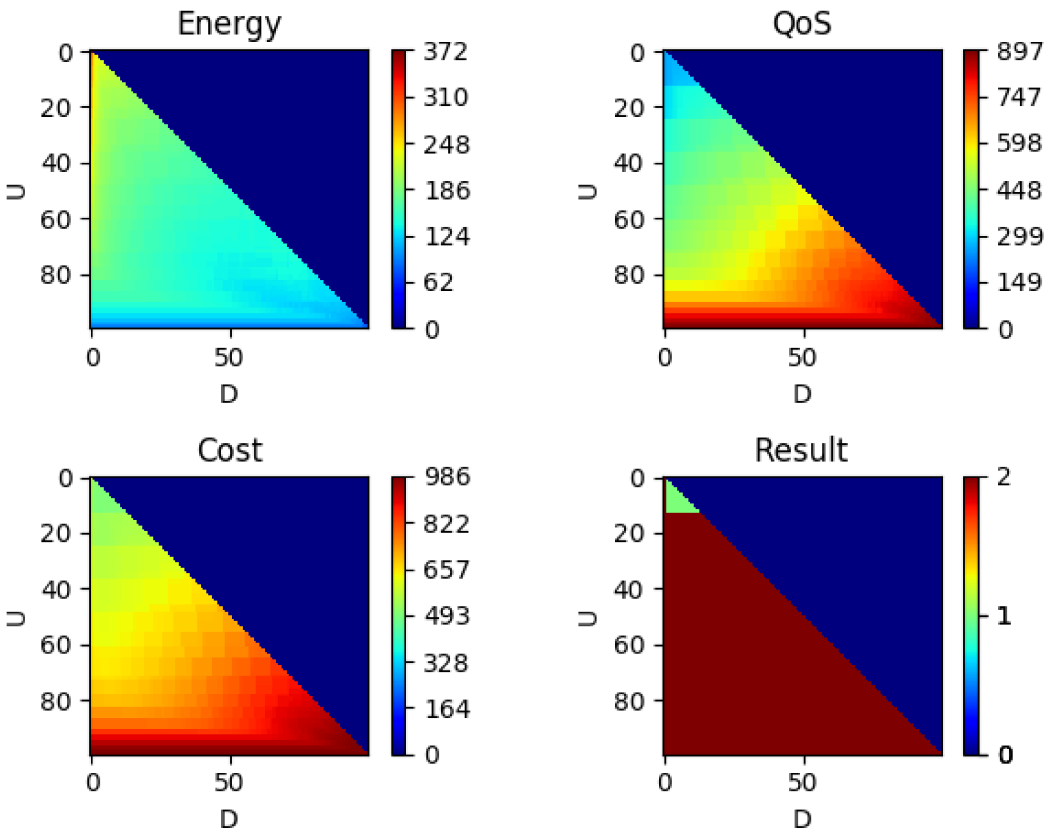


Figure 6.7: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible ($D > U$).

6.1.4 Fourth Experimentation

In this experimentation we increase further the total number of servers $\mathcal{M}$. We also double the horizon h.

Table 6.7: Settings of the numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	100	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	100	jobs	buffer size
h	100	slots	horizon
w	5	slots	window
$\mathbb{E}(\mathcal{H}_A)$	12.1	jobs	expected arrival jobs per slot
$\lvert S_A \rvert$	50	bins	arrival jobs histogram size
$\max(S_A)$	52	jobs	max arrival jobs per slot
c_M	0.3		cost of energy needed by a server
c_N	0.2		cost of waiting a job
c_{On}	0.1		cost of switching on a server

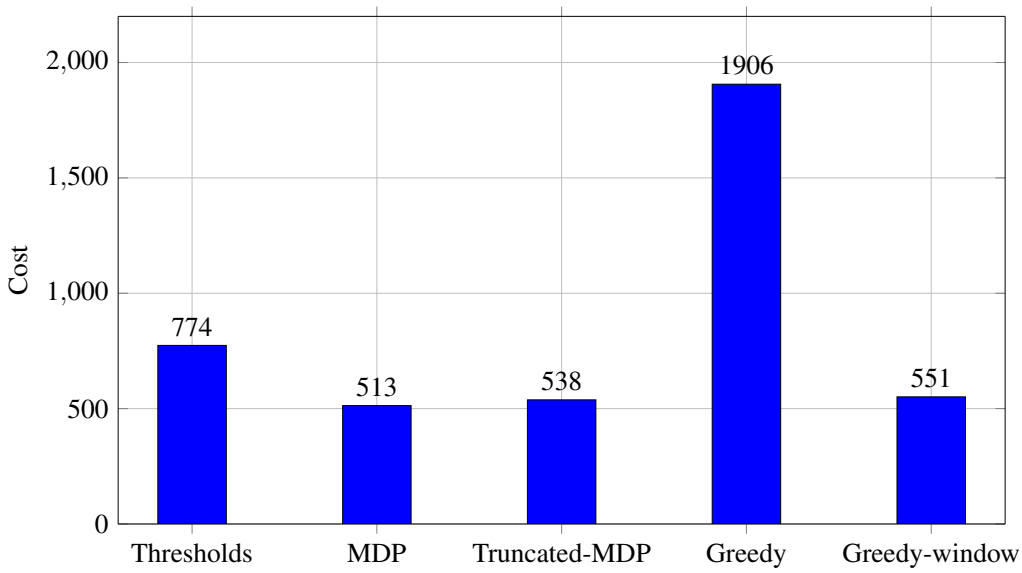


Figure 6.11: Cost comparison: In the first position, the MDP cost is the best one, because this method computes the best policy. The second position is the Truncated-MDP. Then at the third position we found the Greedy-window algorithm which is close to the optimal cost. After that at the fourth position it's the threshold algorithm with a cost quite far from the optimal one. And the last position is held by the greedy algorithm.

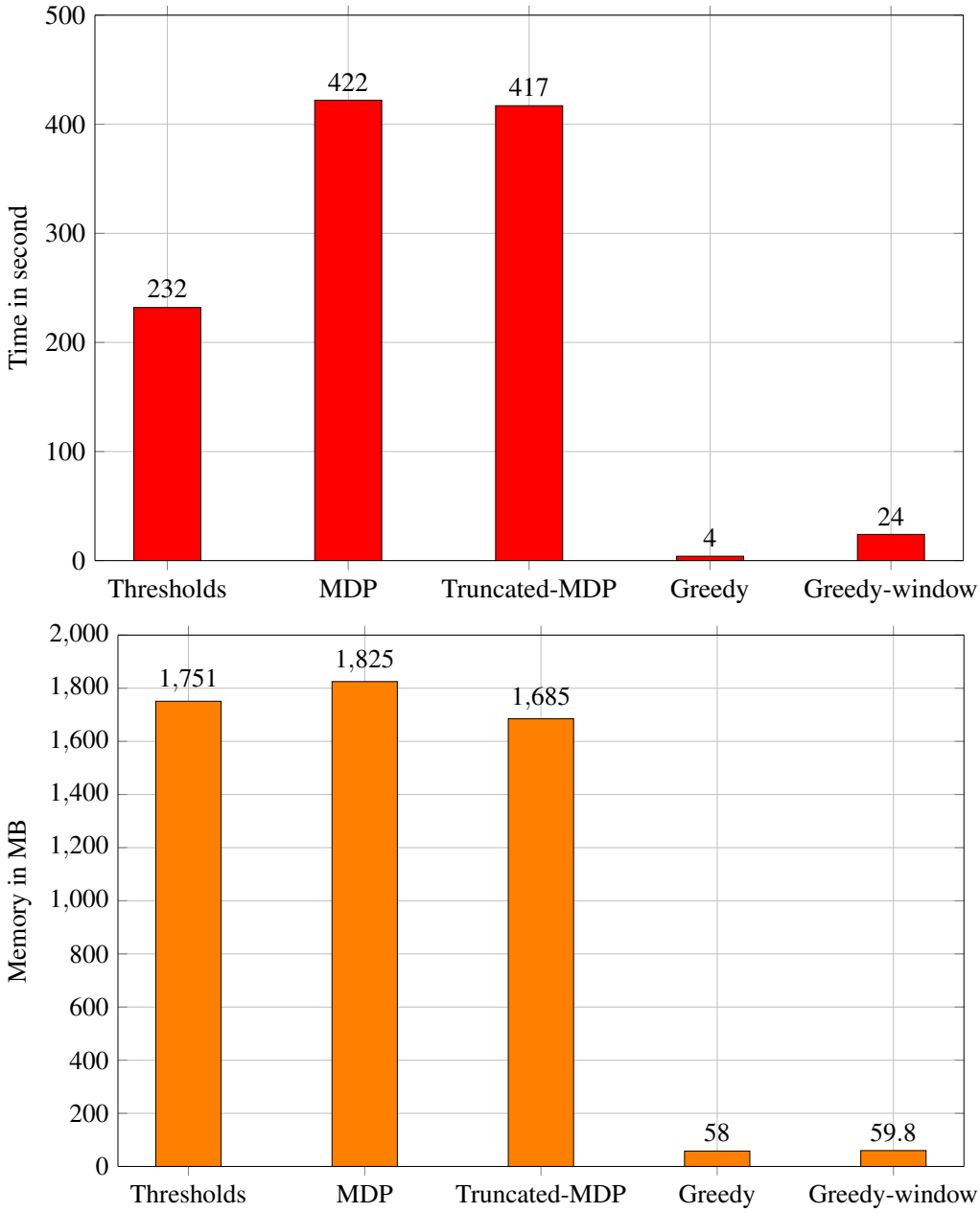


Figure 6.9: Space and computation-time comparison: In the first position, the greedy and the greedy window algorithms consume a negligible amount of space and take a small period of time to compute the strategy. At the second position the threshold method use a medium amount of space and time. And at the last position MDP and Truncated-MDP need a huge amount of memory and time.

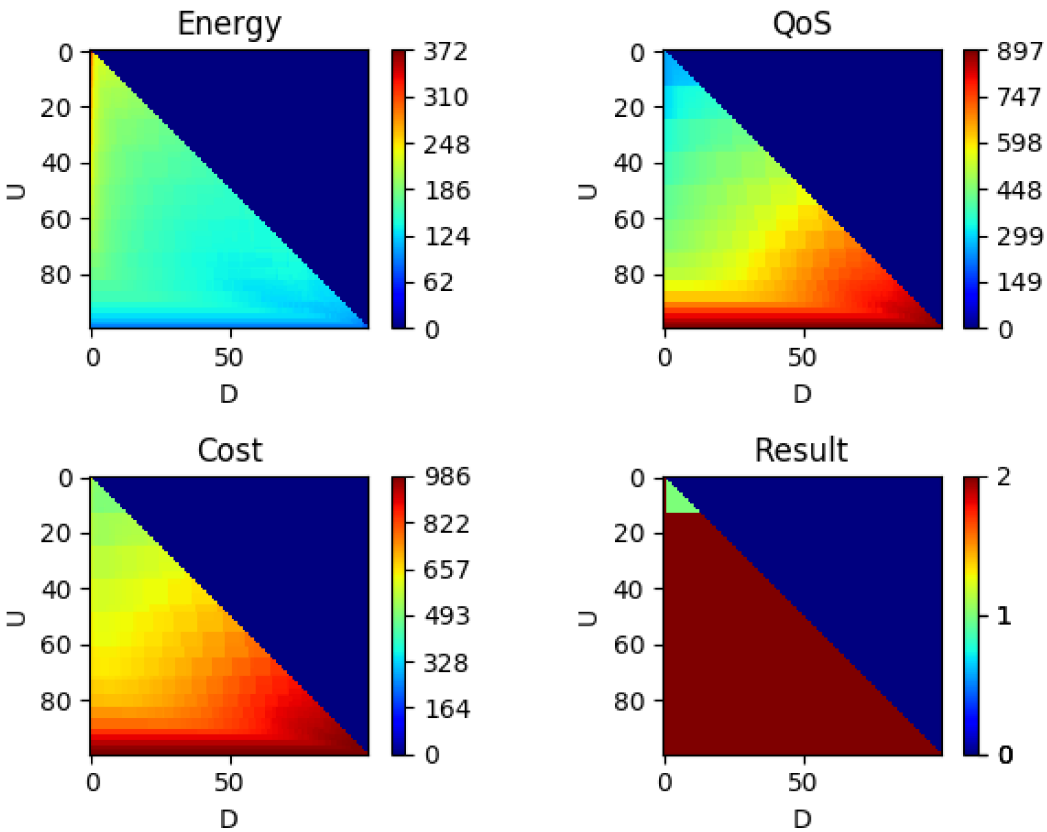


Figure 6.10: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible ($D > U$).

6.1.5 Fifth Experimentation

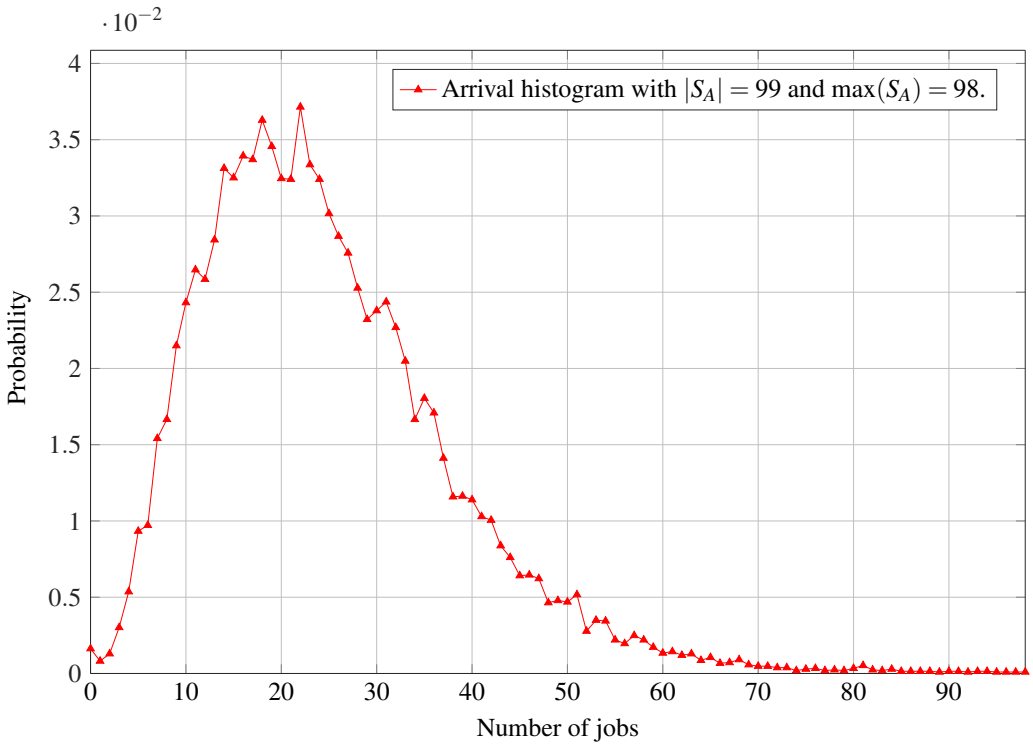


Figure 6.14: Arrival traffic distribution.

