



# Leveraging Cloud unused heterogeneous resources for applications with SLA guarantees

Jean-Emile Dartois

## ► To cite this version:

Jean-Emile Dartois. Leveraging Cloud unused heterogeneous resources for applications with SLA guarantees. Software Engineering [cs.SE]. Univ-Rennes1, 2020. English. NNT : . tel-03009816

**HAL Id: tel-03009816**

**<https://inria.hal.science/tel-03009816>**

Submitted on 17 Nov 2020

**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.

# THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES 1  
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601  
*Mathématiques et Sciences et Technologies  
de l'Information et de la Communication*  
Spécialité : Informatique

Thèse présentée et soutenue à Rennes, le 04/09/2020 par

« **Jean-Emile DARTOIS** »

« **Leveraging Cloud unused heterogeneous resources for  
applications with SLA guarantees** »

Unité de recherche : IRISA (UMR 6074) Institut de Recherche en Informatique et Systemes  
Aléatoires

## Rapporteurs avant soutenance :

Daniel HAGIMONT Professeur, Université de Toulouse.  
Romain ROUVOY Professeur, Université de Lille.

## Composition du Jury :

Président :	Adrien Lebre	Professeur, IMT Atlantique
Examineurs :	Kaoutar El MAGHRAOUI	Principal Research Staff Member IBM AI Engineering
	Adrien Lebre	Professeur, IMT Atlantique
Dir. de thèse :	Olivier BARAIS	Professeur, Université de Rennes 1
Co-dir. de thèse :	Jalil BOUKHOBZA	Professeur, Université de Bretagne Occidentale

# ACKNOWLEDGEMENT

---

# PUBLICATIONS

---

## Journal

1. J. Dartois, J. Boukhobza, A. Knefati, and O. Barais. Investigating machine learning algorithms for modeling ssd i/o performance for container-based virtualization. **IEEE Transactions on Cloud Computing** , pages 1–14, 2019.

## International Conferences

2. J. Dartois, A. Knefati, J. Boukhobza, and O. Barais. Using quantile regression for reclaiming unused cloud resources while achieving sla. In 2018 **IEEE International Conference on Cloud Computing Technology and Science** (CloudCom), pages 89–98, Dec 2018.
3. J. Dartois, H. B. Ribeiro, J. Boukhobza, and O. Barais. Cuckoo: Opportunistic mapreduce on ephemeral and heterogeneous cloud resources. In 2019 **IEEE 12th International Conference on Cloud Computing** (CLOUD), pages 396–403, July 2019.
4. J. Dartois, J. Boukhobza, V. Francoise, and O. Barais. Tracking application fingerprint in a trustless cloud environment for sabotage detection. In 2019 **IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems** (MASCOTS), pages 74–82, Oct 2019. This publication is subject to a patent application.
5. J. Dartois, I. Meriau, M. Handaoui, J. Boukhobza and O. Barais. Leveraging cloud unused resources for Big data application while achieving SLA. In 2019 **IEEE 27th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems** (MASCOTS), pages 1–2, Oct 2019.



- 
6. M. Handaoui, J. Dartois, L. Lemarchand, J. Boukhobza. Salamander: a holistic scheduling of mapreduce jobs on ephemeral cloud resources. In 2020 **IEEE/ACM 20th International Symposium on Cluster, Cloud and Grid Computing** (CCGRID), May 2020. <sup>1</sup>

## Communication

7. Meetup Machine Learning Rennes: <https://www.youtube.com/watch?v=UnHIUvNz27Y>

---

1. This publication is not integrated into the thesis, but it addresses one of its limitations.

# TABLE OF CONTENTS

---

1	Contexte . . . . .	12
2	Défis . . . . .	15
3	Énoncé du problème . . . . .	17
4	Contributions . . . . .	19
4.1	Détermination de la capacité réelle du système . . . . .	19
4.2	Estimation des futures ressources inutilisées du Cloud . . . . .	20
4.3	Optimisation de l'exécution d'applications sur des ressources inutilisées du Cloud . . . . .	20
4.4	Vérification de la bonne exécution d'une application dans un environnement sans confiance . . . . .	21
<b>1</b>	<b>Introduction</b>	<b>23</b>
1.1	Context . . . . .	24
1.2	Motivation: Datasets Analysis . . . . .	27
1.3	Challenges . . . . .	29
1.4	Problem Statement . . . . .	31
1.5	Thesis Contributions . . . . .	32
1.5.1	Determining system real capacity . . . . .	33
1.5.2	Estimating future Cloud unused resources . . . . .	33
1.5.3	Adapting applications to run efficiently on Cloud unused resources . . . . .	34
1.5.4	Verifying the correctness of an execution in a trustless environment . . . . .	34
1.6	Outline . . . . .	35
<b>I</b>	<b>Background and State of the Art</b>	<b>37</b>
<b>2</b>	<b>Background</b>	<b>38</b>
2.1	Cloud Computing . . . . .	38

## TABLE OF CONTENTS

---

2.1.1	Fundamentals . . . . .	39
2.1.2	Infrastructure Virtualization . . . . .	44
2.1.3	Cloud Resource Management . . . . .	48
2.1.4	Cloud Infrastructure Management solutions . . . . .	51
2.1.5	Discussion . . . . .	55
2.2	An introduction to Machine Learning . . . . .	56
2.2.1	Learning algorithms . . . . .	58
2.2.2	Machine learning workflow . . . . .	62
2.2.3	Machine learning frameworks and libraries . . . . .	63
2.3	Summary . . . . .	64
<b>3</b>	<b>State of the Art</b>	<b>65</b>
3.1	Performance modeling and I/O interference . . . . .	69
3.2	Cloud time series forecast strategies . . . . .	72
3.3	Improving Hadoop efficiency in volatile and heterogeneous Cloud environments . . . . .	75
3.4	Sabotage-tolerance mechanisms . . . . .	78
3.5	Summary . . . . .	81
<b>II</b>	<b>Contributions and Validations</b>	<b>83</b>
<b>4</b>	<b>PhD Overview</b>	<b>84</b>
<b>5</b>	<b>Estimating real system capacity by considering SSD interferences</b>	<b>87</b>
5.1	Introduction . . . . .	87
5.2	Modeling SSD I/O performance: a machine learning approach . . . . .	90
5.2.1	Approach scope and overview . . . . .	90
5.2.2	Dataset generation step . . . . .	93
5.2.3	Learning step . . . . .	96
5.3	Evaluation . . . . .	99
5.3.1	Evaluation metric . . . . .	99
5.3.2	Experimental setup . . . . .	100
5.3.3	Datasets characteristics . . . . .	100
5.3.4	Prediction accuracy and model robustness . . . . .	101
5.3.5	Learning curve . . . . .	102

5.3.6	Feature importance	104
5.3.7	Training time	105
5.4	Limitations	106
5.5	Summary	106
<b>6</b>	<b>Estimating future use to provide availability guarantees</b>	<b>108</b>
6.1	Introduction	108
6.2	Methodology	110
6.2.1	Quantiles	110
6.2.2	Approach overview	111
6.2.3	Forecast Strategy step	112
6.2.4	Data pre-processing step	114
6.2.5	Evaluation step	115
6.3	Evaluation	115
6.3.1	Experimental setup	116
6.3.2	Flexibility: potential cost savings (RQ1)	118
6.3.3	Exhaustivity: impact of relying on a single resource (RQ2)	121
6.3.4	Robustness: resilience to workload change (RQ3)	122
6.3.5	Applicability: training overhead (RQ4)	122
6.3.6	Threats to validity	123
6.4	Summary	124
<b>7</b>	<b>Leveraging Cloud unused resources for big data</b>	<b>126</b>
7.1	Introduction	126
7.2	MapReduce and Hadoop	128
7.2.1	MapReduce programming model	128
7.2.2	Hadoop framework architecture	128
7.3	Cuckoo: a Mechanism for Exploiting Ephemeral and Heterogeneous Cloud Resources	129
7.3.1	The Cuckoo Framework Architecture Overview	129
7.3.2	Forecasting Builder	131
7.3.3	Data Placement Planner	131
7.3.4	QoS Controller	134
7.4	Experimental Validation	135
7.4.1	Experimental Methodology	135

## TABLE OF CONTENTS

---

7.4.2	Data sets . . . . .	136
7.4.3	Experimental Results . . . . .	137
7.5	Limitations . . . . .	140
7.6	Summary . . . . .	141
<b>8</b>	<b>Preventing malicious infrastructure owners from sabotaging the computation</b>	<b>142</b>
8.1	Introduction . . . . .	142
8.2	Methodology . . . . .	144
8.2.1	Fingerprint Builder: Building the fingerprint models in an environment of trust . . . . .	145
8.2.2	Fingerprint Tracker: tracking application executions . . . . .	149
8.3	Evaluation . . . . .	150
8.3.1	Experimental setup . . . . .	151
8.4	Discussions . . . . .	156
8.5	Summary . . . . .	157
<b>9</b>	<b>An architecture implementation to leverage Cloud unused resources</b>	<b>158</b>
<b>III</b>	<b>Conclusion &amp; Perspectives</b>	<b>162</b>
<b>IV</b>	<b>Indexes</b>	<b>171</b>
	<b>Bibliography</b>	<b>176</b>

# GLOSSARY

---

**AdaBoost** Adaptive Boosting. [88](#), [96](#)

**CaaS** Container as a Service. [40](#), [41](#), [51](#), [56](#)

**CP** Cloud Provider. [14](#), [20](#), [26](#), [33](#), [115](#), [117](#), [118](#)

**CPs** Cloud Providers. [14](#), [24](#), [26](#), [43](#), [44](#), [49](#), [81](#), [123](#)

**DT** Decision trees. [88](#), [96](#)

**GBDT** Gradient Boosting Decision Trees. [20](#), [33](#), [88](#), [96](#), [114](#), [118–120](#), [122](#), [123](#), [125](#)

**GC** Garbage Collection. [70](#), [71](#), [87–90](#)

**HDDs** Hard Disk Drives. [81](#)

**IaaS** Infrastructure as a Service. [40](#), [41](#), [55](#), [158](#)

**LSTM** Long Short Term Memory. [20](#), [33](#), [72](#), [73](#), [114](#), [115](#), [118–120](#), [122](#), [123](#), [125](#)

**MARS** Multivariate adaptive regression splines. [88](#), [96](#)

**ML** Machine Learning. [56](#)

**NRMSE** Normalized Root-Mean-Square Error. [20](#), [33](#)

**OS** Operating System. [44](#), [45](#), [47](#), [56](#), [84](#)

**PaaS** Platform as a Service. [40](#), [41](#)

**QoS** Quality of Service. [14](#), [16](#), [21](#), [24](#), [25](#), [29](#), [34](#), [42](#), [49](#), [55](#)

**RF** Random Forests. [20](#), [21](#), [33](#), [34](#), [72](#), [88](#), [96](#), [114](#), [118–120](#), [122](#), [123](#), [148](#), [157](#)

**RNN** Recurrent Neural Network. [72](#)

**SaaS** Software as a Service. [40](#), [41](#), [56](#)

**SLA** Service Level Agreements. [14](#), [15](#), [17](#), [18](#), [20](#), [25](#), [26](#), [29](#), [31–33](#), [41](#), [65](#), [70](#), [78](#), [81](#), [82](#), [85](#), [114–119](#), [121](#), [123–125](#), [173](#)

**SLO** Service Level Objective. [90](#)

**SLOs** Service Level Objectives. [87](#)

**SSD** Solid-state drive. [19](#), [33](#), [68–72](#), [81](#), [87–90](#)

**SSDs** Solid-state drives. [69–72](#), [81](#), [88–90](#)

**SVM** Support Vector Machine. [57](#), [72](#)

**TCO** Total Cost of Ownership. [13](#), [18](#), [24](#), [25](#), [32](#), [142](#)

# RÉSUMÉ EN FRANÇAIS

---

Depuis des milliers d'années, l'humanité n'a jamais cessé de produire des données et de partager des connaissances. Les premières traces remontent au 4<sup>e</sup> millénaire av. J.-C. lorsque la Mésopotamie a fait face à la complexité du commerce et de l'administration. Les connaissances dépassant les capacités de la mémoire humaine, l'écriture était devenue une nécessité pour enregistrer les échanges commerciaux [167].