Table 6.8: Settings of the numerical analysis.

Parameters	Value	Unit	Description
$\mathcal{M}$	100	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	100	jobs	buffer size
h	200	slots	horizon
w	5	slots	window
$\mathbb{E}(\mathcal{H}_A)$	24.69	jobs	expected arrival jobs per slot
$\lvert S_A \rvert$	99	bins	arrival jobs histogram size
$\max(S_A)$	98	jobs	max arrival jobs per slot
c_M	0.1		cost of energy needed by a server
c_N	0.1		cost of waiting a job
c_{On}	0.1		cost of switching on a server

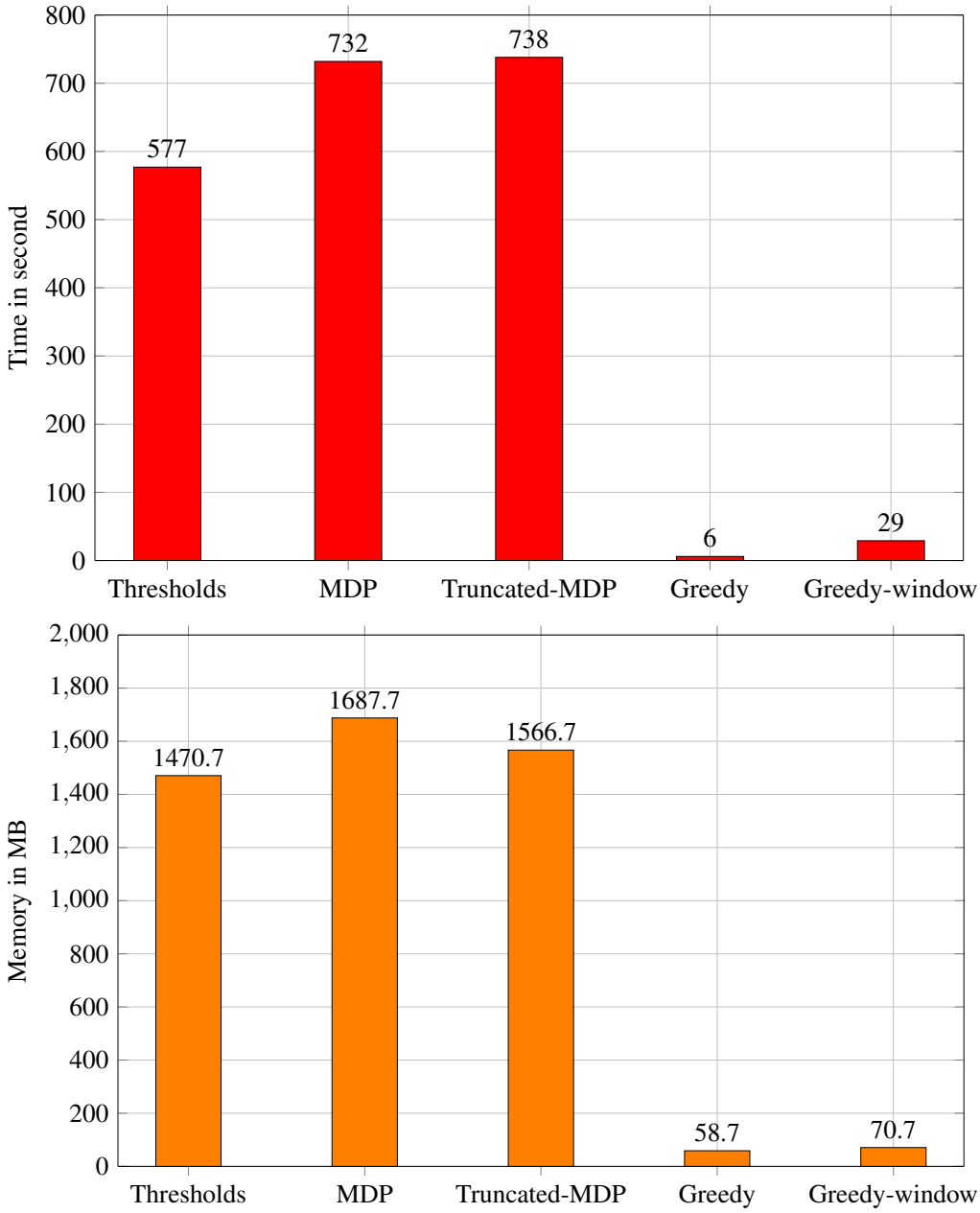


Figure 6.12: Space and computation-time comparison: In the first position, the greedy and the greedy window algorithms consume a negligible amount of space and take a small period of time to compute the strategy. At the second position the threshold method use a medium amount of space and time. And at the last position MDP and Truncated-MDP need a huge amount of memory and time.

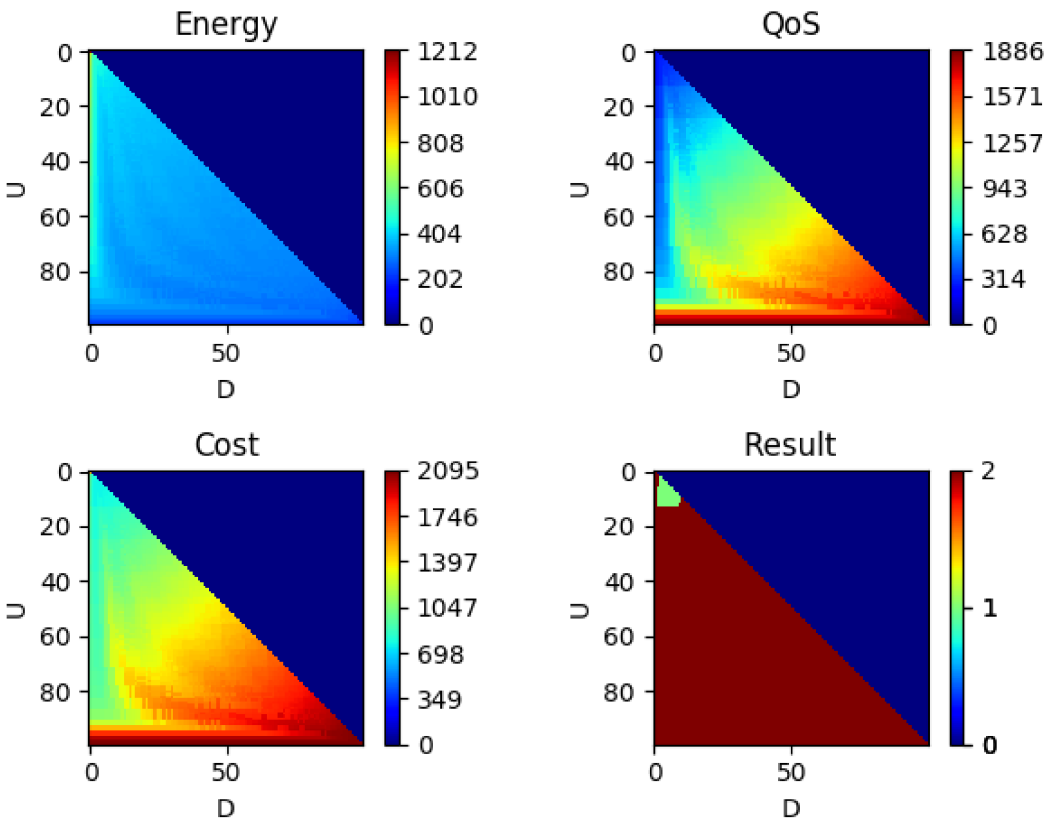


Figure 6.13: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible ($D > U$).

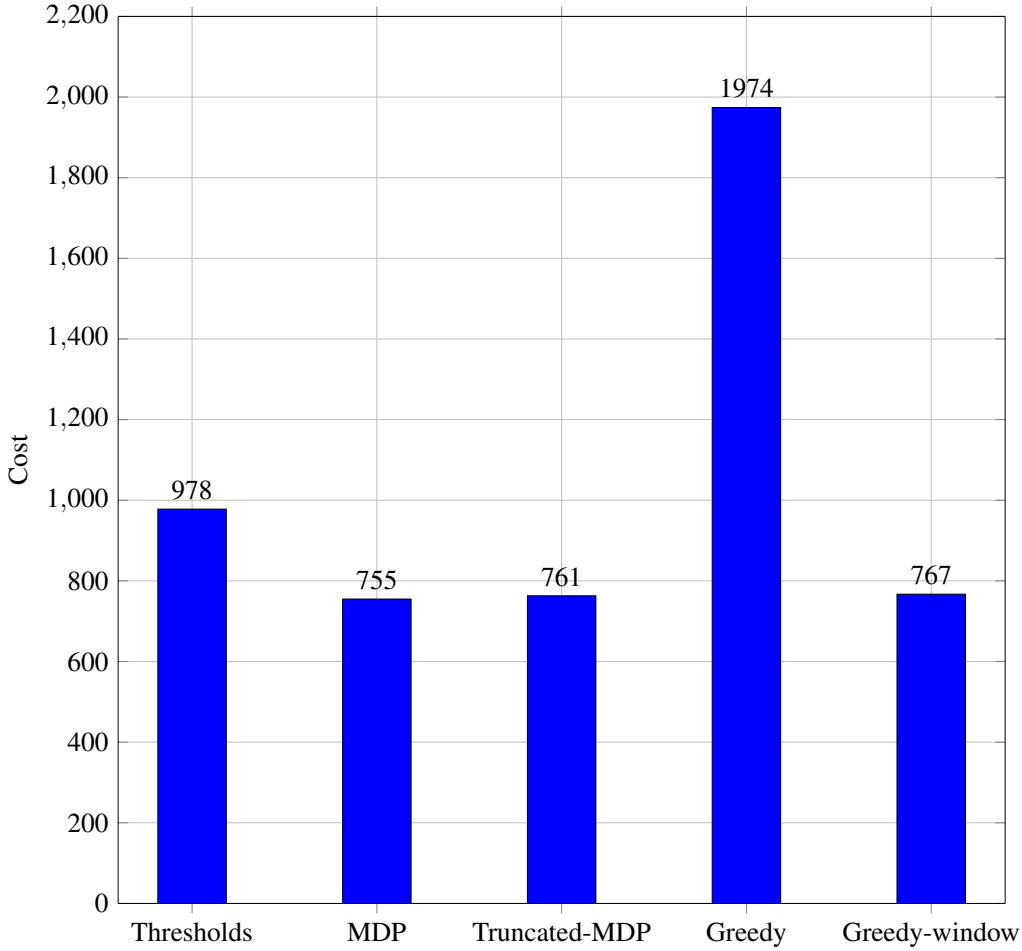


Figure 6.15: Cost comparison: In the first position, the MDP cost is the best one, because this method computes the best policy. The second position is the Truncated-MDP. Then at the third position we found the Greedy-window algorithm which is close to the optimal cost. After that at the fourth position it's the threshold algorithm with a cost quite far from the optimal one. And the last position is held by the greedy algorithm.

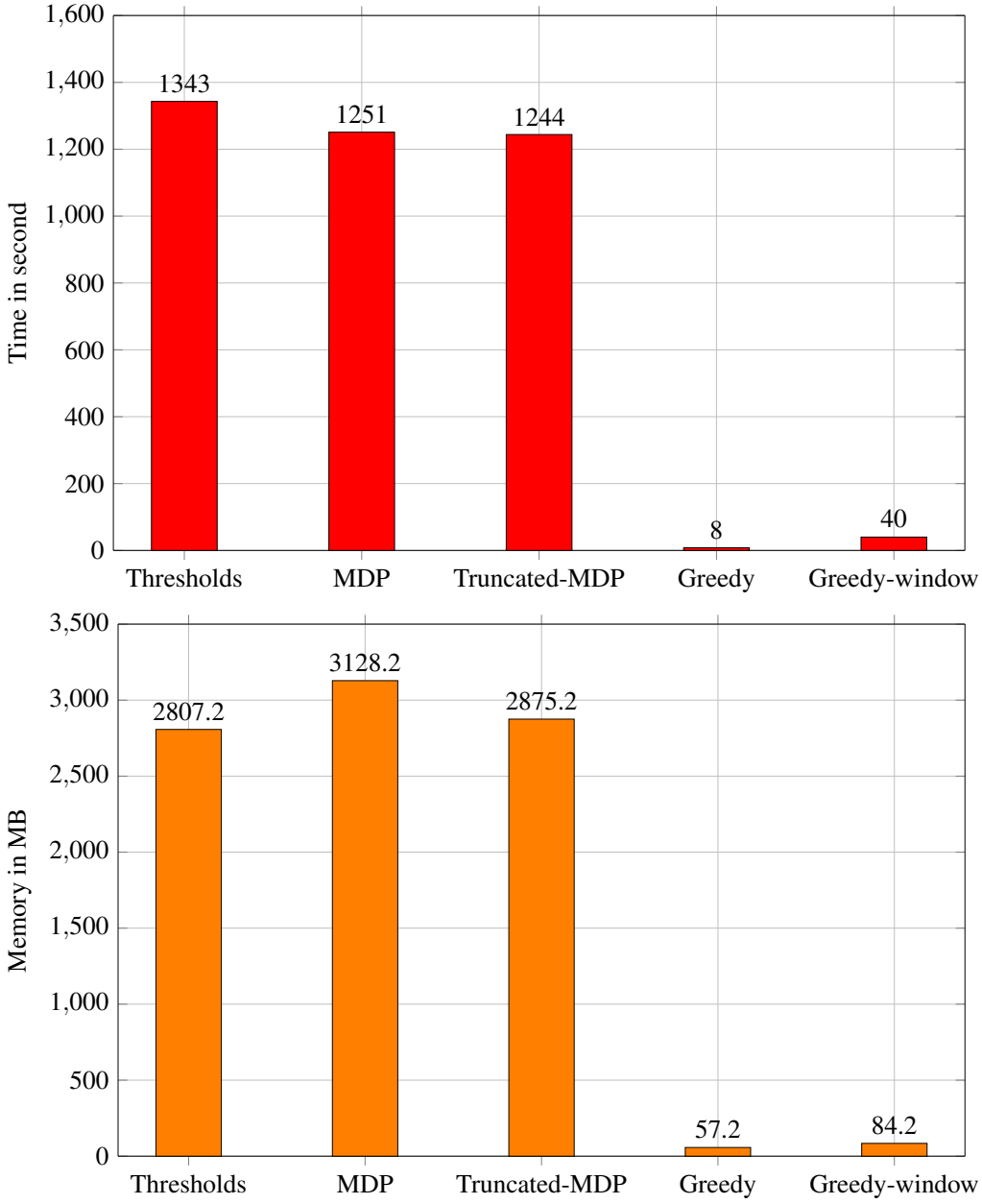


Figure 6.16: Space and computation-time comparison: In the first position, the greedy and the greedy window algorithms consume a negligible amount of space and take a small period of time to compute the strategy. At the second position the threshold method use a medium amount of space and time. And at the last position MDP and Truncated-MDP need a huge amount of memory and time.

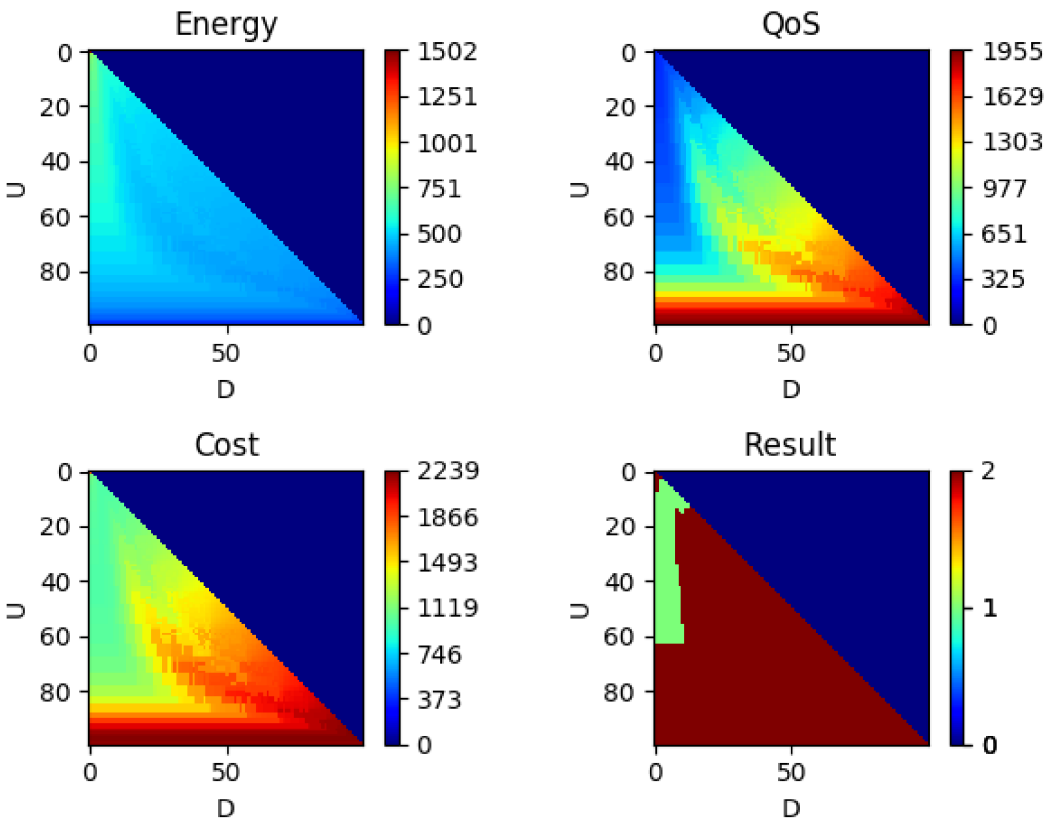


Figure 6.17: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible ($D > U$).

6.1.6 Sixth Experimentation

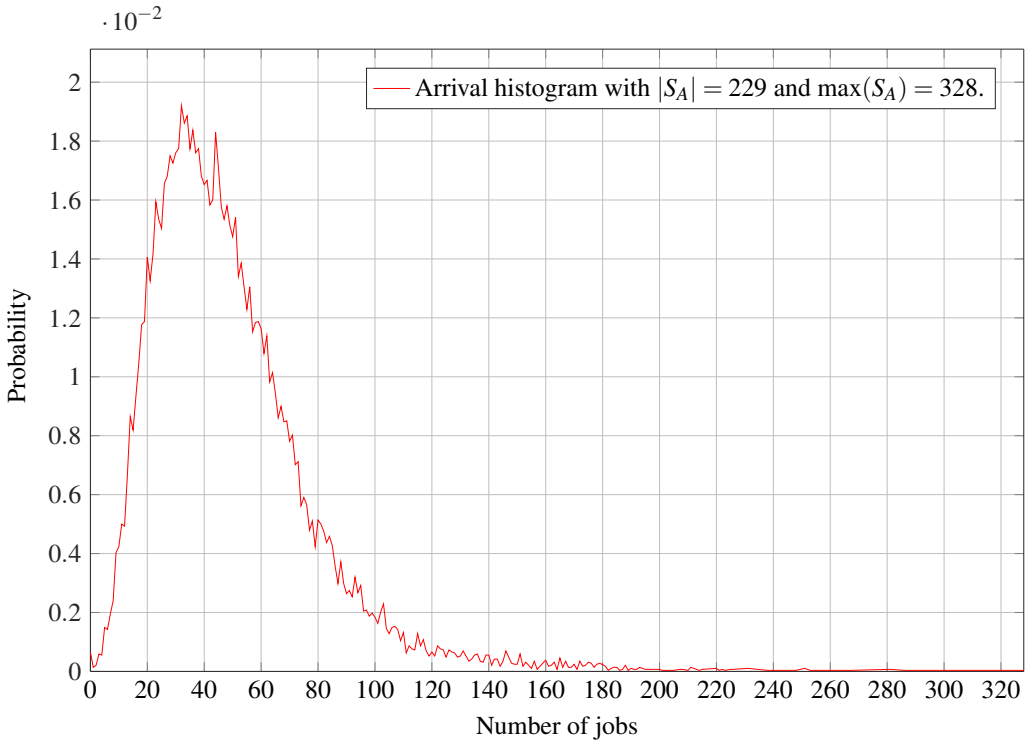


Figure 6.18: Arrival traffic distribution.

Table 6.9: Settings of the numerical analysis.

Parameters	**Value**	**Unit**	**Description**
$\mathcal{M}$	200	servers	total number of servers
d	1	jobs/server	processing capacity of a server
b	200	jobs	buffer size
h	200	slots	horizon
w	5	slots	window
$\mathbb{E}(\mathcal{H}_A)$	48.92	jobs	expected arrival jobs per slot
$\lvert S_A \rvert$	229	bins	arrival jobs histogram size
$\max(S_A)$	328	jobs	max arrival jobs per slot
c_M	0.1		cost of energy needed by a server
c_N	0.1		cost of waiting a job
c_{On}	0.1		cost of switching on a server

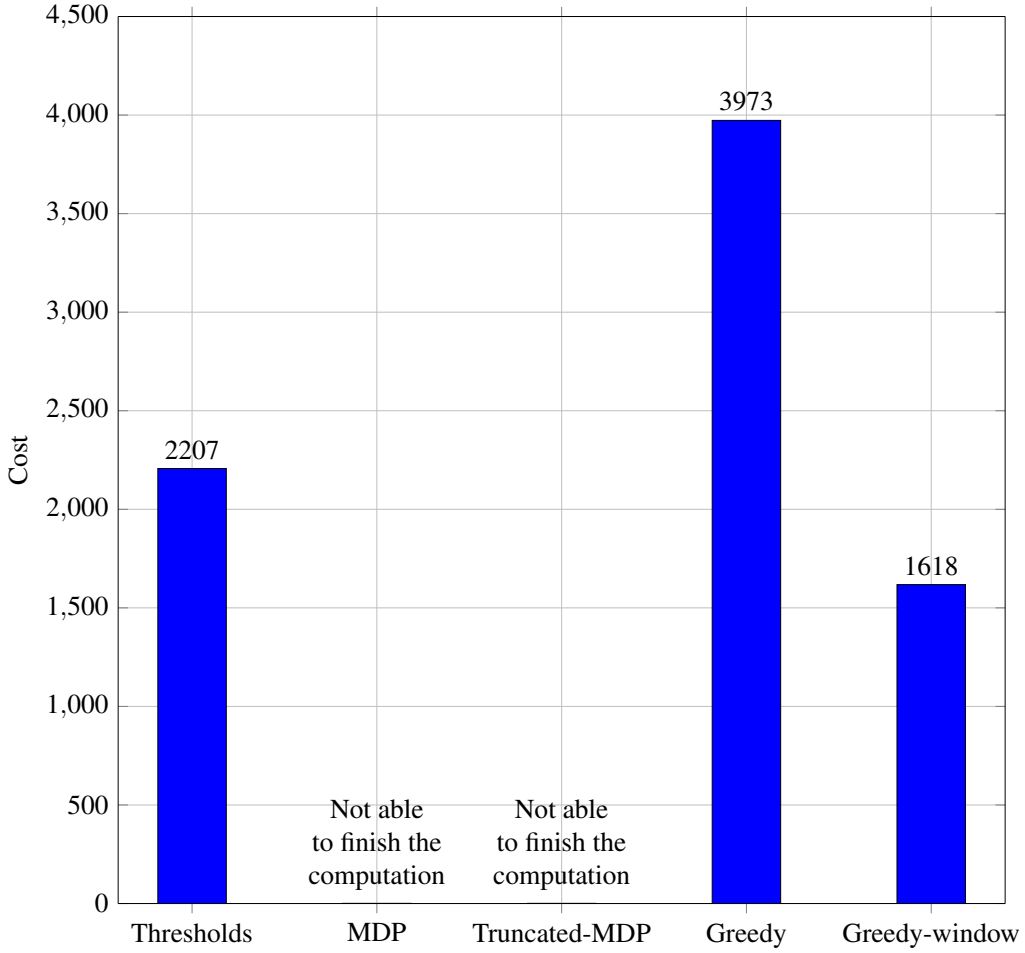


Figure 6.19: Cost comparison: In the first position, we found the greedy-window algorithm. After that at the second position it's the greedy algorithm with a cost quite far from the greedy-window one. And the last position is held by the threshold method. MDP and Truncated-MDP were executed for a long period of time and because of the out of memory, they were not able to finish the computation of the strategy.

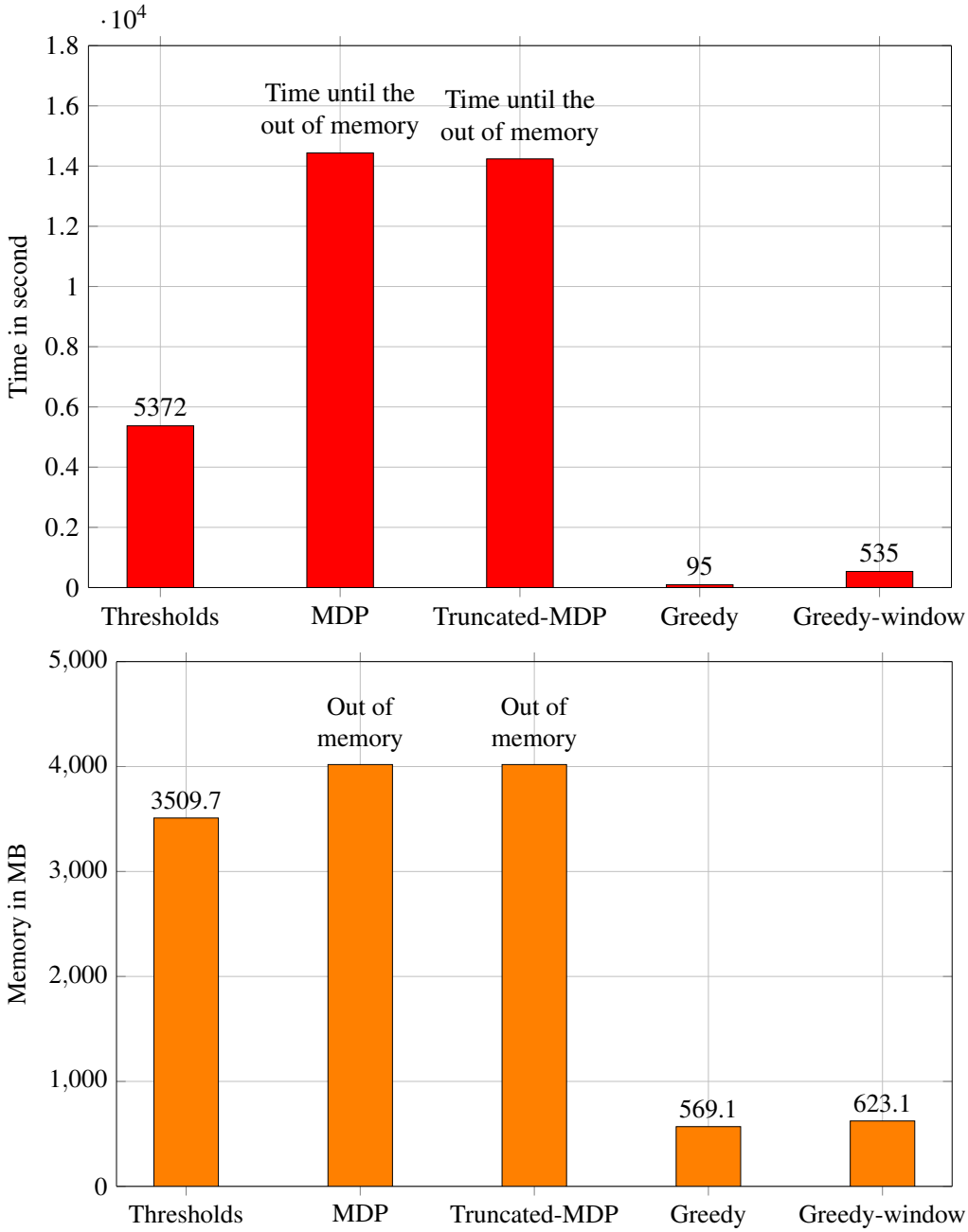


Figure 6.20: Space and computation-time comparison: In the first position, the greedy and the greedy window algorithms consume a negligible amount of space and take a small period of time to compute the strategy. At the second position the threshold method use a medium amount of space and time. And at the last position MDP and Truncated-MDP need a very huge amount of memory and time which are not available in our machine.