Aujourd'hui, les avancées technologiques telles que l'Internet des Objets ont abouti à une avalanche de données. Traiter ces données pourrait permettre, par exemple, de réduire la mortalité et la morbidité des nouveau-nés en prévoyant les risques d'infection, de comprendre l'univers en utilisant les données produites par le Grand collisionneur de hadrons (LHC) [42], de minimiser la consommation énergétique du refroidissement des centres informatiques [60], d'accroître le chiffre d'affaire et la rentabilité des entreprises, ou d'autres scénarios.

Il est important de stocker et d'analyser ces données pour des raisons à la fois économiques et sociales. Le traitement de ces données exige cependant une quantité considérable de ressources informatiques et de stockage [92].

Selon des estimations récentes (*i.e.*, 2019) [92] d'ici 2025, la quantité de données générées par l'humanité sera d'environ 160 zettabytes<sup>2</sup>. En 2016, la capacité de calcul requise par l'Organisation Européenne pour la Recherche Nucléaire (CERN) devrait être en 2025 de 50 à 100 fois supérieures à celle d'aujourd'hui, les besoins de stockage des données devant être de l'ordre des exabytes [32]. Les progrès des nouvelles technologies permettent de relever progressivement ces défis. Par exemple, en 1983, CompuServe a offert à ses clients un stockage de données Cloud de 128Koctets [124]. Ensuite, le paradigme du Cloud computing a été popularisé [36] en fournissant un accès à la demande à des ressources informatiques et de stockage évolutives, élastiques et fiables. Enfin, en 2005, Hadoop propose une mise en oeuvre Open-Source de MapReduce qui permet de traiter une grande quantité de données sur des clusters constitués de milliers de nœuds de calcul [154].

---

2. zettabytes est une unité de mesure égale à  $10^{21}$  bytes



## 1 Contexte

Pour traiter les données, de nombreux acteurs s'appuient aujourd'hui sur des plateformes de Cloud computing qui permettent la mobilisation de ressources physiques à grande échelle. Les infrastructures Clouds sont complexes à opérer, et leur efficacité peut être encore améliorée. Ainsi, de nombreux travaux de recherche sont menés pour améliorer leur performance, réduire leur coût d'exploitation et améliorer la sécurité.

Du point de vue des clients, les plateformes de Cloud computing présentent de nombreux avantages comme l'accès à la demande à des ressources informatiques évolutives, élastiques et fiables, une interface simplifiée, et des mécanismes tolérants aux pannes. De plus, l'offre de services permet de choisir le matériel adapté et fournit des technologies simplifiant le traitement des données massives en tant que service.

Du point de vue d'un fournisseur de Cloud computing, l'objectif principal est de garantir une bonne qualité de service (QoS) pour les clients tout en réduisant leur coût total de possession (en anglais : Total Cost of Ownership ou TCO) [11]. Le TCO est la somme de tous les coûts liés à l'achat, à l'exploitation, à l'entretien et à la maintenance d'une infrastructure Cloud. Pour atteindre cet objectif, les fournisseurs de Cloud computing ont construit des centres de données à plus grande échelle et ont massivement adopté des technologies de virtualisation pour partager les ressources entre les clients. Ces centres de données représentent un investissement important. En 2019, Google prévoit d'investir plus de 13 milliards de dollars dans les centres de données et des bureaux aux États-Unis [76]. 45% des coûts des centres de données sont liés à l'achat des serveurs physiques et de leurs composantes (c.-à-d., CPU, mémoire et stockage), et environ 25% sont liés à la distribution d'électricité et systèmes de refroidissement [78].

La gestion des ressources est une préoccupation majeure pour les fournisseurs de Cloud afin d'améliorer l'utilisation des infrastructures et ainsi réduire les coûts. Bien que l'usage de la virtualisation ait amélioré l'utilisation des ressources informatiques dans les centres de données [130], plusieurs études ont démontré que l'utilisation moyenne des ressources reste faible, entre 20% et 50% pour le CPU [41, 126]. Cette faible utilisation peut s'expliquer par plusieurs facteurs :

- **Gestion des pics** : L'infrastructure Cloud doit être surdimensionnée pour

répondre aux pics de la demande. Par conséquent, une partie des serveurs physiques de l'infrastructure tend à être inutilisée pendant les périodes creuses. Par exemple, les fans de Lady Gaga ont généré un pic de charge après que son album "Born This Way" ait été proposé en ligne pour 99 cents [144].

- **Tolérance aux Pannes** : Pour faire face aux pannes matérielles ou aux besoins de reprise après un incident, la capacité des centres de données est surdimensionnée, allant au-delà des besoins réels et/ou volontairement déployée dans plusieurs zones géographiques. Ce surdimensionnement augmente le TCO pour les fournisseurs de Cloud et se traduit par une faible utilisation moyenne des ressources.
- **Gestion de la demande future** : L'achat du matériel informatique doit tenir compte de la demande future et est souvent surprovisionné.
- **Conception** : L'architecture et la conception du réseau peuvent constituer des contraintes ou des obstacles, parfois pour des raisons de sécurité, qui empêchent l'utilisation des ressources entre plusieurs services au sein d'une même entreprise. Cela inclut par exemple des architectures hétérogènes au sein d'une même entreprise, comme OpenStack et VMware [171, 94].

L'optimisation des ressources dans une infrastructure Cloud nécessite de surveiller en permanence les ressources inutilisées sur la base d'un ensemble de mesures ( $mtr$ , *e.g.*, utilisation CPU) au moment  $t$  comme suit :

$$Nonutilise_{(t,mtr)} = Cap_{(t,mtr)} - Utilise_{(t,mtr)} \quad (1)$$

Ainsi,  $Cap_{(t,mtr)}$  est la capacité de performance maximale accessible par le système pour  $mtr$  au moment  $t$  ; et  $Used_{(t,mtr)}$  est la capacité utilisée pour  $mtr$  au temps  $t$ .

Avant d'investir dans de nouvelles infrastructures physiques qui impliquent des coûts de matériel et d'énergie, une façon d'améliorer l'utilisation des ressources du centre de données, et donc de réduire le TCO, est de (re)vendre à d'autres entreprises les ressources inutilisées [39]. Cependant, la revente des ressources

doit répondre aux attentes des clients en terme de **QoS** tout en évitant des interférences entre les applications utilisant les ressources inutilisées du Cloud et les charges de travail co-résidentes (*i.e.*, les fournisseurs de ressources). La **QoS** est généralement définie en termes de Service Level Agreements (**SLA**). En cas de non-respect de ces accords, les fournisseurs de services Clouds sont exposés aux plaintes des clients et risquent des pénalités.

Le but des **CPs** est de maximiser la quantité de ressources récupérées tout en évitant les risques de pénalités. Il existe trois types de pénalités basées sur l'application d'une remise [72] : (i) une *pénalité fixe* où chaque fois que le **SLA** est transgressé un rabais est appliqué, (ii) une *pénalité dépendante du délai* pour laquelle la remise est liée au retard dans la restitution de la capacité convenue par le **CP**. Dans ce cas, le client a négocié avec le **CP** le nombre maximum de minutes consécutives de non-respect. Si le niveau de capacité est restitué avant cet intervalle, aucune remise n'est appliquée. (iii) une *pénalité proportionnelle* où la remise est proportionnelle à la différence entre la capacité convenue et la capacité mesurée. Les Clouds publics tels qu'OVH, Amazon et Google utilisent une approche hybride (*pénalité fixe* et *pénalité dépendant du délai*) [72].

Cette thèse fait partie d'un projet mené par l'Institut de Recherche et Technologie b<>com. Ce projet vise à développer une solution de Cloud collaboratif (une forme d'AirBnB des centres de données) ayant pour but de regrouper et de mettre à disposition de manière sécurisée les ressources informatiques non utilisées de multiples entreprises et administrations publiques. Il s'agit d'une alternative au Cloud public, à la fois bas coût et souveraine, particulièrement adaptée à des applications de traitement de données de type Big Data. Cependant, l'opérateur (*i.e.*, interface entre le propriétaire de l'infrastructure et les clients) doit encore répondre aux attentes de ses clients en termes de qualité de service tout en évitant l'interférence entre les traitements massifs et les charges de travail des co-résidents (*i.e.*, les fournisseurs de ressources).

Figure 1 présente les rôles du projet :

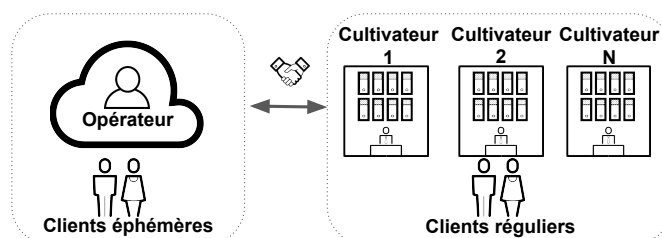


Figure 1: Projet global

- **Clients:** il y a deux types de clients. Les *clients réguliers* qui achètent et/ou réservent des ressources stables du Cloud, et les *clients éphémères* (clients utilisant des ressources éphémères) qui souhaitent héberger des applications sur le Cloud à moindre coût.
- **Cultivateurs:** propriétaires de centres de données, qui cherchent à réduire leur TCO en offrant des ressources inutilisées à des clients éphémères.
- **Opérateur:** sert d'interface entre le(s) cultivateur(s) et les clients. L'objectif de l'opérateur est de minimiser le TCO des cultivateurs en offrant les ressources inutilisées aux clients éphémères avec des exigences SLA tout en évitant des interférences pour les clients réguliers. Dans le cas d'une entreprise qui veut tirer profit de ses propres ressources inutilisées dans le Cloud pour ses propres besoins, l'opérateur peut être internalisé.