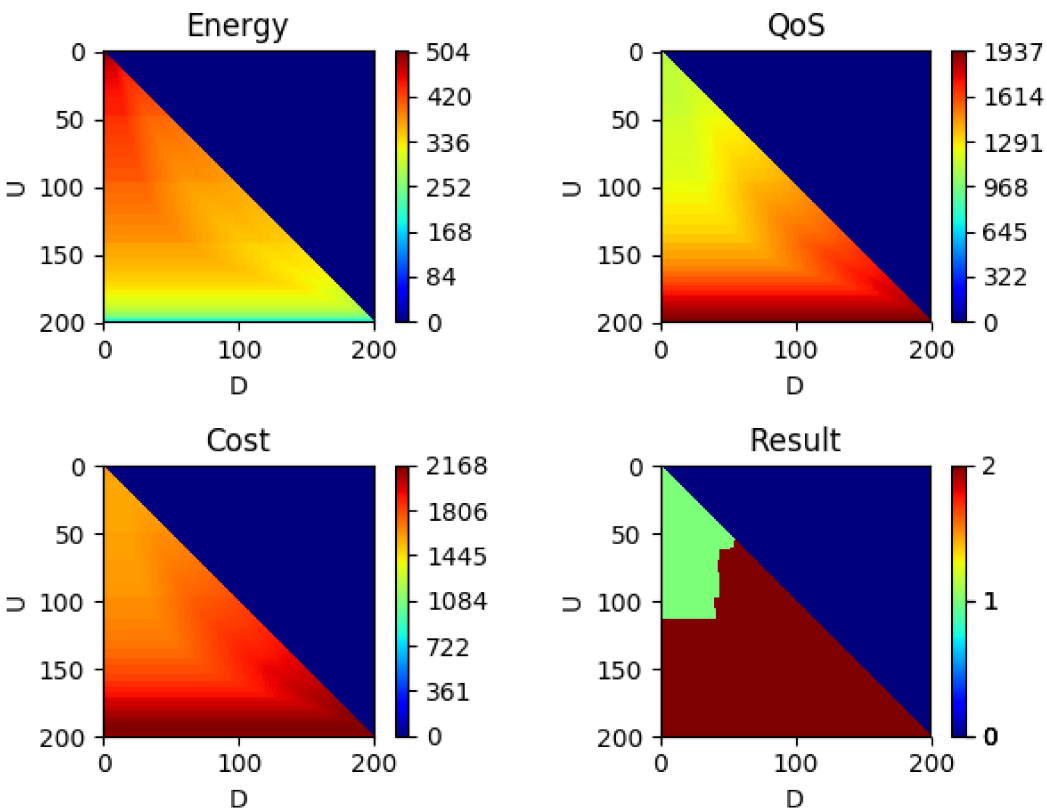


Figure 6.21: Threshold results: The top left sub-figure shows that the cost of energy consumption is more important for small values of U and D. The top right sub-figure shows that the cost of QoS is more important for big values of U and D. The bottom left sub-figure shows the combination of the energy and QoS costs. The bottom right sub-figure shows the green region where the cost is minimal (with a margin of 5%: $min_cost \leq cost \leq min_cost + min_cost \times 5\%$), the red region where the cost is worst and the blue region where the (U, D) couple is not possible ($D > U$).

Additionally we notice that the analysis under greedy-window algorithm can be on-line[1], dynamic[2] and adaptive[3], that means: every slot, we take the parameters of the system (the arrival could be modeled with a histogram that change over time) and we calculate in real time a sub-optimal strategy (close to the optimal one) that gives the number of machines to be turned depending on the number of waiting jobs. Finally, where comparing the time-space complexity of MDP and greedy-window algorithm, we observe that MDP needs a big structure to store the matrix state/action transition and a huge amount of time to solve Bellman's equation in order to return the optimal policy. Our greedy-window algorithm gives a policy close to the optimal one (MDP is only around 2% better than greedy-window algorithm, see for instance Figure 6.22). Greedy-window algorithm is more efficient because it is faster and avoids state space explosion.

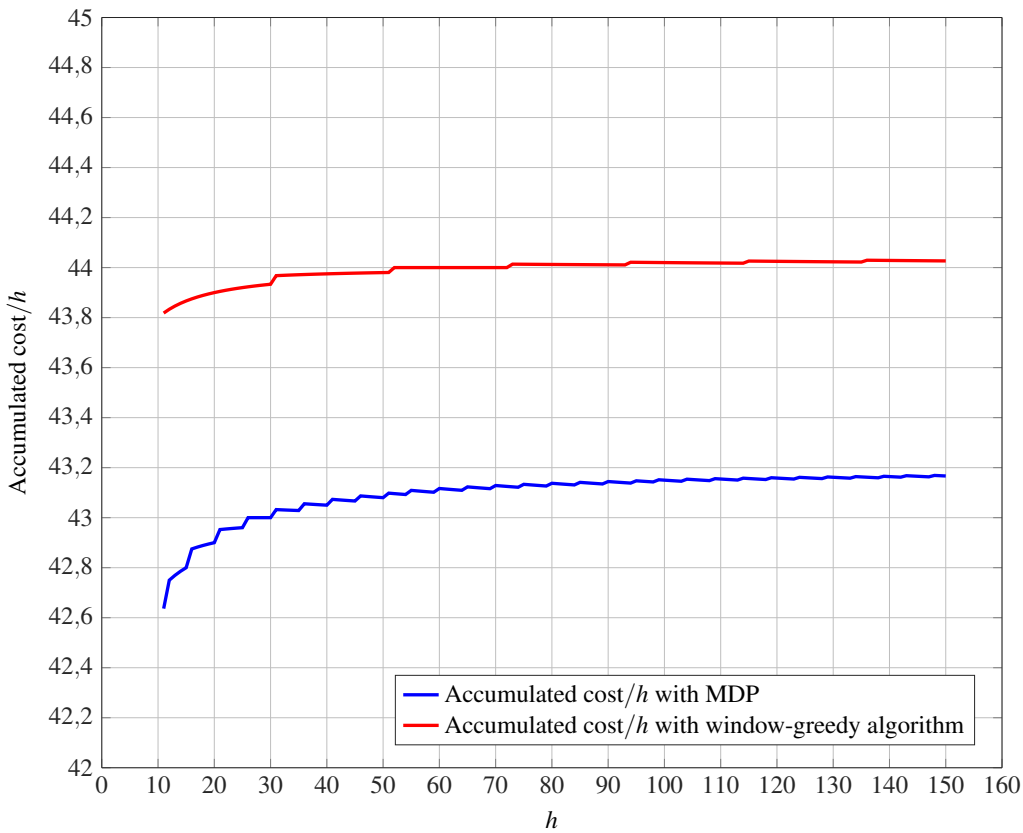


Figure 6.22: Average cost per slot for MDP and window-greedy algorithm when varying h.

[1] An on-line algorithm is able to compute/do the requested value/task while the input is given piece by piece and not entirely.

[2] A dynamic algorithm is able to handle the fact that each time the input data is modified.

[3] An adaptive algorithm is able to handle the fact that the input data may change over time.

6.2 Contributions

This thesis is a contribution to the modeling of the exact energy optimization in data centers in the context of the discrete time with inputs/outputs jobs modeled by general discrete distributions obtained from real traces or empirical data when considering various real aspects of data centers like heterogeneity of the servers (see Section 4.2) or their latency (see Section 4.3).

A first possible solution is to try to express the optimization under simple form as what we done when considering the approaches based on thresholds. We show that threshold-based solutions, which are by definition monotones (see Definition 4.2), are not optimal (see Section 4.1.5.5), use less space but require a large computation time. So they are not dynamic and they can not be online.

In fact, as a second solution, we implemented and analyzed theoretically and practically exact optimization when using MDP in Chapter 4. We show how it causes the problem of the explosion of the use of resources in terms of computation time and memory space (see Theorems 4.8 and 4.9), which makes it impractical, non-scalable, non-dynamic, and non-online. Otherwise we prove that the optimal policy is not monotone which means that it can not be expressed as a double threshold structure (see Theorem 4.6).

Our third solution explained along Chapter 5. is to give an efficient algorithm that computes a quasi-optimal solution, better than the MDP and the thresholds based solutions, in terms of time computation and in memory space. This makes this solution practical, scalable, dynamic. Additionally, as minor contributions:

1. We found that heterogeneity can be exploited to effectively reduce energy consumption (see Section 4.2.5).
2. We have also proposed an algorithm that allows to find the best sampling period of real traces to have an i.i.d. distribution based on statistical tests (see the end of Section 1.3.3).
3. We implemented:
   (a) a tool that allow us to generate automatically PRISM specifications of our different Dynamic Power Management models. With this tool we can perform a lot of experimentation by easily changing parameters without rewriting by hand thousand lines of PRISM code. This tool was used to prepare all results in Chapter 4.
   (b) a tool that implemented the methodology based on threshold policy presented in Section 3.2 and the methodology based on greedy-window algorithm presented in Chapter 5.

6.3 Future work

After having presented how we can model, formalize and solve the problem of energy optimization in data centers according to several approaches, and after having analyzed them theoretically and numerically, as a perspective, we intend to test our solutions and compare them in a real context by deploying them in a true heterogeneous server cluster architecture. The manager is a machine connected to the network of the cluster and host a software application based on the greedy-window that observes the system and decides what action to take. This manager must have access: i) to the cluster network to be able to ask a particular server to turn on or off, ii) to the flow of incoming requests to the system to continually update the distribution of arrivals using the histograms, iii) to the energy consumption of the servers, iv) to the machine dispatcher queries on different servers to estimate the number of queries pending and rejected requests.

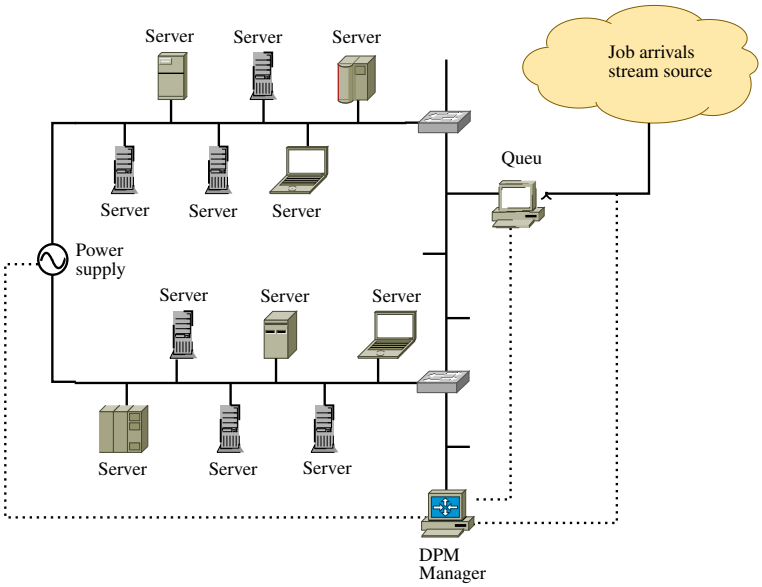


Figure 6.23: Deployment schema.

Otherwise, we are working to include all the tools developed in this work in one tool that implements all the optimization methods described in this thesis. Additionally, we plan to test our greedy-window algorithm for other optimization problems in other domains other then reducing energy in Data Centers.

Bibliography

[AA11] Michele Arnold and Göran Andersson. Model predictive control of energy storage including uncertain forecasts. In *Power systems computation conference (PSCC), Stockholm, Sweden*, volume 23, pages 24–29. Citeseer, 2011.

[ACFP13] Farah Aït-Salaht, Hind Castel-Taleb, Jean-Michel Fourneau, and Nihal Pekergin. Stochastic bounds and histograms for network performance analysis. In *Computer Performance Engineering - 10th European Workshop, EPEW Italy*, volume 8168 of *Lecture Notes in Computer Science*, pages 13–27. Springer, 2013.

[AEM13] K. Aidarov, Paul D. Ezhilchelvan, and Isi Mitrani. Energy-aware management of customer streams. *Electr. Notes Theor. Comput. Sci.*, 296:199–210, 2013.

[ASCTFP16] Farah Ait Salaht, Hind Castel-Taleb, Jean Fourneau, and Nihal Pekergin. Performance analysis of a queue by combining stochastic bounds, real traffic traces and histograms. *The Computer Journal*, 59, 06 2016.

[Bal11] Ayre R. W. Hinton K. Baliga, J. Green cloud computing: Balancing energy in processing, storage, and transport. *Proceedings of the IEEE*, 99(1):149–167, 2011.

[BAM10] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010.

[Bay16] Marziyeh Bayati. Managing energy consumption and quality of service in data centers. In *Proceedings of the 5th International*

Conference on Smart Cities and Green ICT Systems, pages 293–301, 2016.

[Bay18] Marziyeh Bayati. Power management policy for heterogeneous data center based on histogram and discrete-time MDP. *Electr. Notes Theor. Comput. Sci.*, 337:5–22, 2018.

[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 preprint arXiv:1006.0308*, 2010.

[BBPDM99] Luca Benini, Alessandro Bogliolo, Giuseppe Paleologo, and Giovanni De Micheli. Policy optimization for dynamic power management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(6):813–833, 1999.

[BDF+16] Marziyeh Bayati, Mohamed Dahmoune, Jean-Michel Fourneau, Nihal Pekergin, and Dimitrios Vekris. A tool based on traffic traces and stochastic monotonicity to analyze data centers and their energy consumption. *EAI Endorsed Trans. Energy Web*, 3(10):e3, 2016.

[Bel57] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[Ber95] Dimitri P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena Scientific Belmont, MA, 1995.

[BF12] Ana Bušić and Jean-Michel Fourneau. Monotonicity and performance evaluation: Applications to high speed and mobile networks. *Cluster Computing*, 15(4):401–414, December 2012.

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

[Bla04] Paul E. Black. *Dictionary of algorithms and data structures*. National Institute of Standards and Technology Gaithersburg, 2004.

[Cis15] Cisco. Visual networking index (VNI) forecast highlights tool. Technical report, Cisco, 2015.

[CM12] Steven Chu and Arun Majumdar. Opportunities and challenges for a sustainable energy future. *nature*, 488(7411):294, 2012.

[DM10] Dmytro Dyachuk and Michele Mazzucco. On allocation policies for power and performance. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 313–320. IEEE, 2010.

[Efr04] Dmitry Efrosinin. *Controlled queueing systems with heterogeneous servers*. PhD thesis, Univ. Trier, FB 4, Informatik, 2004.

[EMSM17] Reza Entezari-Maleki, Leonel Sousa, and Ali Movaghar. Performance and power modeling and evaluation of virtualized servers in IaaS clouds. *Information Sciences*, 394:106–122, 2017.

[Erl17] Agner Krarup Erlang. Solution of some problems in the theory of probabilities of significance in automatic telephone exchanges. *Post Office Electrical Engineer's Journal*, 10:189–197, 1917.

[FS12] Eugene A. Feinberg and Adam Shwartz. *Handbook of Markov decision processes: methods and applications*, volume 40. Springer Science & Business Media, 2012.

[GDHBSW13] Anshul Gandhi, Sherwin Doroudi, Mor Harchol-Balter, and Alan Scheller-Wolf. Exact analysis of the M/M/k/setup class of Markov chains via recursive renewal reward. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 153–166. ACM, 2013.

[GGHBK10] Anshul Gandhi, Varun Gupta, Mor Harchol-Balter, and Michael A. Kozuch. Optimality analysis of energy-performance trade-off for server farm management. *Performance Evaluation*, 67(11):1155–1171, 2010.

[GHB11] Anshul Gandhi and Mor Harchol-Balter. How data center size impacts the effectiveness of dynamic power management. In *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pages 1164–1169. IEEE, 2011.

[GHBA10] Anshul Gandhi, Mor Harchol-Balter, and Ivo Adan. Server farms with setup costs. *Performance Evaluation*, 67(11):1123–1138, 2010.

[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.

[GLNT13] Rahul Ghosh, Francesco Longo, Vijay K. Naik, and Kishor S. Trivedi. Modeling and performance analysis of large scale IaaS clouds. *Future Generation Computer Systems*, 29(5):1216–1234, 2013.

[GMIL+00] Dirk Grunwald, Charles B. Morrey III, Philip Levis, Michael Neufeld, and Keith I. Farkas. Policies for dynamic clock scheduling. In *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation - Volume 4*, OSDI'00, pages 6–6, Berkeley, CA, USA, 2000. USENIX Association.

[GOA18] Pratik Gajane, Ronald Ortner, and Peter Auer. A sliding-window algorithm for Markov decision processes with arbitrarily changing rewards and transitions. *arXiv preprint arXiv:1805.10066*, 2018.

[Gra88] Winfried K. Grassmann. Finding the right number of servers in real-world queuing systems. *Interfaces*, 18(2):94–104, 1988.

[HH88] Sabine K. Hipp and Ulrich D. Holzbaur. Decision processes with monotone hysteretic policies. *Operations Research*, 36(4):585–588, 1988.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[How60] R. A. Howard. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA, 1960.

[IRCS13] Rocio Ilhuicatzi-Roldán and Hugo Cruz-Suárez. Rolling horizon procedures for the solution of an optimal replacement problem of n-machines with random horizon. *Investigación Operacional*, 34(2), 2013.

[Jan00] Jiri Jan. Digital signal filtering, analysis and restoration (telecommunications series), inspec, 2000.

[JWC12] Yichao Jin, Yonggang Wen, and Qinghua Chen. Energy efficiency and server virtualization in data centers: An empirical investigation. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 133–138. IEEE, 2012.

[KAR11] Jung Hyun Kim, Hyun-Soo Ahn, and Rhonda Righter. Managing queues with heterogeneous servers. *Journal of Applied probability*, 48(02):435–452, 2011.

[Ken73] M.G. Kendall. *Time-series*. Griffin, 1973.

[Koo11] Jonathan Koomey. Growth in data center electricity use 2005 to 2010. *A report by Analytical Press, completed at the request of The New York Times*, page 9, 2011.

[KR95] Mikhail Yu Kitaev and Vladimir V. Rykov. *Controlled queueing systems*. CRC press, 1995.

[Li12] Keqin Li. Optimal power allocation among multiple heterogeneous servers in a data center. *Sustainable Computing: Informatics and Systems*, 2(1):13–22, 2012.

[LKGN16] B. Leng, B. Krishnamachari, X. Guo, and Z. Niu. Optimal operation of a green server with bursty traffic. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2016.

[LL96] Onésimo Hernández Lerma and Jean Bernard Lasserre. *Discrete-time Markov control processes: basic optimality criteria*. Springer-Verlag, 1996.

[LS84] F. V. Lu and Richard F Serfozo. M/M/1 queueing decision processes with monotone hysteretic optimal policies. *Operations Research*, 32(5):1116–1132, 1984.

[LV02] Hsing Paul Luh and Ioannis Viniotis. Threshold control policies for heterogeneous server systems. *Mathematical Methods of Operations Research*, 55(1):121–142, 2002.

[LZ12] Young Choon Lee and Albert Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2):268–280, 2012.

[MD12] Michele Mazzucco and Dmytro Dyachuk. Optimizing cloud providers revenues via energy efficient server allocation. *Sustainable Computing: Informatics and Systems*, 2(1):1–12, 2012.

[MD15] Vincent J. Maccio and Douglas G. Down. On optimal control for energy-aware queueing systems. In *Teletraffic Congress (ITC 27), 2015 27th International*, pages 98–106. IEEE, 2015.

[MDD10] Michele Mazzucco, Dmytro Dyachuk, and Marios Dikaiakos. Profit-aware server allocation for green internet services. In *Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), 2010 IEEE International Symposium on*, pages 277–284. IEEE, 2010.

[Mit13] Isi Mitrani. Managing performance and power consumption in a server farm. *Annals OR*, 202(1):121–134, 2013.

[MO70] H. Mine and S. Osaki. *Markovian decision processes*. Modern analytic and computational methods in science and mathematics. American Elsevier Pub. Co., 1970.

[MS02] A. Muller and D. Stoyan. *Comparison Methods for Stochastic Models and Risks*. Wiley, New York, NY, 2002.

[Nel16] Patrick Nelson. Just one autonomous car will use 4,000 gb of data/-day. `https://www.networkworld.com/article/3147892/one-autonomous-car-will-use-4000-gb-of-dataday.html`, 2016.

[NIG07] Ripal Nathuji, Canturk Isci, and Eugene Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Autonomic Computing, 2007. ICAC'07. Fourth International Conference on*, pages 5–5. IEEE, 2007.

[NPK+05] Gethin Norman, David Parker, Marta Kwiatkowska, Sandeep Shukla, and Rajesh Gupta. Using probabilistic model checking for dynamic power management. *Formal aspects of computing*, 17(2):160–176, 2005.

[PBSB03] C. D. Patel, C. E. Bash, R. Sharma, and M. Beitelmal. Smart cooling of data centers. In *Proceedings of IPACK*, 2003.

[PD13] Karas Pavel and Svoboda David. Algorithms for efficient computation of convolution. In *Design and Architectures for Digital Signal Processing*. IntechOpen, 2013.

[Plu91] Hans-Joachim Plum. Optimal monotone hysteretic Markov policies in an M/M/1 queueing model with switching costs and finite time horizon. *Mathematical Methods of Operations Research*, 35(5):377–399, 1991.

[Put94] Martin L Puterman. *Markov Decision Processes. J.* Wiley and Sons, 1994.

[RDSJ08] Chheda Rajesh, Shookowsky Dan, Stefanovich Steve, and Toscano Joe. Profiling energy usage for efficient consumption. *The Architecture Journal*, 18:24, 2008.

[Ros70] S.M. Ross. *Applied probability models with optimization applications*. Holden-Day series in management science. Holden-Day, 1970.

[Ros14] Sheldon M. Ross. *Introduction to stochastic dynamic programming*. Academic press, 2014.

[RWH11] Charles Reiss, John Wilkes, and Joseph L. Hellerstein. Google cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, CA, USA, November 2011. Revised 2012.03.20.

[Ryk66] Vladimir V. Rykov. Markov decision processes with finite state and decision spaces. *Theory of Probability & Its Applications*, 11(2):302–311, 1966.

[Ser79] Richard F. Serfozo. Technical Note—An Equivalence Between Continuous and Discrete Time Markov Decision Processes. *Operations Research*, 27(3):616–620, 1979.

[Ser99] Bruno Sericola. Availability analysis of repairable computer systems and stationarity detection. *IEEE Trans. Computers*, 48(11):1166–1172, 1999.

[Sor15] Steve Sorrell. Reducing energy demand: A review of issues, challenges and approaches. *Renewable and Sustainable Energy Reviews*, 47:74–82, 2015.

[SPTG12] C. Schwartz, R. Pries, and P. Tran-Gia. A queuing analysis of an energy-saving mechanism in data centers. In *Information Networking (ICOIN), 2012 International Conference on*, pages 70–75, Feb 2012.

[STM08] Joris Slegers, Nigel Thomas, and Isi Mitrani. Dynamic server allocation for power and performance. In *SPEC International Performance Evaluation Workshop*, pages 247–261. Springer, 2008.

[Top78] Donald M. Topkis. Minimizing a submodular function on a lattice. *Operations research*, 26(2):305–321, 1978.

[Wil11] John Wilkes. More Google cluster data. Google research blog, November 2011. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`.

[WL11] Bing Wang and Dongpo Liu. Rolling horizon procedure for large-scale equipment replacement problems with multiple alternatives. In *2011 Chinese Control and Decision Conference (CCDC)*, pages 2741–2746. IEEE, 2011.

[XT08] Xiuli Xu and Naishuo Tian. The M/M/c queue with (e, d) setup time. *Journal of Systems Science and Complexity*, 21(3):446–455, 2008.

[YCNH11] Zexi Yang, Meng-Hsi Chen, Zhisheng Niu, and Dawei Huang. An optimal hysteretic control policy for energy saving in cloud computing. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE, 2011.

[ZW+14] Matt Zwolenski, Lee Weatherill, et al. The digital universe: Rich data and the increasing value of the internet of things. *Australian Journal of Telecommunications and the Digital Economy*, 2(3):47, 2014.

Appendix A

Schwartz et al. paper detail [SPTG12]

Schwartz et al. in [SPTG12] present a theoretical queuing model to evaluate the trade-off between waiting time and energy consumption if only a subset of servers is active all the time and the remaining servers are enabled on demand. In this paper the problem is modeled by a queuing model with an infinite buffer size $b = +\infty$. Authors assume that the jobs in their model arrive according to an independent Poisson process with rate λ and each server accepts only one job at a time with an exponentially distributed service time with mean $\frac{1}{\mu}$. Then, the system can be modeled using a $M/M/k$ queuing system. A number of m_0 servers called base-line-servers is active all the time, and a number of m_1 additional servers called reserve-servers is initially turned off and will be activated when the total number of jobs in the system increases and exceeds some threshold θ_1. The reserve-servers will be turned off again if the number of jobs in the system decreases and becomes less than another threshold θ_0. For a server, the authors consider three level of energy consumption:

1. c_{off} when the server is turned off,
2. c_{busy} when the server is turned on and proceeding a job, and
3. c_{idle} when the server is turned on without proceeding a job.

They also consider mainly two random variables. Random variable x gives the number of jobs in the system where only the base-line-servers are activated and in this case $x(i)$ is the stationary probability that i jobs are in the system. And the random variable y gives the number of jobs in the system where additionally the reserve-servers are activated and in this case $y(i)$ is the stationary probability that i jobs are in the system. Using CTMC, and based on local balance equations of

each state, they use analytical calculus to find the expression of each $x(i)$ and $y(i)$ in function of m_0, m_1, θ_0 and θ_1:

$$x(i) = \begin{cases} 1 - \sum_{k=1}^{m_0+\theta_1} x(k) - \sum_{k=\theta_0}^{+\infty} y(k) & \text{For } i = 0 \\ \frac{a^i}{i!} x(0) & \text{For } 0 < i < \theta_0 \\ x(m_0) \frac{a^{i-\theta_0+1}}{(i-\theta_0+1)!} - y(\theta_0) \frac{\theta_0 s_{i-\theta_0}}{i!} & \text{For } \theta_0 \leq i \leq m_0 \\ x(m_0) \frac{a^{i-\theta_0+1}}{m_0^{i-m_0}(m_0-\theta_0+1)!} - y(\theta_0) \frac{\theta_0 s_{m_0-\theta_0} a^{i-m_0}}{m_0!} & \\ + y(\theta_0) \frac{\theta_0 \left(1 - a^{i-m_0}\right)}{1-a} & \text{For } m_0 < i \leq m_0 + \theta_1 \end{cases} \tag{A.1}$$

$$y(i) = \begin{cases} \frac{a^{m_0+\theta_1+1}(\theta_0-1)! \left(a^{\theta_1} s_{n-\theta_0} + \frac{(1-a^{\theta_1})\theta_0}{1-a}\right)}{\left(1+\frac{a}{\theta_0}\right) m_0^{\theta_1} m_0! (m_0-\theta_0+1)!} & \text{For } i = \theta_0 \\ y(\theta_0) \frac{a^{i-\theta_0}\theta_0!}{i!} + x(m_0+\theta_1) \sum_{k=1}^{i-\theta_0} \frac{a^k (i-k)!}{i!} & \text{For } \theta_0 < i \leq m_0 + \theta_1 + 1 \\ y(m_0+\theta_1+1) \frac{a^{i-(m_0+\theta_1+1)} (m_0+\theta_1+1)!}{i!} & \text{For } m_0 + \theta_1 + 1 < i \leq m_0 + m_1 \\ y(m_0+m_1) \left(\frac{a}{m_0+m_1}\right)^{i-(m_0+m_1)} & \text{For } i > m_0 + m_1 \end{cases} \tag{A.2}$$

where $a = \frac{\lambda}{\mu}$, $\rho = \frac{a}{m_0+m_1}$, and $s_i = \sum_{k=0}^{i} a^k (m_0-k-1)!$.

Authors suggest finding the best m_0, m_1, θ_0 and θ_1 that minimize the following cost function:

$$cost = \alpha \mathbb{E}(e) + (1-\alpha)\mathbb{E}(w) \tag{A.3}$$

where:

$$\mathbb{E}(w) = \sum_{i=m_0}^{m_0+\theta_1} x(i) \frac{i-m_0}{\lambda} + \sum_{i=m_0+m_1}^{+\infty} y(i) \frac{i-m_0-m_1}{\lambda} \tag{A.4}$$

and

$$
\begin{aligned}
\mathbb{E}(e) &= \sum_{i=0}^{m_0} x(i)\left(ic_{busy} + (m_0 - i)\, c_{idle} + m_1 c_{off}\right) \\
&+ \sum_{i=m_0+1}^{m_0+\theta_0} x(i)\left(m_0 c_{busy} + m_1 c_{off}\right) \\
&+ \sum_{i=\theta_1}^{m_0+m_1} y(i)\left(ic_{busy} + (m_0 + m_1 - i)\, c_{idle}\right) \\
&+ \sum_{i=m_0+m_1+1}^{+\infty} y(i)\,(m_0 + m_1 + 1)\, c_{busy}
\end{aligned}
\tag{A.5}
$$

where $\mathbb{E}(e)$ is the expected value of energetic cost and $\mathbb{E}(w)$ is the expected value of waiting time. Parameter $\alpha \in [0,1]$ can be chosen in such a way that a desirable trade-off is made.

Du to the hardness to solve analytically the above optimization problem, authors suggest to solve it in a trivial way by computing the cost function for all valid combinations of m_0, m_1, θ_0 and θ_1, sorting the cost function values and choosing the minimum.

Finally, the experimental simulation results done in this paper, show that configurations as the one given in Table 2.1, leads to an energy consumption significantly reduced while still having an acceptable waiting time. They observe that the service was guaranteed while still saving about 40% of energy.

Table A.1: Experimental simulation settings in [SPTG12].

Parameter	**Value**	**Description**
$\mathcal{M} = m_0 + m_1$	100	Total number of servers
λ	0.10	Exponential inter-arrival job arrivals rate
μ	0.0025	Exponential service rate

Appendix B

Mitrani's paper detail [Mit13]

In [Mit13] Mitrani considers the problem of managing servers of a data center in order to satisfy the conflicting objectives between high QoS and low energy consumption. As in [SPTG12] Mitrani modeled the problem by a queuing model with an infinite buffer size $b = +\infty$. He assumes that the jobs arrive according to an independent Poisson process with rate λ and each server accepts only one job at a time with an exponentially distributed service time with mean $\frac{1}{\mu}$. So, the system can be modeled using a $M/M/k$ queuing system. In addition, $\mathcal{M}$ is the total number of servers, a subset of m_1 servers called *reserve* is turned on when the number of waiting jobs in the system is sufficiently high and exceeds a threshold U (for Up), and is turned off when that number of jobs is sufficiently low and exceeds another threshold D (for Down). Unlike Schwartz et al. [SPTG12], Mitrani takes into account the latency of the servers, which is the period of time needed by a server to be switched on. During this period a server consumes an energy surcharge to be powered up while no job is served. All the reserve servers become operational together after an interval of time distributed exponentially with mean $\frac{1}{\nu}$. The main objective of this paper is to find the best size of the reserve m_1 and the best values of thresholds U and D that allow a minimum total cost. The author considers one level of energy consumption of a server, which means that energy consumption depends on the number of switched on servers. He mainly considers three random variables:

- Variable x gives the number of jobs in the system where the reserve is off and in this case $x(i)$ is the stationary probability that i jobs are in the system.
- Variable y gives the number of jobs in the system during latency period (the reserve is powering up). In this case $y(i)$ is the stationary probability that i

jobs are in the system.