## 2 Défis

Cette thèse s'intéresse à quatre des six défis (voir Figure 2) du projet IRT b<>com, qui cherche à exploiter les ressources inutilisées du Cloud pour déployer des applications tout en respectant des [SLA](#). Les six défis sont les suivants:



**Garantir le SLA des utilisateurs:** Lors de l'exécution d'applications sur des ressources allouées mais non utilisées, il faut éviter le non-respect de [SLA](#) pour l'utilisateur ayant réservé ces ressources. Ainsi, il est nécessaire

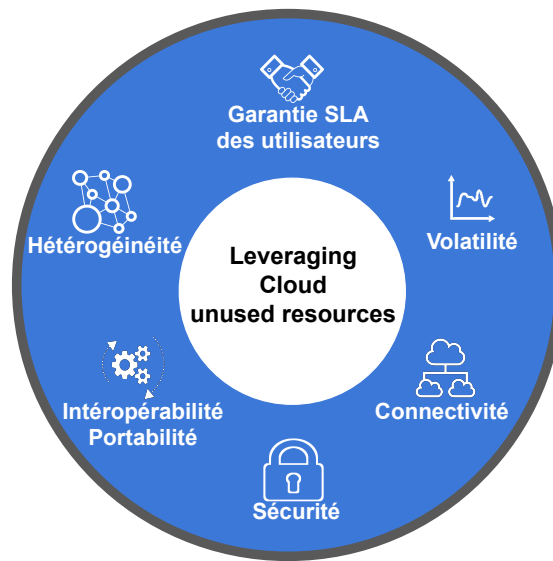


Figure 2: Défis du projet IRT b<com

d'avoir la capacité d'allouer, de réagir et d'adapter rapidement le provisionnement des ressources inutilisées de manière à éviter de dégrader les **QoS** pour les clients réguliers qui ont réservé ces ressources.



**Gérer la volatilité des ressources:** Dans les systèmes Cloud, les utilisateurs peuvent réserver, consommer et libérer unilatéralement des ressources informatiques à la volée. D'autre part, la nature des charges de travail est très hétérogène et leur intensité peut varier significativement (*i.e.*, augmenter ou diminuer brusquement) en fonction du comportement de l'utilisateur [37].



**Assurer la connectivité:** L'interconnexion de deux ou plusieurs centres de données est indispensable pour permettre d'agréger et de partager les ressources inutilisées. Toutefois, les technologies utilisées et les performances réseaux peuvent varier, ce qui rend difficile le déploiement d'applications sans perturbation.



**Garantir la sécurité:** La sécurité est essentielle pour protéger la vie privée des clients, les données ou le code informatique sensibles. C'est pourquoi,

plusieurs défis doivent être pris en compte dans le Cloud computing, tels que l'authentification, la confidentialité, et l'intégrité des données et du code informatique.



**Prendre en compte la portabilité et l'interopérabilité:** Chaque fournisseur de Cloud computing peut utiliser une approche différente pour fournir le service à ses clients et peut déployer un large éventail d'API différentes, ce qui entraîne une complexité et des difficultés pour l'interopérabilité.



**S'adapter à l'hétérogénéité des Clouds:** Les infrastructures de Cloud sont construites sur des ressources hétérogènes pour éviter l'effet de verrouillage des fournisseurs mais aussi à cause du renouvellement du matériel. Ainsi, la récupération des ressources doit être flexible et s'adapter à des capacités de stockage et de traitement variables.

### 3 Énoncé du problème

Répondre à ces défis implique de définir les questions ou problèmes clés (voir Figure 3):

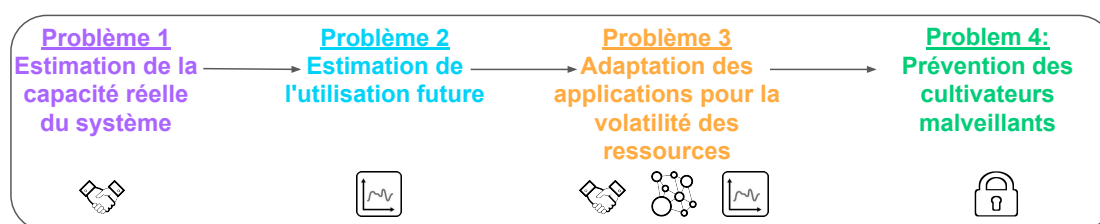


Figure 3: Une carte des problèmes et des défis

**Problème 1 (Capacité réelle du système):** Garantir le SLA des utilisateurs implique d'estimer la performance maximale atteignable par un système pour déterminer la capacité. Plusieurs études ont souligné que des applications colocalisées peuvent interférer et entraîner des baisses de performance.

Cela peut être dû à des interférences au niveau du matériel, des mécanismes du système (*e.g.*, SSD, CPU, mémoire) ou de la virtualisation. Parmi les ressources partagées, les entrées/sorties (E/S) constituent le principal goulot d'étranglement [7]. Fournir une estimation précise des E/S  $Cap_{(t,mtr)}$  est essentiel pour les garanties SLA, mais:

***comment modéliser les variations de performance ?***

**Problème 2 (Estimation de l'utilisation future):** Une fois la capacité estimée, il est important de fournir une estimation précise des quantités futures de ressources utilisées  $Used_{(t,mtr)}$ . Toutefois, dans un contexte de forte volatilité des ressources, il est nécessaire d'atténuer le risque d'une estimation inexacte.

***Comment pouvons-nous estimer, de manière flexible et précise, l'utilisation future des ressources et garantir la disponibilité ?***

**Problème 3 (Applications consciente de la volatilité des ressources):** Les applications sont conçues et développées en partant de l'hypothèse que les ressources sont disponibles tant que les utilisateurs paient pour le service. Cette hypothèse n'est pas compatible avec nos défis (*i.e.*, garantie des utilisateurs SLA, volatilité des ressources). En effet, pour garantir que les applications en cours d'exécution des *clients réguliers* n'interfèrent pas avec les charges de travail régulières des *clients réguliers*, la ressource allouée pourrait être préemptée.





***Comment des applications de type big data peuvent elle être adaptées pour s'exécuter sur des ressources éphémères hétérogènes ?***

**Problème 4 (Prévention des cultivateurs malveillants):** Bien que les problèmes 1, 2 et 3 s'appliquent à tous les types de modèles de Clouds, le problème 4 traite d'un problème spécifique rencontré dans un environnement de Clouds communautaires. Dans une telle infrastructure de Cloud, tout *cultivateurs* peut se joindre pour fournir/partager ses capacités de calcul. Ces cultivateurs cherchent à réduire leur TCO en revendant leurs ressources informatiques inutilisées. Permettre à tout cultivateur de rejoindre de telles plateformes expose un *Opérateur* à des comportements malveillants. Des

cultivateurs malveillants peuvent potentiellement produire des résultats erronés ou imprécis sans exécuter efficacement les applications pour obtenir des bénéfices plus importants (e.g., tout en économisant leurs capacités de calcul) [155].

***Comment empêcher les propriétaires d'infrastructures malveillants de saboter le calcul en soumettant de mauvais résultats ?***

## 4 Contributions


Dans cette thèse, nous défendons l'idée que les ressources inutilisées peuvent être utilisées pour déployer des applications à moindre coût. Parmi les six défis étudiés à b<>com, nous abordons spécifiquement dans cette thèse quatre défis (i.e.,  garantie SLA des utilisateurs,  volatilité des ressources,  hétérogénéité des Clouds, et  sécurité).

### 4.1 Détermination de la capacité réelle du système

Pour répondre au problème 1 (*Capacité réelle du système*), nous avons conçu un cadre basé sur le calcul autonome qui vise à réaliser un placement intelligent des conteneurs sur les systèmes de stockage en empêchant les mauvais scénarios d'interférence E/S. Une condition préalable à un tel cadre est de concevoir des modèles de performance SSD qui prennent en compte les interactions entre les processus/conteneurs en cours d'exécution, le système d'exploitation et le SSD. Ces interactions sont complexes. Nous avons étudié l'utilisation de l'apprentissage automatique pour construire de tels modèles dans un environnement Cloud basé sur des conteneurs. Nous avons étudié cinq algorithmes populaires d'apprentissage automatique ainsi que six applications et benchmarks différents à forte intensité d'E/S. Nous avons analysé la précision de la prédiction, la courbe d'apprentissage, l'importance des caractéristiques et le temps d'entraînement des algorithmes testés sur quatre modèles de SSD différents. Au-delà de la description de la composante modélisation de notre cadre de travail, ce travail vise à fournir des indications aux fournisseurs de Cloud computing pour mettre en œuvre des algorithmes de placement de conteneurs conformes




à la norme SLA sur les SSD. Notre framework basé sur l'apprentissage automatique a réussi à modéliser les interférences d'E/S avec une médiane **NRMSE** de 2,5%.

Cette contribution traite de  Garantie SLA des utilisateurs. Ce travail a été publié dans la revue IEEE Transaction on Cloud computing 2019 [49].

## 4.2 Estimation des futures ressources inutilisées du Cloud




Pour répondre au problème 2 (*Estimation de l'utilisation future*), nous avons introduit un modèle prédictif pour déterminer les ressources disponibles et estimer leur utilisation future afin de fournir des garanties de disponibilité. Notre contribution propose une technique qui utilise des algorithmes d'apprentissage automatique (*i.e.*, **RF**, **GBDT**, et **LSTM**) pour prévoir 24 heures de ressources disponibles pour chaque machine physique. L'une des principales contributions est l'utilisation de la régression quantile pour rendre notre modèle prédictif flexible pour le **CP**, plutôt que d'utiliser la simple régression moyenne de l'utilisation des ressources. Cela permet à un **CP** de faire un compromis pertinent et précis entre le volume des ressources qui peuvent être louées et le risque de non-respect du **SLA**. En outre, plusieurs métriques (*e.g.*, CPU, RAM, disque, réseau) ont été prédites pour fournir des garanties de disponibilité exhaustives. Notre méthodologie a été évaluée en s'appuyant sur quatre traces de centres de données de production. Nos résultats montrent que la régression quantile est pertinente pour récupérer les ressources inutilisées. Notre approche permet de réaliser jusqu'à 20% d'économie par rapport aux approches traditionnelles.

Cette contribution porte sur  Le défi de la volatilité des ressources. Ce travail a été publié dans la conférence internationale de l'IEEE Cloud computing 2018 (CloudCom) [50].

## 4.3 Optimisation de l'exécution d'applications sur des ressources inutilisées du Cloud


Pour répondre au problème 3 (*Applications consciente de la volatilité des ressources*), nous avons conçu un framework qui exploite les ressources inutilisées des centres de données, qui sont par nature éphémères, pour exécuter

les jobs MapReduce. Notre approche permet : *i*) d'exécuter efficacement des jobs Hadoop sur des ressources hétérogènes du Cloud, grâce à notre stratégie de placement des données, *ii*) de prédire précisément la volatilité des ressources éphémères, grâce à la méthode de régression quantile (basée sur la contribution 4.2), et *iii*) d'éviter l'interférence entre les jobs MapReduce et les charges de travail co-résidentes, grâce à notre contrôleur réactif QoS. Nous avons étendu l'implémentation de Hadoop avec notre framework et l'avons évalué avec trois différentes charges de travail de centre de données. Les résultats expérimentaux montrent que notre approche améliore le temps d'exécution des jobs Hadoop jusqu'à 7 fois par rapport à l'implémentation standard de Hadoop.