- Variable z gives the number of jobs in the system where additionally the reserve is powered on. In this case $z(i)$ is the stationary probability that i jobs are in the system.

Using CTMC, and based on local balance equations of each state, he uses analytical calculus (generating functions) to find the expression of each probability $x(i)$, $y(i)$ and $z(i)$ in function of m_1, U, and D where $0 < D < U < \mathcal{M}$. Probabilities $x(i)$, $y(i)$, and $z(i)$ allow to give expression of $\mathbb{E}(n)$ the average number of jobs in the system:

$$\mathbb{E}(n) = \sum_{i=0}^{U} ix(i) + \sum_{i=D+1}^{U} i(y(i)+z(i)) + g_1'(1) + g_2'(1) \tag{B.1}$$

where:

$$\begin{cases} g_1(1) & = \dfrac{x(U)+y(U)}{\zeta_2 - 1} \\ g_1'(1) & = g_1(1)\left(U+1+\dfrac{1}{\zeta_2-1}\right) \\ g_2(1) & = \dfrac{1}{\mathcal{M}\,\mu-\lambda}\left(\lambda z(U)+\dfrac{g_1(1)\nu\zeta_2}{\zeta_2-1}\right) \\ g_2'(1) & = (U+1)g_2(1)+\dfrac{1}{\mathcal{M}\,\mu-\lambda}\left(\lambda g_2(1)+\dfrac{g_1(1)\nu\zeta_2}{(\zeta_2-1)^2}\right) \\ \zeta_2 \in \mathbb{R} & : \text{The biggest zero of quadratic polynomial } \lambda\zeta(\zeta-1)+\mathcal{M}\,\mu(1-\zeta)-\nu\zeta \end{cases}$$

Additionally probabilities $x(i)$ allow to give expression of $\mathbb{E}(m)$ the average number of servers consuming energy in the system:

$$\mathbb{E}(m) = \mathcal{M} - m_0 \sum_{i=0}^{U} x(i). \tag{B.2}$$

Mitrani suggests an heuristics solution for the above equations in order to give optimal values of m_1, U, and D that minimize the following objective cost function:

$$cost = c_1\mathbb{E}(n) + c_2\mathbb{E}(m) \tag{B.3}$$

where $\mathbb{E}(n)$ represents the average number of jobs in the system, $\mathbb{E}(m)$ gives the average number of servers consuming energy in the system, coefficients c_1 and c_2 reflect the relative importance placed on QoS and energy consumption, respectively. He obtained a heuristic value of m_1 (size of the reserve) by arguing as follows. Let $n(m_0)$ be the average number of jobs in a system where there are m_0 permanently operative servers. Consider the change in costs as the number

of servers is increased from $m_0 - 1$ to m_0: the average queuing cost per unit time would decrease by $c_1\,(n(m_0-1)-n(m_0))$. At the same time, the energy consumption cost would increase by c_2 . Hence, the increase would be advantageous if $c_1\,(n(m_0-1)-n(m_0)) > c_2$. Consequently, the optimal number of permanently operative servers would be the largest m_0 for which an increase from $m_0 - 1$ to m_0 is advantageous. Mitrani proposed an heuristic that consists in approximating $n(m_0)$ by the $M/M/1$ expression: $n(m_0) = \dfrac{\frac{\lambda}{\mu}}{\mathcal{M} - \frac{\lambda}{\mu}}$, an optimal value of m_0 should be equal to:

$$m_0^* = \left\lfloor \frac{\lambda}{\mu} + \frac{1}{2}\left(1 + \sqrt{1 + 4\frac{\lambda c_1}{\mu c_2}}\right)\right\rfloor. \tag{B.4}$$

In fact the optimal size of the reserve is: $m_1^* = \max\{0, \mathcal{M} - m_0^*\}$.

Consider that we turn off the reserve when some of the non-reserve servers become idle. In this case, the heuristic value of the lower threshold D can be fixed to:

$$D^* = \mathcal{M} - m_0 - 1. \tag{B.5}$$

To derive a heuristic for the upper threshold, Mitrani uses deterministic fluid approximation of the queuing process and he deduces:

$$U^* = \max\left\{m_0 - 1, \left\lceil \frac{m_0\mu - \lambda}{\nu} + \frac{c_2(m_0\mu - \lambda)}{c_1\mu}\left(1 + \sqrt{1 + \frac{2c_1(\mathcal{M}\,\mu - \lambda)}{c_2\nu(m_0\mu - \lambda)}}\right)\right\rceil\right\}. \tag{B.6}$$

Over parameters shown in Table B.1, Mitrani carried out several numerical experiments to evaluate the quality of his heuristics.

Table B.1: Experimental simulation settings in [Mit13].

Parameter	**Value**	**Description**
$\mathcal{M} = m_0 + m_1$	20	Total number of servers
λ	4, 8, 10, 12	Exponential inter-arrival job arrivals rate
μ	1	Exponential service rate
ν	0.1	Exponential latency rate

He observed that when the size of reserve m_1 is well chosen, it is not important if the upper threshold U is overestimated. On the other hand, if m_1 is badly chosen, then both underestimating and overestimating the optimal threshold U can increase the cost substantially. And as expected, the heavier the offered load, the fewer servers should be reserved. Finally he compares the performance of the optimal

policy (m_0, U, and D are chosen optimally), with that of the heuristic policy (using the heuristic values for m_0, U, and D), and also with the *do nothing policy* of not reserving any servers. His experimental results confirm that the heuristic policy is practically indistinguishable from the optimal policy, with only a small difference. This has been observed to be the case for a large variety of system configurations and cost coefficients. As future work, Mitrani suggests to extend his model by introducing k blocks of reserves, of sizes $m_1, m_2, \ldots, m_k$, with associated upper and lower thresholds. Reserve 1 is turned on when the queue exceeds level U_1, reserve 2 is turned on when the queue exceeds level $U_2 > U_1$, etc.

Appendix C

Including lost into the state of the system

This section is an additional experimental analysis for the homogeneous model of Chapter 4. We analyze the impact of considering an MDP model in two way:

1. each state of the MDP is modeled only by the number of operational servers and the number of waiting jobs in the buffer (m,n),
2. each state of the MDP is modeled by the number of operational servers and the number of waiting jobs in the buffer, and additionally by the number of rejected jobs $(m,(n,l))$.

As shown previously in Theorems 4.1 and 4.2, Figures C.1 and C.2 show that the size (size can be seen as the number of states or the number of transitions) of the MDP is more important when we consider the second model. Nevertheless, the spatial complexity of both model still important even when lost l is not considered.

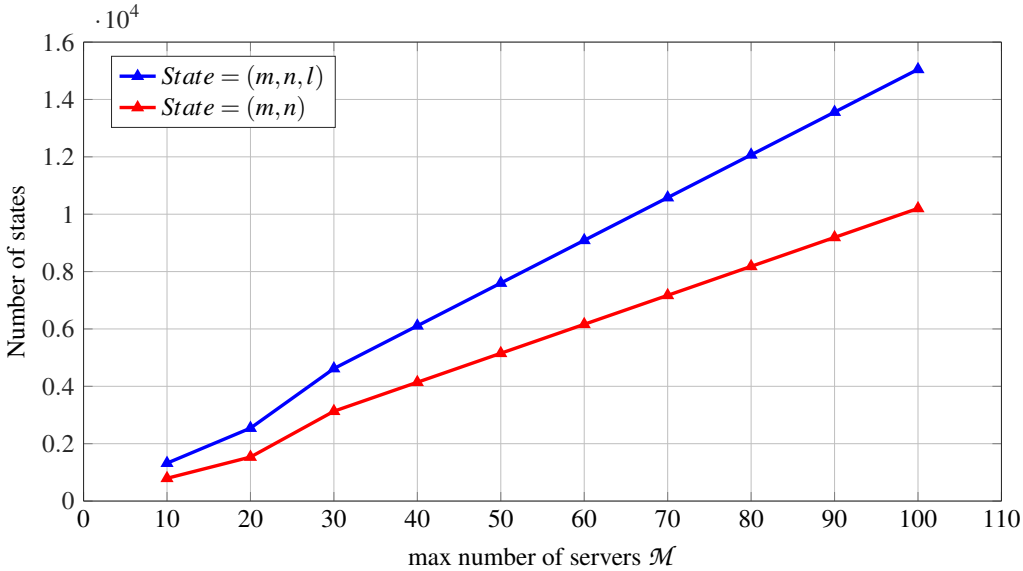


Figure C.1: MDP state number when considering or not the lost jobs ($b = 100$).

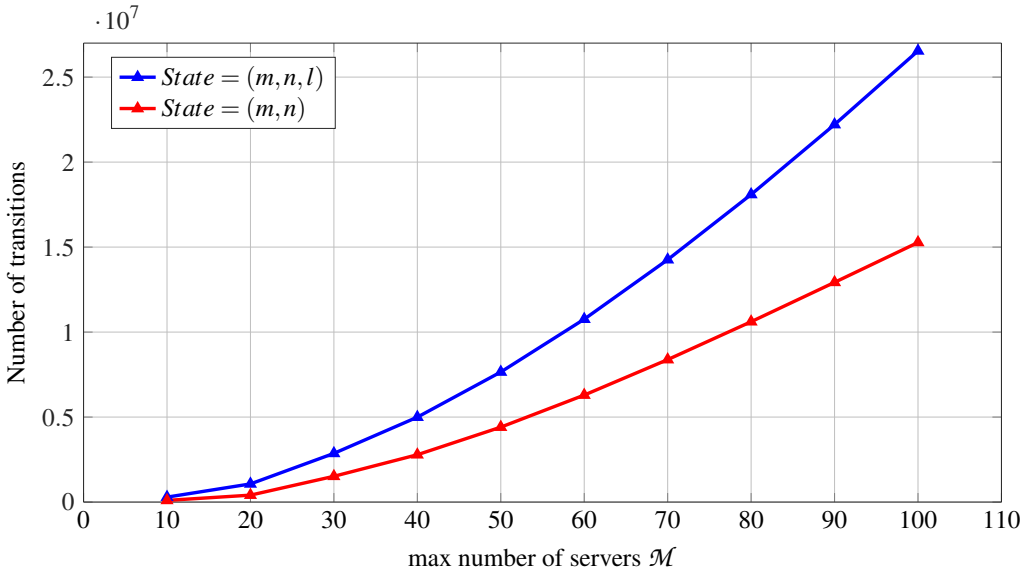


Figure C.2: MDP transition number when considering or not the lost jobs ($b = 100$).

Appendix D

MDP for variable arrival distribution

Our analysis of the Google trace arrival jobs [Wil11, RWH11] shows that arrival traffic changes during time and presents a non negligible level of fluctuation daily along the month (see Figure 1.7 of Chapter 1), and hourly as shown in Figure 4.28 which present three samples of the arrival traffic for three different hours. For example the traffic between 6h and 7h for 27 May 2011 (blues curve), and the traffic between 10h and 11h for 19 May 2011 (red curve), are close and may be modeled by the same histogram. However it is clear that the traffic between 14h and 15h for 18 May 2011 (orange curve), must be modeled by another histogram as the traffic during this period is significantly higher than the two first one. Figure 4.29 shows that even from a minute to another minute the traffic my change significantly. In fact, in contrary of the previous models of MDP where we consider that arrivals are modeled by the same histogram over time. In this section, we will consider that each slot we may receive a number of arrival jobs modeled by a different histogram.

D.1 Problem specification

Let DC be a data center composed of $\mathcal{M}$ identical servers. DC receives jobs requesting the offered service. We assume that the arrival process may change every slot. This allows us to model for instance hourly or daily variations of the job arrivals, even generally a variation each slot of time. Suppose that the analysis will be done over h a whole period of time. Thus, the number of jobs arriving to the data center per slot during time slot t is modeled by a histogram $\mathcal{H}_{A(t)}$ where $P_{A(t)}(i)$ gives the probability to have i arrival jobs during slot t. Note that we assume that during a slot arrivals of jobs are independent, and their distribution is obtained from

real traces, empirical data, or incoming traffic measurements. In order to focus more on the MDP modelization for the case of a job arrivals process modeled by a variable histogram over time, we assume some restrictions that were considered in the previous MDP model:

1. The maximal number of jobs that can be served by one server in one slot is modeled by histogram $\mathcal{H}_D$.
2. A server needs no time to be switched on or off, the switching-on leads instantaneously to an additional energetic cost c_{on} however the switching-off takes place without consuming any additional amount of energy.
3. For the QoS metric, the rejected jobs are not considered, only the waiting jobs will be considered.

In this manner, the queuing model is a batch arrival queue with constant services and finite capacity buffer b (buffer size). The number of waiting jobs in the buffer is denoted by n. The number of operational servers is denoted by m. We assume that initially the number of operational servers, the number of waiting jobs. The maximal number of servers that can be operational is $\mathcal{M}$. As shown before the number of waiting jobs n can be computed by induction where the following equations give the number of waiting jobs in the buffer. For a number of i arrival jobs, and if each server processes at most d jobs per slot, we will have:

$$n \leftarrow \min\{b, \max\{0, n+i-d\times m\}\}. \tag{D.1}$$

D.1.1 Energy and Performance metric

The energy consumption takes into account the number of operational servers. Each server consumes some units of energy per slot when a server is operational and it costs an average of $c_M \in \mathbb{R}^+$ monetary unit. During the latency period a server may consume an additional amount of energy that costs an average of $c_{On} \in \mathbb{R}^+$ monetary unit which is the energetic cost needed to switch-on a server. Additionally, we consider that a server switches-on immediately. The total consumed energy is the sum of all units of energy consumed among a specific period. QoS takes into account the number of waiting. Each waiting job costs $c_N \in \mathbb{R}^+$ monetary unit per slot.

D.1.2 Objective function

For a given period of time, a dynamic power management system consists in doing each slot an action (turn on or turn-off a specific number of servers) in order to adapt

the number of operational servers to incoming job changes. The optimal strategy consists in finding the best sequence of actions to minimize the accumulated overall cost (energetic cost plus performance cost). The cost generated for each slot is a linear combination of energetic and QoS costs called objective function:

$$n \times c_N + m \times c_M + \max\{0, m' - m\} \times c_{On} \tag{D.2}$$

where n is the number of waiting jobs, m is the current number of operational servers, and m' is the number of operational servers after doing an action.

The problem we have to consider is to find a trade-off between the performance and the energy consumption. As the number of servers changes with time, the system becomes more complex to analyze than a system with servers operational all the time.

In order to find the optimal strategy and then analyze the performance and the energy consumption of the data center under this optimal strategy, we will continue to use the concept of *Markov Decision Process* to formulate our optimization problem.

D.2 Optimization

In order to model the above specification we may consider two kind of solution. The second one is more adapted to be implemented in a model checker in-which the value iteration algorithm is predefined and encoded in the kernel of the model checker as in PRISM or MdpToolBox (see Annex D for more details). The first one is an adaptation of the generic value iteration algorithm for the case of variable distribution arrival jobs. In fact for the first solution we consider the possibility to write a version of value iteration algorithm which is specific for this solution. Thus we adapt the computation of the value function V to take into account the appropriate arrival job histogram for each slot t. Notice that both approaches give rise to the same number of transitions (see Table D.1).

Table D.1: Approaches comparison.

	States	**Transitions**	**Advantage**
First approach	$O(\mathcal{M} \times b)$	$O(\mathcal{M}^2 \times b \times h \times \lvert S_A \rvert)$	Less states
Second approach	$O(\mathcal{M} \times b \times h)$	$O(\mathcal{M}^2 \times b \times h \times \lvert S_A \rvert)$	Use of PRISM

D.3 First approach

In this section we give a modelization that can not be written in a model checker in-which the value iteration algorithm is predefined as in PRISM or MdpToolBox, because it is based on a modified version of the generic value iteration algorithm that is able to take into account the fact that each slot we may have a different distribution for the arrival jobs process.

D.3.1 Modelization

Let $\mathcal{H}_{A(t)} = (S_{A(t)}, P_{A(t)})$ be the histogram used to model the arrival of jobs during slot t. Assume that the optimization will be done over a period of h slot (the horizon). Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ be an MDP where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P} = \{\mathcal{P}_t \mid t \in [0..h]\}$ is a set of transition probability matrices where $\mathcal{P}_t$ is the transition probability matrix during slot t, and finally $\mathcal{C}$ is the immediate cost of each action. The state of the system is defined by the couple (m,n) where m is the number of operational servers, n is the number of waiting jobs. However t the age of the system, will be associated to the probability function $\mathcal{P}$ (remark that we need to add t into $\mathcal{P}$ in order to be able to catch the changes in the distribution of arrival jobs $\mathcal{H}_{A(t)}$). Indeed the state space $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{(m,n) \mid m \in [0..\mathcal{M}] \text{ and } n \in [0..b]\}. \tag{D.3}$$

At the beginning of each slot, and based on the current state of the system, an action $\alpha_j \in \mathcal{A}$ will be made to determine how many servers will be operational during the current slot. In fact the action space $\mathcal{A}$ is defined as $\mathcal{A} = \{\alpha_j \mid 0 \leq j \leq \mathcal{M}\}$, where action α_j consists in keeping exactly j operational servers during the current slot. We have a probability of $\mathcal{P}^{\alpha_j}_{t,ss'}$ to move from state $s = (m,n)$ to $s' = (j,n'))$ under action α_j, during slot t. This probability is defined as:

$$\mathcal{P}^{\alpha_j}_{t,ss'} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_{A(t)} \text{ and each } d \in S_D \text{ satisfying:} \\ n' & = \min\{b, \max\{0, n+i-d\times j\}\} \end{array}\right.} P_{A(t)}(i) \times P_D(d) \tag{D.4}$$

Consequently moving from state $s = (m,n)$ to $s' = (j,n')$ under action α_j induces immediately a cost $\mathcal{C}_s^{\alpha_j}$ defined as:

$$\mathcal{C}_s^{\alpha_j} = j \times c_M + \max\{0, j-m\} \times c_{On} + n \times c_N. \tag{D.5}$$

Algorithm 12 is an adaptation of the generic value iteration algorithm to include the fact that arrival jobs distribution may change over time.

Algorithm 12: Value iteration algorithm for MDP.

```
Data: 𝓜, 𝓒, 𝓢, horizon h                                   1
Data: 𝓟_0, 𝓟_1, ···, 𝓟_t, ···, 𝓟_h                          2
Result: (V_h, π*_h)                                         3
foreach s ∈ 𝓢 do                                            4
    V_0(s) = 0                                              5
end                                                         6
for k ← 1 to h do                                           7
    foreach s ∈ 𝓢 do                                        8
        (m, n) ← s                                          9
        for a_j ← a_0 to a_𝓜 do                             10
            C_s^{α_j} ← j × c_M + max{0, j − m} × c_On + n × c_N   11
            Q_k(s, a_j) ← C_s^{α_j} + Σ_{s'} 𝓟^{α_j}_{h−k,ss'} V_{k−1}(s')   12
        end                                                 13
        π*_k(s) ← argmin_a Q_k(s, a)                        14
        V_k(s) ← Q_k(s, π*_k(s))                            15
    end                                                     16
end                                                         17
return (V_h, π*_h)                                          18
```

D.3.2 Space and time complexities

In this section we will evaluate the space complexity of the data structure behind our MDP formulation.

Theorem D.1. *The number of states of the MDP is in* $O(\mathcal{M} \times b)$.

Proof. Every state of the MDP includes two element:

1. the number of operational servers which is between 0 and $\mathcal{M}$,
2. the number of waiting jobs in the buffer which is bounded above by b, and

So, $|\mathcal{S}|$ is bounded above by $(\mathcal{M}+1) \times (b+1)$. □

Theorem D.2. *The number of transitions of the MDP is in* $O(\mathcal{M}^2 \times b \times h \times |S_A|)$.

Proof. From each state of the MDP we have at most $(\mathcal{M}+1)$ actions, and each action leads to a number of transitions equals to $|S_A|$ (one transition for each bin in the support of the arrival distribution). However as we need to consider the

set of $(h+1)$ matrices of probability transition the total number of transitions is $|\mathcal{S}| \times |S_A| \times (h+1)$. In fact, as the number of states was already evaluated in Theorem D.1, we deduce that the number of transitions is bounded above by $\mathcal{M} \times b \times (\mathcal{M}+1) \times |S_A| \times h$ which is in $O(\mathcal{M}^2 \times b \times h \times |S_A|)$. □

Theorem D.3. *The computation-time to find the optimal strategy with the value iteration algorithm is in* $O(h \times b^2 \times \mathcal{M}^3)$.

Proof. As we know, at each iteration, for each of $|\mathcal{S}|$ states, the value iteration algorithm, which is polynomial in number of states, computes expectation for $|\mathcal{A}|$ actions. In general, each expectation takes $O(|\mathcal{S}|)$ time, however in this modelization each expectation takes only $O(\mathcal{M} \times b)$ time. In fact the total time complexity is $O(h \times |\mathcal{S}| \times |\mathcal{A}| \times (\mathcal{M} \times b))$ which equals $O(h \times b^2 \times \mathcal{M}^3)$ (see proof of Theorem 4.3 for more details). □

In fact, even if the number of servers in the data center is low, if we want an optimization over an important period of time (a big horizon h), the MDP structure will be very huge and parsing it to find the optimal solution becomes very important.

D.3.3 Example

To illustrate our second formalization Let us show an MDP for a very simple data center:

1. One server and buffer size equals one which means that $\mathcal{M} = 1$ and $b = 2$.
2. We set service rate of each machine as $\mathcal{H}_D = \Delta_1$.
3. For a period of 3 slots, job arrivals are modeled as the following:
   (a) During $t = 0$ by histogram $\mathcal{H}_{A(t)} = \Delta_1$.
   (b) During $t = 1$ by histogram $\mathcal{H}_{A(t)} = \Delta_1$.
   (c) During $t = 2$ by histogram $\mathcal{H}_{A(t)} = (S_A, P_A)$ where $S_A = \{0,1\}$, $P_A(0) = 0.1$ and $P_A(1) = 0.9$.

So, MDP$= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ where $\mathcal{A} = \{\alpha_0, \alpha_1\}$, $\mathcal{C}$ can be deduced from Relations D.4 and D.5, and $\mathcal{P}$ from Figures D.3, D.1, D.2. And space of states defined as:

$$\mathcal{S} = \{(0,0),(0,1),(0,2),(1,0),(1,1),(1,2)\}.$$

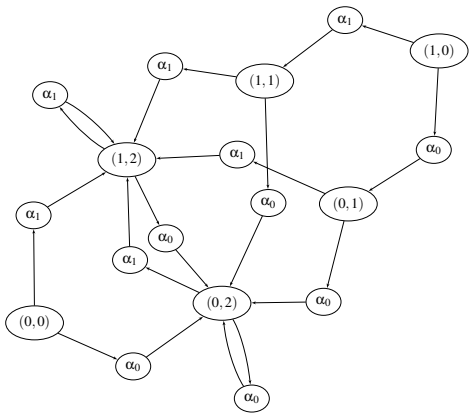


Figure D.1: Transition probability matrix for the second slot $\mathcal{P}_{t=1}$.

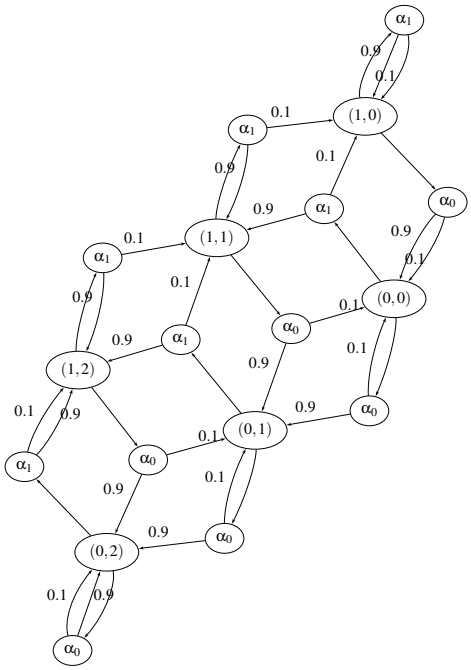


Figure D.2: Transition probability matrix for the second slot $\mathcal{P}t = 2$.

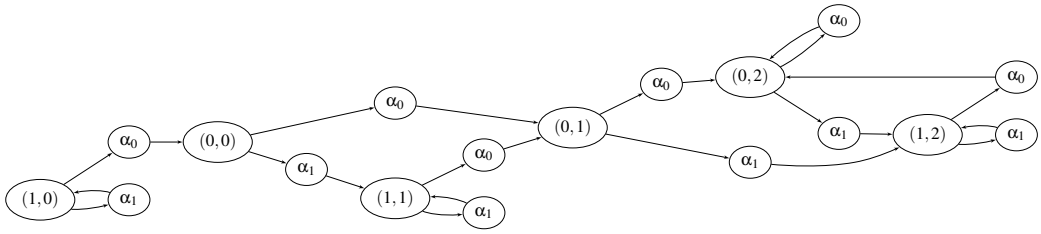


Figure D.3: Transition probability matrix for the first slot $\mathcal{P}_{t=0}$.

D.4 Second approach

In this section we give a modelization that can be written in any model checker in-which the value iteration algorithm is predefined as in PRISM or MdpToolBox.

D.4.1 Modelization

Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ be an MDP where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the set of actions, $\mathcal{P}$ is the transition probability, and $\mathcal{C}$ the immediate cost of each action. Let

$\mathcal{H}_{A(t)} = (S_{A(t)}, P_{A(t)})$ be the histogram used to model the arrival of jobs during slot t. Assume that the optimization will be done over a period of h slot (the horizon). The state of the system is defined by the tuple (m,n,t) where m is the number of operational servers, n is the number of waiting jobs, and t the age of the system (remark that we need to add t into the state of the system in order to be able to catch the changes in the distribution of arrival jobs $\mathcal{H}_{A(t)}$). Indeed the state space $\mathcal{S}$ is defined as:

$$\mathcal{S} = \{(m,n,t) \mid m \in [0..\mathcal{M}] \text{ and } n \in [0..b] \text{ and } t \in [0..h]\}. \tag{D.6}$$

At the beginning of each slot, and based on the current state of the system, an action $\alpha_j \in \mathcal{A}$ will be made to determine how many servers will be operational during the current slot. In fact the action space $\mathcal{A}$ is defined as $\mathcal{A} = \{\alpha_j \mid 0 \leq j \leq \mathcal{M}\}$, where action α_j consists in keeping exactly j operational servers during the current slot. We have a probability of $\mathcal{P}_{ss'}^{\alpha_j}$ to move from state $s = (m,n,t)$ to $s' = (j,n',t'))$ under action α_j. This probability is defined as:

$$\mathcal{P}_{ss'}^{\alpha_j} = \sum_{\left\{\begin{array}{ll} \text{for} & \text{each } i \in S_A : \\ n' & = \min\{b, \max\{0, n+i-d\times j\}\} \\ t' & = t+1 \end{array}\right.} P_A(i). \tag{D.7}$$

Consequently moving from state $s = (m,n,t)$ to $s' = (j,n',t')$ under action α_j induces immediately a cost $\mathcal{C}_s^{\alpha_j}$ defined as:

$$\mathcal{C}_s^{\alpha_j} = j \times c_M + \max\{0, j-m\} \times c_{On} + n \times c_N. \tag{D.8}$$

The immediate cost $\mathcal{C}_s^{\alpha_j}$, incorporate three parts:

1. The first part is $j \times c_M$, where c_M is the cost of energy consumption of one working server per slot and j is the number of working servers during the current slot. This part presents the total cost of energy consumed by the operational servers during the current slot.

2. The second part is $\max\{0, j-m\} \times c_{On}$, where c_{On} is the energetic cost of switching-on one server from stopping mode to working mode and $\max\{0, j-m\}$ is the number of servers switched-on at the beginning of the slot. This part presents the total cost of energy used to switch-on servers at the beginning of the current slot.

3. The third part is $n \times c_N$, where c_N is the cost of keeping one job in the buffer during the current slot and n is the number of waiting jobs. This part presents the total cost of maintaining waiting jobs in the buffer during the current slot.

Table D.2 summarizes parameters used in our model and our MDP formulation.

Table D.2: Model and MDP Parameters.

Parameters	**Description**
h	duration of analysis
$\mathcal{M}$	total number of servers
b	buffer size
m	number of operational servers
n	number of waiting jobs
$\mathcal{H}_{A(t)}$	job arrivals histogram for slot t
Δ_d	histogram of service jobs rate
c_{On}	energetic cost of switching-on one server from stopping mode to working mode
c_M	energetic cost of one working server during one slot
c_N	cost of one waiting job in the buffer during one slot
$\mathcal{S}$	set of all possible states
$\mathcal{A}$	set of all possible actions
$s = (m,n,t)$	system state
$s_0 = (0,0,0)$	starting state
α_j	action to keep exactly j operational servers
$\mathcal{P}_{ss'}^{\alpha_j}$	probability of transition from s to s' under action α_j
$\mathcal{C}_s^{\alpha_j}$	immediate cost from s under action α_j

D.4.2 Space and time complexities

In this section we will evaluate the space complexity of the data structure behind our MDP formulation.

Theorem D.4. *The number of states of the MDP is in* $O(\mathcal{M} \times b \times h)$.