Cette contribution aborde :  garantie SLA des utilisateurs,  la volatilité des ressources, et  les défis de l'hétérogénéité des Clouds. Ce travail a été publié dans la conférence internationale de l'IEEE Cloud 2019 (Cloud) [48].

#### 4.4 Vérification de la bonne exécution d'une application dans un environnement sans confiance

Pour répondre au problème 4 (*Prévention des cultivateurs malveillants*), nous avons proposé une approche qui permet de détecter le sabotage dans un environnement sans confiance. Pour ce faire, (1) nous avons conçu un mécanisme qui construit une empreinte d'application en considérant un large ensemble d'utilisation de ressources (e.g., CPU, I/O, mémoire) dans un environnement de confiance en utilisant l'algorithme Random Forest (RF), et (2) un dispositif de reconnaissance par empreinte fonctionne en continu et à distance pour surveiller la bonne exécution de l'application. Ce dispositif permet de détecter un comportement imprévu de l'application. Notre approche a été testée en construisant l'empreinte digitale de 5 applications sur des machines de confiance. Lors de l'exécution de ces applications sur des machines non fiables (avec un matériel homogène, hétérogène ou non spécifié par rapport à celui qui a été utilisé pour construire le modèle), le système de reconnaissance par empreinte digitale a pu déterminer si l'exécution de l'application est correcte ou non avec une précision médiane d'environ 98% pour le matériel hétérogène et d'environ 40% pour le matériel non spécifié.

Cette contribution traite du problème de sécurité . Ce travail a été publié

dans la conférence internationale de l'IEEE : Modeling, Analysis, and Simulation On Computer and Telecommunication Systems (MASCOTS) 2019 [47] et ces résultats ont été brevetés.

# INTRODUCTION

---

For thousands of years, humanity has never stopped accumulating data and transferring knowledge. First signs were found back to the 4th millennium BC when Mesopotamia decided to address the complexity of trade and administration. Knowledge exceeding human memory, writing became a necessity for storing transactions [167].

Nowadays, advances in technologies such as the Internet of Everything led us to a data deluge. The processing of this data could enable, for instance, to reduce mortality and morbidity of newborns by forecasting their sepsis risk, to understand the universe using the data produced by Large Hadron Collider (LHC) [42], to minimize energy consumption of data center cooling [60], to increase revenue and profitability, and many other applications.

It is a high stake to store and analyze these data both for economic and social reasons. However, processing these data demands a considerable amount of computing and storage resources [92].

According to recent estimations (*i.e.*, 2019) [92], by 2025 the amount of data generated by humanity will be about 160 zettabytes <sup>1</sup>. In 2016, the European Organisation for Nuclear Research (CERN) required computing capacity is expected to be in 2025 up to 50-100 times greater than today's, with data storage needs expected to be in the magnitude of exabytes <sup>2</sup> [32]. Advance in new technologies enables to progressively address these challenges. For example, in 1983 CompuServe offered to its customers a 128KB Cloud data storage [124]. Then, the Cloud computing paradigm was popularized [36] providing on-demand access to scalable, elastic, and reliable computing and storage resources. These features make Cloud infrastructures good candidates for processing big data workloads. An example in 2005, Hadoop offered an Open-source implementation of MapRe-

---

1. zettabytes is a unit of measurement equal to  $10^{21}$  bytes

2. exabytes is a unit of measurement equal to  $10^{18}$  bytes

duce that enables to process large amounts of data across clusters of thousands of computing nodes [154].

## 1.1 Context

To process data, many stakeholders rely nowadays on Cloud platforms that enable mobilization of large-scale physical resources. Cloud infrastructures are complex to operate, and their efficiency has yet to be improved. Many research works are conducted to improve Cloud performance, security and reduce their operating costs.

From the customers' point of view, Cloud platforms have numerous benefits such as on-demand access to scalable, elastic, reliable computing resources, simplified interface, and fault-tolerant mechanisms. Services in the Cloud offer a choice for the underlying hardware and provide Big Data technologies as a service able to manage the complexity of the underlying system.

From a Cloud provider's (CPs) view, the main objective is to ensure a good quality of service (QoS) for customers while reducing their Total Cost of Ownership (TCO) [11]. The TCO is the sum of all costs involved in the purchase, operation and maintenance of a Cloud infrastructure. To achieve this goal, CPs have built large scale data centers and massively adopted virtualization technologies to share resources between customers. These data centers represent a significant investment. In 2019, Google plans to invest more than 13 billion dollars on data centers and offices in the United States [76]. 45% of the costs of data centers are related to the purchase of the physical servers and their components (*i.e.*, CPU, memory, and storage), and about 25% are related to the power distribution and cooling systems [78].

Managing resources in order to improve their utilization and reduce costs is a major concern for Cloud providers. Although the use of virtualization has improved the utilization of computing resources in data centers [130], several studies have demonstrated that the average usage of resources remains low, between 25-35% for the CPU and 40-50% for the RAM [41, 52, 50]. This low utilization can be explained by several factors:

- **Peak Handling:** The Cloud infrastructure needs to be over-provisioned to

handle peak demand. Consequently, a portion of the infrastructure physical servers tends to be unused during non-peak periods. For example, Lady Gaga fans have generated a peak load that brought down the vast server resources of Amazon.com after her album "Born This Way" was offered online for 99 cents [144].

- **Risk Taming:** To handle hardware failures or disaster recovery needs, data centers capacity is oversized, going beyond the real needs and/or need to be deployed in several geographic zones. This oversizing increases the TCO for Cloud providers and results in a low average resource utilization.
- **Future demand handling:** Purchase of hardware equipment is based on expected future demands and peaks, and for that reason is over-provisioned.
- **Design:** The architecture and network design, sometimes due to security reasons, may include constraints or barriers that prevent resource utilization across a wider range of services within the same company. This includes for example heterogeneous architectures within a single company, such as OpenStack and VMware [171, 94].

Optimizing resources in a Cloud infrastructure requires to constantly monitor unused resources based on a set of metrics ( $mtr$ , e.g., CPU usage) at time  $t$  as follows:

$$Unused_{(t,mtr)} = Cap_{(t,mtr)} - Used_{(t,mtr)} \quad (1.1)$$

where  $Cap_{(t,mtr)}$  is the maximum performance capacity reachable by the system for  $mtr$  at time  $t$  and  $Used_{(t,mtr)}$  is the used capacity for  $mtr$  at time  $t$ .

One way to improve Cloud data center resource utilization and thus reduce the TCO is to (re)sell to other companies unused resources [39]. However, re-selling resources needs to meet the expectations of its customers in terms of QoS while avoiding the interference between applications relying on Cloud unused resources and co-resident workloads (i.e., the resource providers). QoS is usually defined in terms of Service Level Agreements (SLA). In case of violations of these agreements, Cloud providers are exposed to complaints by customers and are prone to penalties.

The goal of **CPs** is to maximize the amount of reclaimed resources while avoiding the risk of penalties. There are three types of penalties based on applying a discount [72]: (i) a *fixed penalty* where each time the **SLA** is violated a discount is applied; (ii) a *delay-dependent penalty* for which the discount is related to the response delay by the **CP** in providing the agreed capacity. In this case, the customer has negotiated with the **CP** the maximum number of consecutive minutes of violations. If the capacity level is provided back before this interval, no discount is applied; (iii) a *proportional penalty* where the discount is proportional to the difference between the agreed upon and the measured capacity. Public Cloud such as OVH, Amazon and Google use a hybrid approach (*fixed penalty* and *delay-dependent penalty*) [72].

This thesis is part of a project led by the Institute of Research and Technology b<>com. This project aims to make unused and heterogeneous private IT resources available through a highly secured distributed Cloud to deploy applications at a cheaper price. The first use case of the project is to provide a framework that leverages unused Cloud resources to run big data jobs. However, the operator (*i.e.*, interface between the infrastructure owner and the customers) still has to meet the expectations of its customers in terms of Quality of Service while avoiding the interference between big data jobs and co-resident workloads (*i.e.*, the resource providers).

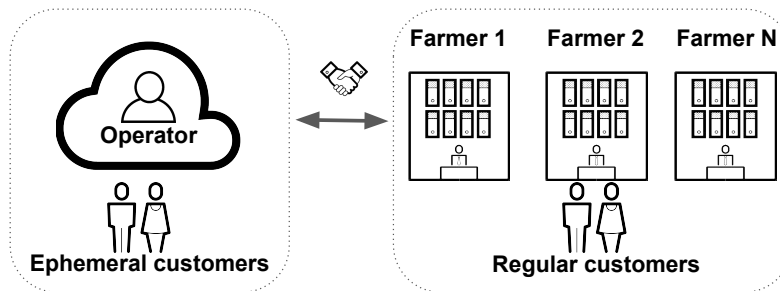


Figure 1.1: Overall project

Figure 1.1 presents the roles of the project:

- **Customers:** there are two types of customers. First, *regular customers* that buy and/or reserve stable Cloud resources. Second, *ephemeral customers* (customers using ephemeral resources) that want to host applications on the Cloud at a lower cost.

- **Farmers:** data center owners, that seek to reduce their TCO by offering unused resources to ephemeral customers.
- **Operator:** acts as the interface between farmer(s) and customers. The operator objective is to minimize farmers' TCO by offering unused resources to ephemeral customers with SLA requirements while avoiding interference with regular customers. In the case of a company that wants to make profit of its own Cloud unused resources for its own needs, the operator is internalized.

## 1.2 Motivation: Datasets Analysis

This section motivates the work in this thesis by providing some analysis about four in-production data center traces. These traces were collected between 2015 and 2017 from various types of organizations (*i.e.*, one university, one public administration and two private companies).

First, we focus on one data center at the host level, and then we give an overview of the resources for all data centers. Table 1.1 shows the hardware characteristics of the hosts for *private Company 1*. A first observation one may draw is that its hosts are heterogeneous (proportion between CPU and RAM). Thus, reclaiming resources on some hosts could be more effective than on others.

Table 1.1: Hosts characteristics of private company 1

HostID	CPU Cores	RAM [GB]	CPU MODEL
12.0.0.1	20	300	Intel(R) Xeon(R) 2.20GHz
12.0.0.2	20	130	Intel(R) Xeon(R) 2.20GHz
12.0.0.3	12	130	Intel(R) Xeon(R) 2.30GHz
12.0.0.4	8	130	Intel(R) Xeon(R) 2.40GHz
12.0.0.5	12	130	Intel(R) Xeon(R) 2.30GHz
12.0.0.6	12	130	Intel(R) Xeon(R) 2.30GHz
12.0.0.7	12	130	Intel(R) Xeon(R) 2.30GHz
12.0.0.8	12	130	Intel(R) Xeon(R) 2.30GHz
12.0.0.9	12	130	Intel(R) Xeon(R) 2.30GHz

Let us focus on CPU and RAM in this section. Fig. 1.2a shows the box plots of CPU usage for the nine hosts. We observed that 75% of the time, the CPU median usage is under 40% for the hosts 12.0.0.1, 12.0.0.2 and 12.0.0.5. For the other hosts, the CPU median usage is even less than 20% during 75% of the time.



We notice in the box-plots of Fig. 1.2b that the median usage of *RAM* is higher (about 50%) compared to *CPU*. This may be explained by the fact that in a virtualized environment the *RAM* is progressively allocated to the virtual machines but rarely released except when a memory management technique, such as memory ballooning (see Background I part), is enabled.