Proof. Every state of the MDP includes three element:

1. the number of operational servers which is between 0 and $\mathcal{M}$,
2. the number of waiting jobs in the buffer which is bounded above by b, and
3. the age of the system which can be at most equals to the horizon h.

So, $|\mathcal{S}|$ is bounded above by $(\mathcal{M}+1) \times (b+1) \times (h+1)$. □

Theorem D.5. *The number of transitions of the MDP is in* $O(\mathcal{M}^2 \times b \times h \times |S_A|)$.

Proof. From each state of the MDP we have at most $(\mathcal{M}+1)$ actions, and each action leads to a number of transitions equals to $|S_A|$ (one transition for each bin in the support of the arrival distribution). In fact, as the number of states was already evaluated in Theorem D.4, we deduce that the number of transitions is bounded above by $\mathcal{M} \times b \times h \times (\mathcal{M}+1) \times |S_A|$ which is in $O(\mathcal{M}^2 \times b \times h \times |S_A|)$. □

Theorem D.6. *The computation-time to find the optimal strategy with the value iteration algorithm is in* $O(h^2 \times b^2 \times \mathcal{M}^3)$.

Proof. As we know, at each iteration, for each of $|\mathcal{S}|$ states, the value iteration algorithm, which is polynomial in number of states, computes expectation for $|\mathcal{A}|$ actions. In general, each expectation takes $O(|\mathcal{S}|)$ time, however in this modelization, an action can only moves the system from a state (m,n,t) to any state $(m',n',t+1)$, which means that each expectation takes only $O(\mathcal{M} \times b)$ time. In fact the total time complexity is $O(h \times |\mathcal{S}| \times |\mathcal{A}| \times (\mathcal{M} \times b))$ which equals $O(h^2 \times b^2 \times \mathcal{M}^3)$ (see proof of Theorem 4.3 for more details). □

In fact, even if the number of servers in the data center is low, if we want an optimization over an important period of time (a big horizon h), the MDP structure will be huge and parsing it to find the optimal solution becomes important.

D.4.3 Example

To illustrate our formalization Let us show an MDP for a very simple data center:

1. One server and buffer size equals one which means that $\mathcal{M} = 1$ and $b = 2$.
2. We set service rate of each machine as $d = 1$.
3. For a period of $h = 8$ slots, job arrivals are modeled as the following:
   (a) During $t = 0,1,2$ by histogram $\mathcal{H}_{A(t)} = (S_A, P_A)$ where $S_A = \{0,1\}$, $P_A(0) = P_A(1) = \frac{1}{2}$.
   (b) During $t = 3,4,5$ by histogram $\mathcal{H}_{A(t)} = (S_A, P_A)$ where $S_A = \{0,1,2\}$, $P_A(0) = P_A(1) = P_A(2) = \frac{1}{3}$.
   (c) During $t = 6,7$ by histogram $\mathcal{H}_{A(t)} = (S_A, P_A)$ where $S_A = \{0,1,2,3\}$, $P_A(0) = P_A(1) = P_A(2) = P_A(3) = \frac{1}{4}$.

So, MDP$= (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{C})$ where $\mathcal{A} = \{\alpha_0, \alpha_1\}$, $\mathcal{P}$ and $\mathcal{C}$ can be deduced from Relations D.7 and D.8. And space of states defined as:

$$\mathcal{S} = \{(0,0,t),(0,1,t),(0,2,t),(1,0,t),(1,1,t),(1,2,t) \mid t = 0,\ldots,8\}.$$

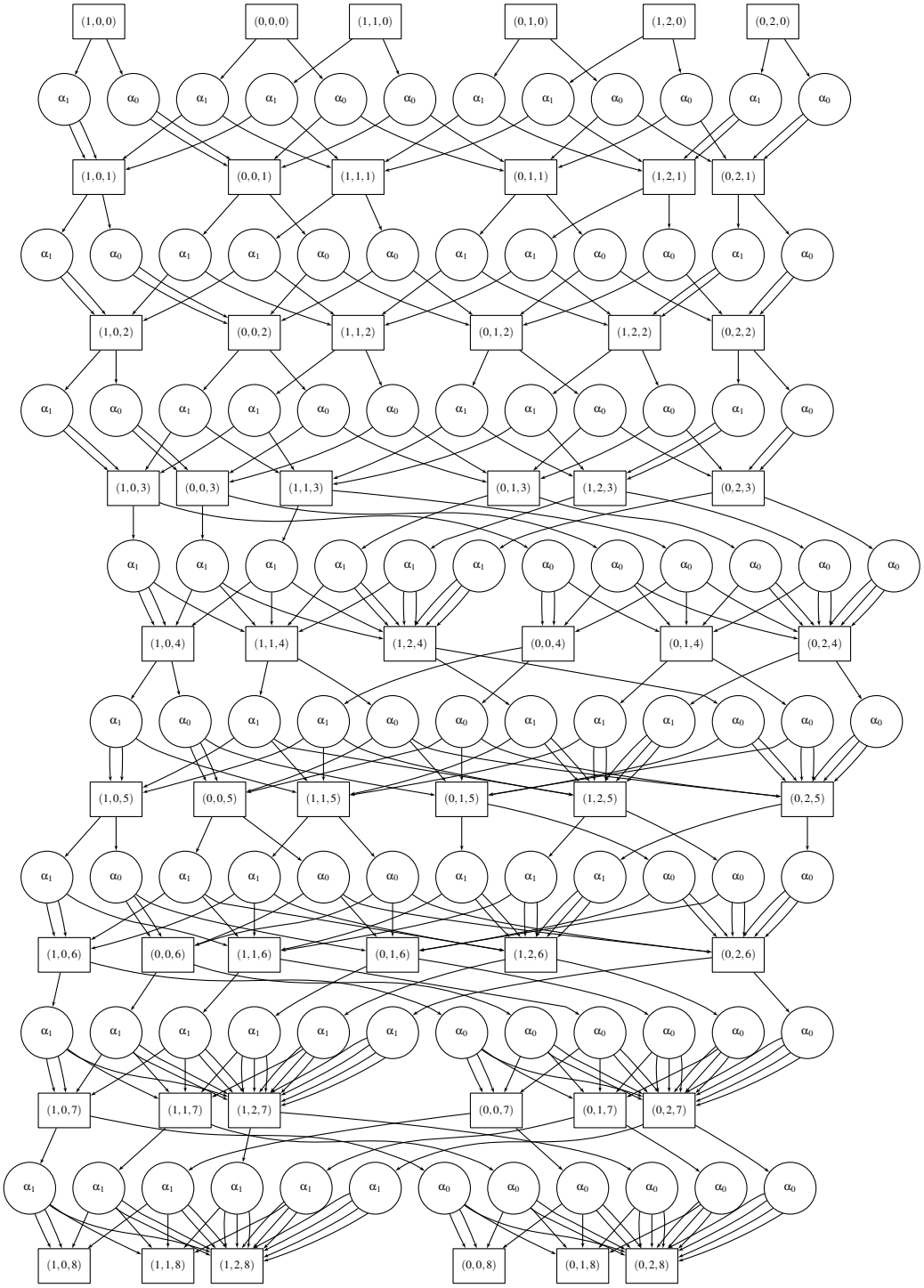


Figure D.4: MDP example for arrival distribution that change over time.

Appendix E

Additional material

E.1 Energy

What is energy? Each of us experiences to feel like to have energy or to be energetic. Thus, energy is this thing that allows us to walk, to act, to be active, to go work... However, we can lose energy in the evening when we get back from work. In fact we imagine energy as a volatile entity or a mystical substance. But that intuitive sense has inspired us to discover the most powerful and useful concept in physics. According to *Oxford English Dictionary* energy, in physics, is the property of matter and radiation which is manifest as a capacity to perform work (such as causing motion or the interaction of molecules). According to *Cambridge English Dictionary* energy is the power from something such as electricity or oil that can do work, such as providing light and heat.

In physics, energy is not a substance. Energy, is a number, a quantity. And the concept of energy began taking it rigorous mathematical shape during the 17th century initially by Gottfried Leibniz and Isaac Newton. However, in simple words, we can say that energy is the ability to do work, all kinds of work. And work is just the act of displacing something by applying force.

For example, when you enjoy your preferable meal, the energy from that food is transported into the trillions cells of your body in order to accomplish their several physiological and biological activities as making muscle cells contract, transporting materials from one organ to another, making copies of DNA, assembling proteins, and all the rest of work of a living being.

However there is a very important thing to remember: once the work is done, the energy is not done. Because energy never goes away. It can never be destroyed, and in the same way, it can never be created. It can only be transferred from one

source to another, like, how the energy in the food that were in the meal were transferred into the body, or it can be transformed from one form into another like the chemical energy in the wood being transformed to light and heat as fire. In fact when using expression as *energy production* or *energy consumption* we need to keep in mind that energy can neither be produced nor consumed. Energy consumption is only a convention expression to mention the transformation energy from one form to another form when doing specific work.

To fully introduce energy, it is important to understand how it is measured. As said before energy is the capacity to do work and is equal to force $\times$ distance, in other words it is the force required over a distance. Work is measured in *joules* and therefore this is one of the main units for measuring energy. Joule is a metric unit equal to a *newton-meter* or the amount of work done when a force of one newton is moved by one meter. Work may also be measured in thermal units and *calorie* is the amount of energy needed to raise the temperature of one gram of water by one degree Celsius at normal atmospheric pressure. Another measure of energy that we may have heard about is a BOE, *a barrel of oil equivalent*, which is the energy contained in one barrel of oil.

Power and energy are not the same thing. Unfortunately those terms are sometimes used interchangeably. Power is the rate at which energy is consumed or produced and is equal to energy over time. In the metric system power is measured in *watts* which is equal to one joule per second. Not confused with a *watt hour* which is the energy consumed by a one watt device operating for one hour. Thus watt hours represent a quantity of energy which is the unit we see on our electricity bills.

Table E.1: Units of energy

Unit	**Symbol**	**Joule equivalent**
electronvolt	eV	$1.6021766208 \times 10^{-19}$ J
calorie	cal	4.184 J
kilowatt-hour	kWh	3.6×10^{6} J
tonne of TNT	tonne of TNT	4.184×10^{9} J
barrel of oil equivalent	BOE	6.1178632×10^{9} J

Table E.2: List of some of the most important symbols and notations of the manuscript

Symbol	**Meaning**
$\mathbb{N}$	Set of natural numbers
$\mathbb{R}$	Set of real numbers
$x, y, z, \ldots$	Discrete random variables
$X, Y, Z, \ldots$	Histograms
Δ_v	Dirac function histogram
QoS	Quality of the Service
MDP	Markov decision process
$\mathcal{M}$	Total number of server
b	Buffer size
n	Number of waiting jobs
m	Number of operational servers
U	Up threshold
D	Down threshold
c_M	Energetic cost for running one operational server during one slot
c_N	Waiting cost for one job over one slot
c_L	Rejection cost for one job over one slot
SRN	Stochastic Reward Networks
SAN	Stochastic Activity Networks

Résumé

Pour garantir à la fois une bonne performance des services offerts par des centres de données, et une consommation énergétique raisonnable, une analyse détaillée du comportement de ces systèmes est essentielle pour la conception d'algorithmes d'optimisation efficaces permettant de réduire la consommation énergétique. Cette thèse s'inscrit dans ce contexte, et notre travail principal consiste à concevoir des systèmes de gestion dynamique de l'énergie basés sur des modèles stochastiques de files d'attente contrôlées. Le but est de rechercher les politiques de contrôle optimales afin de les appliquer sur des centres de données, ce qui devrait répondre aux demandes croissantes de réduction de la consommation énergétique et de la pollution numérique tout en préservant la qualité de service. Nous nous sommes intéressés d'abord à la modélisation de la gestion dynamique de l'énergie par un modèle stochastique pour un centre de données homogène, principalement pour étudier certaines propriétés structurelles de la stratégie optimale, telle que la monotonie. Après, comme des centres de données présentent un niveau non négligeable d'hétérogénéité de serveurs en termes de consommation d'énergie et de taux de service, nous avons généralisé le modèle homogène à un modèle hétérogène. De plus, comme le réveil (resp. l'arrêt) d'un serveur de centre de données n'est pas instantané et nécessite un peu plus de temps pour passer du mode veille au mode prêt à fonctionner, nous avons étendu le modèle dans le but d'inclure cette latence temporelle des serveurs. Tout au long de cette optimisation exacte, les arrivées et les taux de service sont spécifiés avec des histogrammes pouvant être obtenus à partir de traces réelles, de données empiriques ou de mesures de trafic entrant. Nous avons montré que la taille du modèle MDP est très grande et conduit au problème de l'explosion d'espace d'états et à un temps de calcul important. Ainsi, nous avons montré que l'optimisation optimale nécessitant le passage par un MDP est souvent difficile, voire pratiquement impossible pour les grands centres de données. Surtout si nous prenons en compte des aspects réels tels que l'hétérogénéité ou la latence des serveurs. Alors, nous avons suggéré ce que nous appelons l'algorithme greedy-window qui permet de trouver une stratégie sous-optimale meilleure que celle produite lorsqu'on envisage un mécanisme spécial comme les approches à seuil. Et plus important encore, contrairement à l'approche MDP, cet algorithme n'exige pas la construction complète de la structure qui encode toutes les stratégies possibles. Ainsi, cet algorithme donne une stratégie très proche de la stratégie optimale avec des complexités spatio-temporelles faibles. Cela rend cette solution pratique et permet son implémentation dans un contexte à temps réel.

Abstract

To ensure both good data center service performance and reasonable power consumption, a detailed analysis of the behavior of these systems is essential for the design of efficient optimization algorithms to reduce energy consumption. This thesis fits into this context, and our main work is to design dynamic energy management systems based on stochastic models of controlled queues. The goal is to search for optimal control policies for data center management, which should meet the growing demands of reducing energy consumption and digital pollution while maintaining quality of service. We first focused on the modeling of dynamic energy management by a stochastic model for a homogeneous data center, mainly to study some structural properties of the optimal strategy, such as monotony. Afterwards, since data centers have a significant level of server heterogeneity in terms of energy consumption and service rates, we have generalized the homogeneous model to a heterogeneous model. In addition, since the data center server's wake-up and shutdown are not instantaneous and a server requires a little more time to go from sleep mode to ready-to-work mode, we have extended the model to the purpose of including this server time latency. Throughout this exact optimization, arrivals and service rates are specified with histograms that can be obtained from actual traces, empirical data, or traffic measurements. We have shown that the size of the MDP model is large and leads to the problem of the explosion of state space and a large computation time. Thus, we have shown that optimal optimization requiring a MDP is often difficult or almost impossible to apply for large data centers. Especially if we take into account real aspects such as server heterogeneity or latency. So, we have suggested what we call the greedy-window algorithm that allows to find a sub-optimal strategy better than that produced when considering a special mechanism like threshold approaches. And more importantly, unlike the MDP approach, this algorithm does not require the complete construction of the structure that encodes all possible strategies. Thus, this algorithm gives a strategy very close to the optimal strategy with low space-time complexities. This makes this solution practical, scalable, dynamic and can be put online.