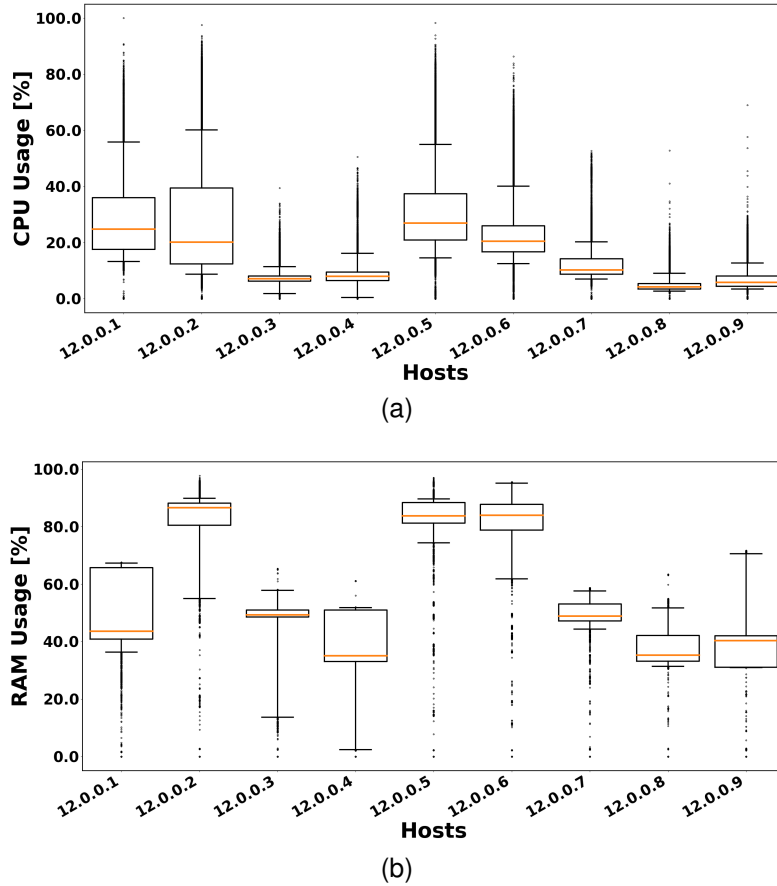


Figure 1.2: Box plots of (a) *CPU* and (b) *RAM* usage for each host with *Private Company 1*

### Potential Reclaimable Resources

Table 1.2 shows the overall capacity of the data centers used in this study. The *Private Company 2* data center is the largest one with 356 cores and 3.8 TB of memory provided by 27 hosts.

Table 1.3 shows the average usage of the data centers for CPU, RAM, storage and network resources. One can notice that the four data centers have a maxi-

Table 1.2: Available aggregated  $Cap(t, mtr)$  of the data centers

Name	Number of Hosts	Duration [months]	CPU cores	RAM [TB]
University	10	22	116	1.5
Public Administration	7	35	240	2.5
Private Company 1	9	12	120	1.2
Private Company 2	27	17	356	3.8

mum average CPU usage of 17 % at the host-level. This motivates our study as one can reclaim large amounts of resources to reduce the CP costs.

Table 1.3: Average usage of resources calculated at the host-level

Name	CPU Usage [%]	RAM Usage [%]	Disk R/W [Mb/s]	Network In/Out [Mb/s]
University	9.7	55.2	7.9/2.9	9.3/4.7
Public Administration	14.4	54.1	12/7.5	2/6.4
Private Company 1	17	57	10.6/3	7.9/2.1
Private Company 2	10.9	48	1/0.3	7.1/7.7

To conclude, from the four data centers investigated, all of them have a low resource usage. Moreover, in [40] authors analyzed 6 real-world, production Cloud computing clusters at Google and show that more than 45% of the CPU, 43% of the memory and 89% of the disk capacity are unused. This encourages the use of reclaiming techniques. Secondly, in a given data center, configurations appear to be heterogeneous, and so resource usage is not balanced among hosts. This motivates the design of reclaiming technique at the host-level granularity.

### 1.3 Challenges

This thesis addresses four out of the six challenges (see Figure 1.3) of the IRT b<>com project, which seeks to leverage Cloud unused resources for deploying applications while achieving SLA.



**Users SLA guarantee:** When running applications on allocated but unused resources, one should hedge against violating SLA for the user's having reserved those resources. Thus, it is necessary to have the ability to quickly allocate, react and adapt the unused resource provisioning in a way to avoid degrading the QoS for the regular customers that have reserved those resources.

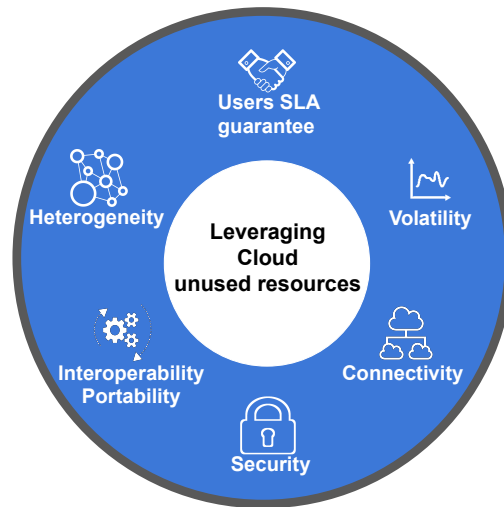


Figure 1.3: IRT b<>com project challenges



**Resources volatility:** In Cloud systems, users are able to unilaterally reserve, consume and release computing resources on-the-fly. On the other hand, the nature of workloads is highly heterogeneous and their intensity may significantly vary (*i.e.*, abrupt grow or shrink) according to the user's behavior [37].



**Connectivity:** The interconnection of two or more data centers together is mandatory to aggregate unused resources and enable data and resource sharing. However, their design or connection capacities may vary, leading to difficulties to deploy applications across multiple Cloud environments without disruption.



**Security:** Security is essential to protect customers' privacy or sensitive data and code. To do so, several challenges have to be considered in Cloud computing such as authentication, confidentiality, and integrity of user's data and code.







**Portability/Interoperability:** Each Cloud provider has its way of providing the service to its customers, and may apply a wide range of different

proprietary APIs, leading to complexity and obstacles for interoperability.



**Cloud heterogeneity:** Cloud infrastructures are built upon heterogeneous resources to avoid vendors lock-in effect and due to frequent hardware updates. Resources reclamation need to be flexible to the storage and processing capacities.

Among the six challenges studied at b<>com, we specifically address in this thesis four challenges (*i.e.*,  users SLA guarantee,  resources volatility,  Cloud heterogeneity, and  security)

## 1.4 Problem Statement

Answering these challenges implies to define key issues or problems (see Figure 1.4):

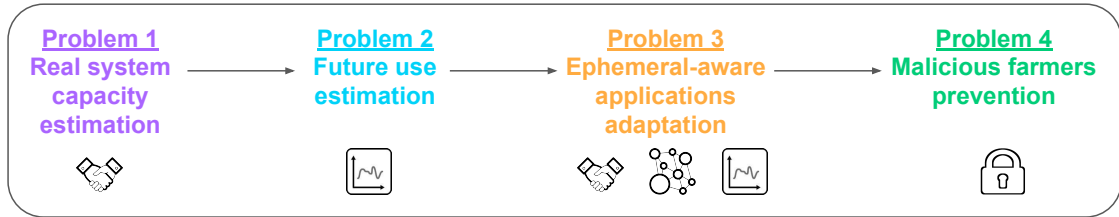


Figure 1.4: The problems addressed




**Problem 1 (Real system capacity estimation):** Guaranteeing users' SLA implies to estimate the maximum performance reachable by a system and determine the real system capacity. Several studies have underlined that co-located jobs may interfere and result in unwanted performance glitches. This may be due to some hardware, system mechanisms (*e.g.*, SSD, CPU, memory) or virtualization interference. Among the shared resources, I/Os are the main bottleneck [7]. Providing an accurate estimation of I/O  $Cap_{(t,mtr)}$  is critical for SLA guarantees and is a complex problem, but ***how to model performance variations?***

**Problem 2 (Future use estimation):** Once capacity is estimated, it is important to provide an accurate estimation of future amounts of used resources  $Used_{(t,mtr)}$ . However, in a context of high resources volatility, there is a need to mitigate the risk of inaccurate estimation. ***How can we estimate, in a flexible and accurate manner, future resources utilization?***

**Problem 3 (Ephemeral-aware applications adaptation):** Applications are designed and developed with the assumption that resources are available as long as users pay for the service. This assumption is not compatible with our challenges (*i.e.*, users SLA guarantee, resources volatility). Indeed, for guaranteeing that the running applications of *ephemeral customers* do not interfere with regular workloads of *regular customers* the allocated resource could be preempted. These ephemeral resources are an opportunity to process big data workloads at a lower cost since they require a considerable amount of computing resources. ***How big data applications can be adapted to run on ephemeral heterogeneous resources?***

**Problem 4 (Malicious farmers prevention):** While problems 1, 2 and 3 apply to all types of Cloud models, problem 4 addresses a specific issue faced in a Community Cloud environment. In such an open Cloud infrastructure, any farmer can join to provide/share his/her computation capacities. These farmers seek to reduce their TCO by (re)selling their unused computing resources. Allowing any farmer to join such platforms exposes an *Operator* to malicious behaviors. Malicious farmers can potentially produce erroneous or inaccurate results without effectively running the applications to obtain higher benefits from the Operator (*e.g.*, while saving their computation capacities) [155]. ***How can we prevent malicious infrastructure owners from compromising the computation?***


## 1.5 Thesis Contributions

In this thesis we claim that unused resources can be used to deploy applications at a low cost. Among the six challenges studied, we specifically address in this thesis four challenges (*i.e.*,  users SLA guarantee,  resources volatility, 

Cloud heterogeneity, and  security)

### 1.5.1 Determining system real capacity

To answer problem 1 (*Real system capacity estimation*), we designed an [SSD](#) performance models that take into account interactions between running processes/containers, operating system and [SSD](#). These interactions are complex. We investigated the use of machine learning for building such models in a container-based Cloud environment. We have investigated five popular machine learning algorithms along with six different I/O intensive applications and benchmarks. We analyzed the prediction accuracy, the learning curve, the feature importance and the training time of the tested algorithms on four different [SSD](#) models. Our machine learning-based framework succeeded in modeling I/O interference with a median [NRMSE](#) of 2.5%.

This contribution addresses  users SLA guarantee challenge. This work has been published in the journal IEEE Transaction on Cloud Computing 2019 [[49](#)].

### 1.5.2 Estimating future Cloud unused resources




To answer problem 2 (*Future use estimation*), we introduced a predictive model to determine the available resources and estimate their future use to provide availability guarantees. Our contribution proposes a technique that uses machine learning algorithms (*i.e.*, [RF](#), [GBDT](#), and [LSTM](#)) to forecast 24 hours of available resources at the host-level. One of the key contributions is the use of quantile regression to make our predictive model flexible for the [CP](#), rather than using the simple mean regression of resource usage. This makes it possible for a [CP](#) to make relevant and accurate trade-off between the volume of resources that can be leased and the risk in [SLA](#) violations. In addition, several metrics (*e.g.*, CPU, RAM, disk, network) were predicted to provide exhaustive availability guarantees. Our methodology was evaluated by relying on four in production data center traces and our results show that quantile regression is relevant to reclaim unused resources. Our approach may increase the amount of savings up to 20% compared to traditional approaches.

This contribution addresses  Resources volatility challenge. This work has

been published in the IEEE International Conference on Cloud Computing 2018 (CloudCom) [50].

### 1.5.3 Adapting applications to run efficiently on Cloud unused resources


To answer problem 3 (*Ephemeral-aware applications adaptation*), we designed a framework that leverages unused resources of data centers, which are ephemeral by nature, to run MapReduce jobs. Our approach allows: *i*) to run efficiently Hadoop jobs on top of heterogeneous Cloud resources, thanks to our data placement strategy, *ii*) to predict accurately the volatility of ephemeral resources, thanks to the quantile regression method (based on contribution in section 1.5.2), and *iii*) to avoid interferences between MapReduce jobs and co-resident workloads, thanks to our reactive QoS controller. We have extended Hadoop implementation with our framework and evaluated it with three different data center workloads. The experimental results show that our approach divides Hadoop job execution time by up to 7 when compared to the standard Hadoop implementation.

This contribution addresses:  users SLA guarantee,  resources volatility, and  Cloud heterogeneity challenges. This work has been published in the IEEE International Conference on Cloud 2019 (Cloud) [48].

### 1.5.4 Verifying the correctness of an execution in a trustless environment

To answer problem 4 (*Malicious farmers prevention*), we proposed an approach that allows sabotage detection in a trustless environment. To do so, we designed (1) a mechanism that builds an application fingerprint considering a large set of resources usage (*e.g.*, CPU, I/O, memory) in a trusted environment using random forest RF algorithm, and (2) an online remote fingerprint recognizer that monitors application execution and that makes it possible to detect unexpected application behavior. Our approach has been tested by building the fingerprint of 5 applications on trusted machines. When running these applications on untrusted machines (with either homogeneous, heterogeneous or unspecified hardware from the one that was used to build the model), the fingerprint recognizer was able

to ascertain whether the execution of the application is correct or not with a median accuracy of about 98% for heterogeneous hardware and about 40% for the unspecified one.

This contribution addresses the  security challenge. This work has been published in the IEEE International Conference: Modeling, Analysis, and Simulation Of Computer and Telecommunication Systems (MASCOTS) 2019 [47] and the results had been patented.

## 1.6 Outline

This thesis manuscript is composed of 9 chapters organized as follows:

---

### **Part I: Background and State of the Art**

---

Chapter 2 introduces Cloud computing with a focus on resource management as the work environment, and machine learning as a set of learning algorithms used in this thesis.

Chapter 3 discusses state of the art work on addressing the challenges of running applications on top of unused resources.

---

### **Part II: Contributions and Validations**

---

Our four contributions are presented in this part. Chapter 4 presents the overall solution that we want to defend in this thesis. Chapter 5 describes our contribution to estimate the maximum performance reachable by a system and determine the real system capacity. Chapter 6 presents a technique to estimate the future amount of used resources for each host to mitigate the impact of volatility. Chapter 7 presents an architecture that leverages unused but volatile Cloud resources to run big data jobs. Chapter 8 presents a technique for tracking the correctness of the application execution over time to prevent malicious infrastructure owners from sabotaging the computation. Chapter 9 provides additional information regarding the technical implementation of our solution.



---

**Part III: Conclusion and Perspectives**

---

Finally, we conclude with a summary of this thesis and give some directions for future work.

PART I

# **Background and State of the Art**

---

# BACKGROUND

---

This thesis deals with Cloud computing with a focus on efficient resource management. We thus start this chapter with an introduction to Cloud computing fundamentals to understand the main characteristics of Cloud computing services, their advantages, and their limitations. This chapter also introduces Machine Learning as a set of learning algorithms used in this thesis.

We first briefly discuss Cloud computing characteristics: Cloud and service models and quality of service. Specifically, we discuss virtualization technology and resource management. We then give an overview of two industrial Cloud computing solutions (*i.e.*, OpenStack and Kubernetes). Second, we introduce machine learning and its workflow, and explain its categories (*i.e.*, supervised learning, unsupervised learning, and reinforcement learning). We also describe six machine learning algorithms used in this thesis and give some elements about their configuration. Finally, we present a short overview of open source frameworks that aims to simplify the implementation of complex learning algorithms.

## 2.1 Cloud Computing

**Definition 2.1.1.** *According to the National Institute of Standards and Technology (NIST), Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [128].*

The concept of Cloud Computing started in the 1960s with the work done by McCarthy [71]. It was then popularized in the years 2006-2008 when IBM and Google announced a collaboration in the area [36]. Nowadays, Cloud computing

is used by many companies as it provides on-demand access to scalable, elastic, and reliable computing and storage resources with a pay-as-you-go pricing model.

### 2.1.1 Fundamentals

In this section, we introduce Cloud fundamentals, including Cloud models, service level agreement and service models.

#### Cloud Models

Four models can be identified: public, private, hybrid and community Cloud [61].

**Public Cloud:** resources are shared and rent to several customers (*i.e.*, individuals or organizations). Also, the infrastructure is owned and operated by a Cloud provider such as OVH <sup>1</sup> and can be reached over the Internet. From the customer point of view, public Cloud service allows to quickly access large amounts of computing resources with almost no setup costs. Moreover, these resources are provided with a high level of availability and reliability. However, public Cloud may not be appropriate for sensitive data or legal constraints (*e.g.*, health data exploitation).

**Private Cloud:** resources are managed and hosted by an organization. The Cloud users are the employees of the company which owns the infrastructure. The private Cloud enables a higher level of data security and confidentiality. In addition, the service is highly customizable to suit business needs. Moreover, according to a white paper of IDC [102], a private Cloud solution is 50% cheaper for highly predictable workload compared to public Cloud solutions.

**Hybrid Cloud:** resources are aggregated from different Clouds models. The hybrid Cloud provides the flexibility to increase the allocated compute resources by outsourcing the spikes of usage on the public Cloud. This outsourcing allows this solution to be cost-effective as organizations pay the spikes without the need to invest in a larger infrastructure. However, one

---

1. <https://ovhCloud.com/>

challenge is the interoperability and another is to combine different Cloud models while keeping a high level of efficiency [105].

**Community Cloud:** resources are combined from different organizations and different Cloud models (*i.e.*, Public, Private, or Hybrid). The community Cloud allows decreasing the initial investment of setting up the infrastructure by sharing the costs among all participants. However, trust and security are the main issues as a community Cloud is exposed to a higher risk of attacks (*e.g.*, malicious 'farmers' [*i.e.*, resource providers] could potentially join and damage the community).

### Service models

The services provided by Cloud computing can be divided mainly into five categories [119]: (i) On-premise, (ii) Infrastructure as a Service (**IaaS**), (iii) Container as a Service (**CaaS**), (iv) Platform as a Service (**PaaS**), and (v) Software as a Service (**SaaS**) (see Figure 2.1).

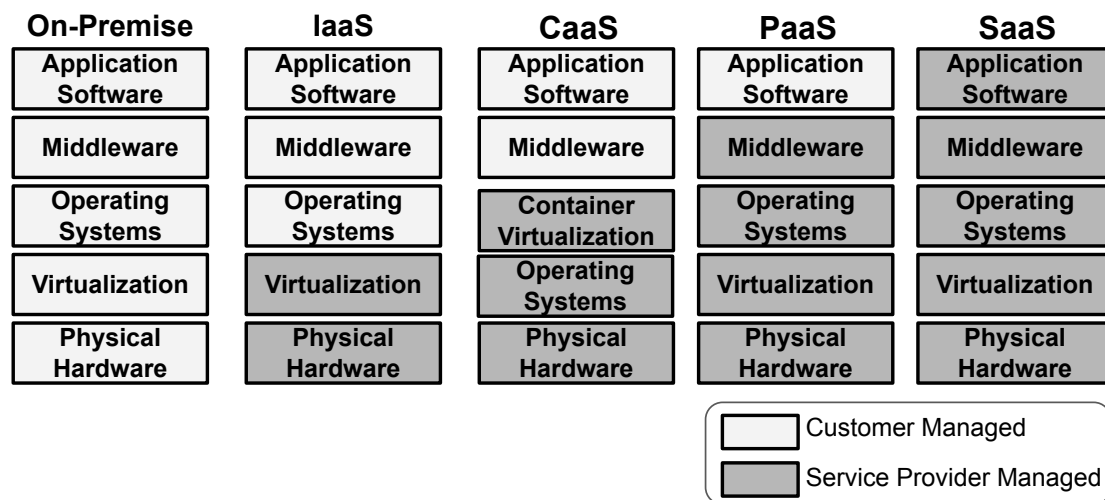


Figure 2.1: Cloud Computing service models

**On-premise** offers complete control on the infrastructure, security, scalability, and configurability. The infrastructure is hosted, managed, maintained in-house, and can be used to deploy a Private Cloud (see Section 2.1.1).

**Infrastructure as a Service (IaaS)** offers an API for provisioning and decommissioning physical and virtual hardware resources such as servers, network, and storage.

**Platform as a Service (PaaS)** offers managed operating systems and middleware. PaaS aims to simplify the application management by minimizing the interaction with the IaaS but also providing integrated features such as autoscaling and failure resiliency of the managed applications. Among the existing PaaS solutions, we can point out Apache Spark as service proposed by OVH<sup>2</sup> which aims to process big data without handling the complexity of the deployment and the configuration.

**Container as a Service (CaaS)** offers an easy way to deploy containers (see Section 2.1.2) on an elastic infrastructure with a fine container orchestration. CaaS service can be placed between the IaaS and the PaaS. However, most of the time CaaS is placed as a subset of IaaS.

**Software as a Service (SaaS)** offers applications on the shelf ready to use and optimized such as Overleaf<sup>3</sup>.

All Service models (*i.e.*, On-premise, IaaS, PaaS, CaaS, and SaaS) and Cloud models (*i.e.*, Public, Private, Hybrid, and Community Cloud) offer the possibility to design solutions for optimizing Cloud infrastructure.

There are many solutions for deploying IaaS and CaaS services including commercial solutions such as vCenter [172] or open source solutions such OpenStack [94] (see Section 2.1.4). More recently, Kubernetes [34] emerged as a CaaS management solution used by many companies (see Section 2.1.4). Cloud providers have adopted these solutions to reach higher levels of service quality for their customers. In the next section, the service-level agreement models proposed by Cloud providers are discussed.

## Service Level Agreement and Service Models

The Service Level Agreement (SLA) is a document that encompasses the terms of the contract negotiated between customers and Cloud providers. It specifies

---

2. <https://labs.ovh.com/analytics-data-compute>

3. <https://www.overleaf.com>

the expected Quality of Service (QoS) and defines the expected resource availability level and service constraints. Cloud providers classically offer two classes of Quality of Service Models according to [103]:

**Reserved Instance**<sup>4</sup>: resources are paid and available on a regular basis (monthly, annually). This type of service fits for users requiring resources on a long-term. Users have to make upfront payment but in return the availability of resources are guaranteed. Challenge for the user is to evaluate the amount of needed resources with a risk of over-provisioning.

**On-demand Instance**: for this model, no upfront payments are made and resources are booked on a minute or hourly basis. This flexibility provides users with the possibility to terminate any instance at any time and thus adapt the amount of resources to the demand. However, resource availability is not guaranteed. Note that, the reserved instances are cheaper (*i.e.*, 50% for OVH) compared to on-demand Instance.

When Cloud providers offer only Reserved or/and On-demand instances the available capacity is not fully utilized/optimized all the time. Figure 2.2 illustrate with CPU cores and memory the different types of unused resources, but they are not limited to these two metrics.

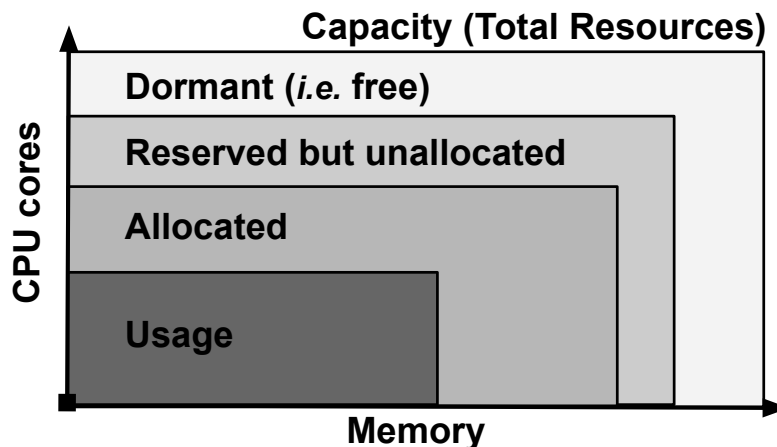


Figure 2.2: Cloud Unused Resources

---

4. An instance is a virtual server (*i.e.*, VM or Container) of a specific class of Quality of Service Model

- **Dormant:** capacity is free (*i.e.*, not assigned to any projects or customers) and can be directly used for handling future demand or/and failures.
- **Reserved:** capacity is reserved by users, but are not currently allocated to them (*i.e.*, free to spawn containers or virtual machines)
- **Allocated:** capacity is allocated, but users are not consuming all the allocated resources at a given time (*e.g.*, reclaim unused pages from running a container or virtual machine that is using only in average 20% of its 50GB allocated virtual memory).

Currently, some Cloud providers improve their resource utilization by reclaiming unused resources and offer an additional instance that we call '**Economy class**'. Economy instances are sold based on available unused computing capabilities. They are available at a significantly lower price than the on-demand and reserved instances, the drawback being that this type of instance can be interrupted at any time. This model enables Cloud providers to earn revenue from unused resources. Amazon spot Instance [18] and Google preemptible [77] are examples of this model (see state of the art chapter 3). The Economy class can be obtained when there are idle instances from Reserved and on-demand pools. It can be setup in all Cloud models (*i.e.*, Public, Private, Hybrid, and Community Cloud). The reliability of Economy instances and thus service quality is not guaranteed. Figure 2.3 summarizes the main characteristics of the three available service models.

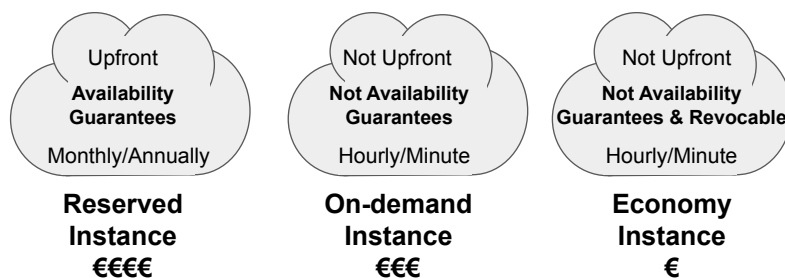


Figure 2.3: Service Models

Cloud providers (CPs) are relying on complex resource management systems (including virtualization technologies) to deliver efficiently these three classes of



service. **CPs** aim to find the best trade-off between customers satisfaction and profit maximization.

In the next section, we introduce the virtualization technologies which enable *isolation* and *abstraction* of processes, storage, and memory of a given physical machine (*i.e.*, a compute node) and we provide an overview of resources management solutions deployed by Cloud providers.

## 2.1.2 Infrastructure Virtualization

**Definition 2.1.2.** *Virtualization is a way to abstract applications and their underlying components away from hardware supporting them and present a logical or virtual view of these resources [116].*

In this section, we explain and compare the advantages and limitations of the two main techniques for virtualizing physical resources: Hardware virtualization and **OS** virtualization (see Figure 2.4). In this thesis, we decided to focus on **OS** virtualization because this virtualization is lightweight, boots very quickly and requires very few memory compared to hardware virtualization.

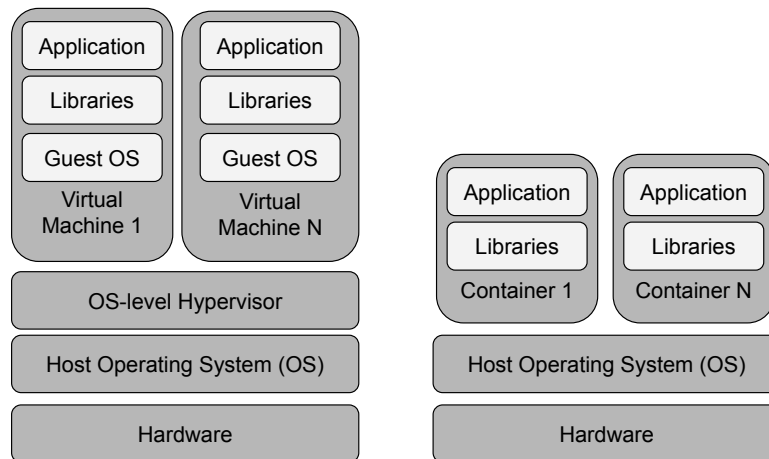


Figure 2.4: hardware-level virtualization (left) vs. operating system-level virtualization (right)

**Hardware virtualization** requires the use of a hypervisor also referred as a virtual machine monitor (VMM) that virtualizes physical server resources among multiple virtual machines. Each virtual machine (VM) has its own operating system and applications. VMs can run different operating systems isolated from the

physical host and from other VMs. The hypervisor is in charge of multiplexing the physical resources among the virtual machines. They are two types of hypervisors (*i.e.*, type 1 and type 2). A type 1 hypervisor runs directly on the physical machine without the need of an operating system. In contrast, a type 2 hypervisor is set up on top of an operating system. The hypervisor proposes several resource allocation policies (*e.g.*, best effort, shared, and guaranteed). KVM [117], VMware ESX [171] are examples of hardware virtualization solutions.

In comparison, OS virtualization applications run in isolation without relying on a separate operating system, thus saving large amounts of hardware resources. Indeed, resource reservation is managed at the operating system-level. Comparable to hardware virtualization several resource allocation policies are available. Containers are now widely used to modularize each application into a graph of distributed and isolated lightweight micro-services [157]. As a result, each micro-service is deployed within a container and has the illusion that it owns the physical resources, yet the system allows them to share objects (*e.g.*, files, pipes, resources). Docker [132] is generally used as a lightweight container system. It provides a common way to package and deploy micro-services [62]. Docker relies on two key technologies provided by the Linux kernel:

**cgroup**: it is a functionality that makes it possible to limit and prioritize resource usage (*e.g.*, CPU, block I/O, network) for each container without the need of starting any virtual machine [178]. For example, *cgroup* provides a specific I/O subsystem named *blkio*, which sets limits on and from block devices. Currently two I/O control policies are implemented in *cgroup* and available in Docker: (1) a Complete Fairness Queuing (CFQ) I/O scheduler for a proportional time-based division of disk throughput, and (2) a throttling policy used to bound the I/O rate for a given container. Also, the Linux traffic Control allows specifying different network-related parameters such as transmission rates, packs, scheduling, network policies, and traffic dropping.

**Namespaces**: Namespaces provide a layer of isolation between multiple users. In Linux, there are currently seven types of namespaces enabling isolation of Cgroup, IPC, Network, Mount, PID, User, and UTS [90].

**Challenges for performance Isolation:** In virtualized environments resources are shared and applications are potentially co-located on the physical host. The performance of containers and virtual machines depends on the type of co-located activities and has impacts on CPU, memory, disk, and network performances [152, 49]. Moreover, by default, the latest version of Docker which uses *cgroup* v1 only works on synchronous I/O traffic. As a consequence, *cgroup* v1 cannot properly limit the bandwidth of each container. This limitation is addressed in *cgroup* v2 but is not yet supported by Docker (*i.e.*, December 2019).

**Resource overcommitment to improve resource utilization:** In most virtualized environments, using available resources from physical machines and allocating them to VMs or containers is a routine task that can be performed dynamically. However, the task is more complex when resources have to be taken back from a running VM or a container to the physical machine. Indeed, to reclaim unused resources from running virtual machines or containers (see Figure 2.2) resource overcommitment is mandatory.

Resource overcommitment allows using more resources than the physical machine capacity can host. It allows improving resource utilization by combining potential complementary workload demands on the same physical machine. However, careful resource allocation has to be implemented in order to prevent severe performance degradation.

To mitigate performance degradation in an overcommitted system, several strategies can be deployed depending on the resource types (*e.g.*, CPU, memory), and on virtual virtualization techniques (*i.e.*, hardware-level and operating system-level virtualization).

The resource types can be classified into two categories: *compressible resources* and *incompressible resources*. *Compressible resources* such as CPU can be throttled, the user's applications will be slowed down proportionally to the throttling while keeping a normal execution. In contrast, *In-compressible resources* cannot be throttled without causing failure (*e.g.*, when the allocated memory is upper than the machine capacity).

Compared to containers, overcommitment techniques provided by hardware-level virtualization are more complex to implement. The operating system and the applications are indeed in most cases black boxes from the host operating sys-

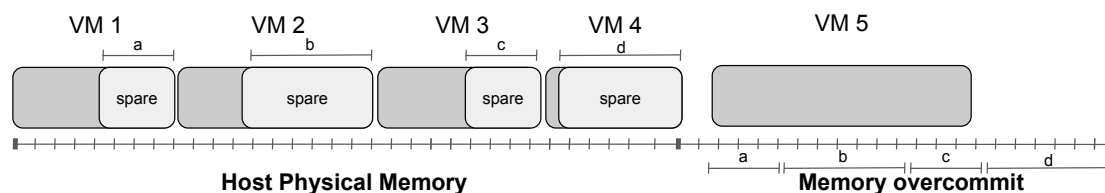


Figure 2.5: Hardware virtualization Memory overcommitment

tem point of view. Hardware-level virtualization memory overcommitment requires more advanced techniques such as memory hotplug/unplug, memory ballooning, or hypervisor paging [12].

Figure 2.5 shows an example of memory ballooning reclamation technique that allows a host to retrieve unused memory from certain running VMs. First, the hypervisor has to determine where the fragmented spare memory (*i.e.*, a, b, c, d in our case) are. Then, this fragmented spare memory have to be redistributed to VMs that need more memory or to new VMs (*e.g.*, VM 5). To achieve that, the memory ballooning technique requires a collaboration of the user's VMs (*i.e.*, an agent has to run inside the VM). Memory ballooning has to keep enough host physical memory to provide to all virtual machines guest physical memory to prevent any virtual machine from running out of host physical memory. In contrast, for operating system-level virtualization, the memory is shared by design in the same manner as regular hosted applications.

**Discussion:** Hardware-level and operating system-level virtualization techniques can be used for providing a standard way for repackaging and reselling physical server unused resources.

Operating system-level virtualization has less performance overhead compared to hardware-level virtualization. However, a drawback of OS virtualization technique is that their attack surface is larger compared to hardware-level virtualization. In most commercial deployments, Cloud providers are using hardware-level virtualization for running untrusted or potentially malicious applications. Kata Containers and Google gVisor propose two approaches seeking to find a trade-off between container performance and security. Using OS virtualization overcommitment allows reusing built-in techniques of the operating system such as memory soft/hard limit or kernel memory. Specifically, soft-limit allows a container to easily

recycle unused memory, but this container has to be destroyed when the memory owner need them to avoid performance degradation.

In contrast, virtual machines are more secure and propose more mature technologies for managing resources with some drawbacks on performance isolation for storage. These conclusions are summarized in Table 2.1.2.

	Hardware virtualization	OS virtualization
Operating System dependency	no	yes
Over-commitment	yes, but complex techniques for memory	yes
Security	mature security models	not mature and complex
Performance overhead	high	low
Performance isolation	mature with some lack on I/O	not mature
Typical boot-time	minutes	seconds

Table 2.1: Comparison between hardware virtualization and OS virtualization

In conclusion, virtualization is essential for Cloud resource management. It enables to smartly share processor, memory, network and storage and thus allows a better utilization of resources. Virtualization is also a key technology to perform resource optimization. In the next section, we will explain how efficient resource management (*i.e.*, monitoring, allocation, and provision) is operated in a virtualized environment.

### 2.1.3 Cloud Resource Management

**Definition 2.1.3.** *Resource Management is the process of allocating computing, storage, networking to a set of applications in a manner that seeks to jointly meet the performance objectives of applications, the infrastructure (*i.e.*, data centers) providers and the users of Cloud resources [98]*

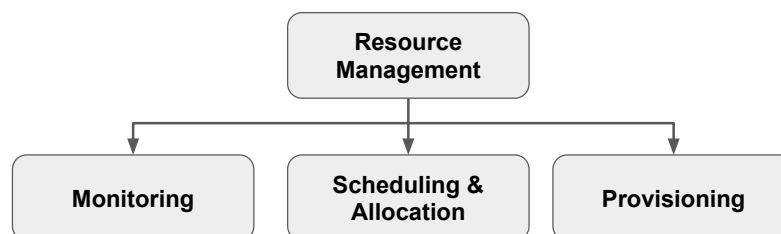


Figure 2.6: Resource Management

Managing resources in order to improve their utilization and reduce costs is a major concern for Cloud providers [98]. A Cloud provider seeks to minimize its operating costs while fully satisfying QoS negotiated with customers, *i.e.*, maximizing the utilization of the physical servers and their components (*i.e.*, CPU, memory, and storage). Figure 2.6 presents an overview of Cloud resource management components.

**Monitoring:** Implementing resource management policies requires a constant monitoring on the status of hardware resources (*e.g.*, CPU or memory usage, virtual machine or container) [2]. These status are commonly stored in time series to allow CPs to manage their infrastructure platform(s), virtual machines, containers and track issues such as performance bottleneck or hardware failures [2]. There are numerous solutions for monitoring Cloud services including commercial offerings such as Dynatrace <sup>5</sup> and open source such as cAdvisor <sup>6</sup>.

**Scheduling/Allocation:** The scheduling process determines where and when a node shall be created in the infrastructure to fulfill the consumer needs and Cloud provider constraints. The scheduling gives the capacity to make intelligent placement of workloads according to a given goal (*e.g.*, reduce the number of servers used, minimize energy consumption, reduce data center operating costs) [25]. In order to adapt to the user's demands, it is necessary to migrate the virtual resources among servers. The challenge is to dynamically decide the mapping of the VMs or containers' among the servers. Indeed, the migration of VMs can introduce runtime overheads and consume more energy, leading to a risk of SLAs violations. Also, combined workloads can generate interferences thus impacting SLA guarantees. Finally, the resource allocation affects the selected resource to the job or task of user's request.

**Provisioning:** The proposed allocation is executed on the real system by calling the adequate APIs of the used infrastructure manager, such as Kubernetes.

Cloud infrastructure is constantly evolving with new deployed/updated/added hardware, software, and configurations. This environment is also highly dynamic

---

5. <https://www.dynatrace.com/technologies/Kubernetes-monitoring/>

6. <https://github.com/google/cadvisor>

where workload changes and failure may occur suddenly. The growing complexity of these systems leads to the necessity to automatically and constantly adapt the infrastructure to ensure efficiency and quality of service.

Autonomic computing reference architecture introduced in 2001 by IBM [89] could be used to cope with these challenges. Autonomic computing aims at making computer systems able to self-manage. In this section, the MAPE-K (Monitor-Analyze-Plan-Execute-Knowledge) [106] loop which is extensively used as a reference architecture for Cloud computing resource management and optimization [137] is presented. For example, the optimization service of OpenStack Watcher implements the MAPE-K control loop model.

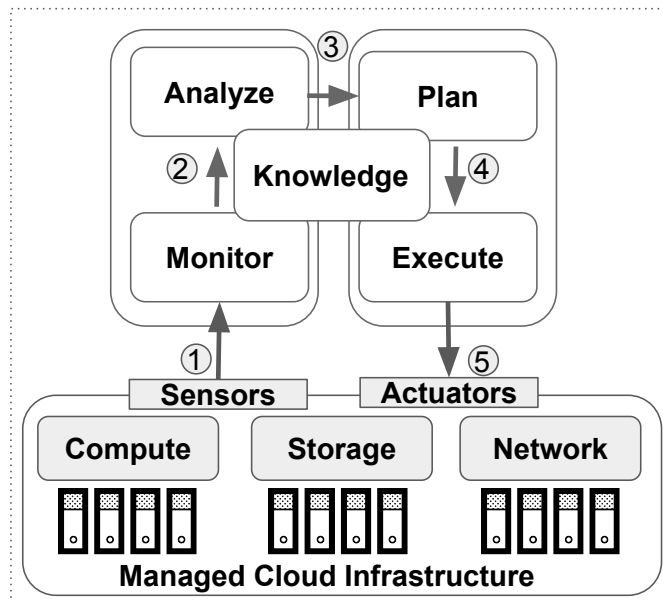


Figure 2.7: MAPE-K Management

The *MAPE-K* loop is composed of five main components depicted in Figure 2.7:

1. **Monitor**: Collect real-time metrics and topology of a data center (i.e., compute, storage, network, etc.)
2. **Analyze**: Perform analysis on collected data and detect workflow patterns. This step is influenced by the stored knowledge data. If changes are required, the Plan function should be triggered.
3. **Plan**: Define a list of actions to be performed on the cluster resources.

4. **Execute:** The proposed container placement is scheduled and executed on the real system by calling the corresponding APIs, such as Kubernetes [87].
5. **Knowledge:** Data is stored and shared at each step.

## 2.1.4 Cloud Infrastructure Management solutions

In the next section, two industrial Cloud infrastructure management solutions in production are presented. First, we introduce OpenStack to describe a complete Open-Source solution for building an IaaS and then Kubernetes is given as an example of a CaaS.

### OpenStack

OpenStack is a software designed to manage large pools of compute, storage and network resources [94]. It can be used for creating a private or a public Cloud (see Section 2.1.1). OpenStack started in July 2010 as a joint project between Rackspace and the NASA to develop an Open-Source IaaS. The first version of OpenStack was made up of two services: the *nova* service in charge of managing virtual machines mainly based on the code of the *Nebula* project and the *swift* service for the storage which is based on the Rackspace Cloud File Platform project.

OpenStack is composed of several loosely coupled components allowing modular deployments. Each component is in charge of a specific functionality (*e.g.*, compute or network) for operating an IaaS. At a minimum level, OpenStack requires the installation of the following services *Nova*, *Keystone*, *Glance*, and *Cinder*. Since the Stein release (*i.e.*, 2019-04-10), the placement service also has to be installed. In total, about 67 official components are supported by the OpenStack Technical Committee such as *Magnum* (Container Infrastructure Management service) and *Watcher* (Infrastructure Optimization service).

Figure 2.8 shows an overview of the nine most deployed OpenStack components:

**Horizon:** provides a graphical user interface that allows end users and administrators to manage OpenStack resources such as network, access controls, or virtual machines;



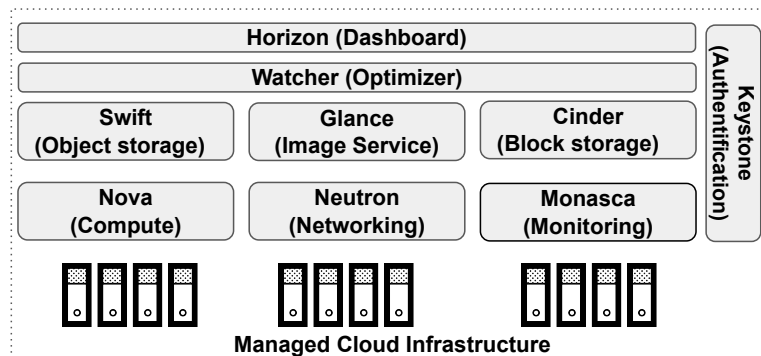


Figure 2.8: Architectural overview of OpenStack

**Watcher:** provides a robust framework to implement a wide range of Cloud optimization goals, including reduced data centers operating costs and improved system performance <sup>7</sup>;

**Swift:** provides an object storage management <sup>8</sup>;

**Glance:** allows to discover, register, and retrieval of virtual machine images <sup>9</sup>;

**Cinder:** provides a persistent block storage management <sup>10</sup>;

**Nova:** manages and automates all steps necessary to provision computing instances <sup>11</sup>;

**Neutron:** manages networks and IP between computing instances <sup>12</sup>;

**Monasca:** provides a monitoring and logging the status of application and hardware resources that can be used for example for billing or alerts <sup>13</sup>;

**Magnum:** manages containers orchestration engines having a distinctly different life cycle and operations than Nova <sup>14</sup>;

**Keystone:** allows to authenticate the OpenStack users'. The service is used by all OpenStack services. The authentication could be based on credentials,

---

7. <https://github.com/openstack/watcher>

8. <https://github.com/openstack/swift>

9. <https://github.com/openstack/glance>

10. <https://github.com/openstack/cinder>

11. <https://github.com/openstack/nova>

12. <https://github.com/openstack/neutron>

13. <https://github.com/monasca/>

14. <https://github.com/openstack/magnum/>

token-based, or LDAP <sup>15</sup>.

## Kubernetes

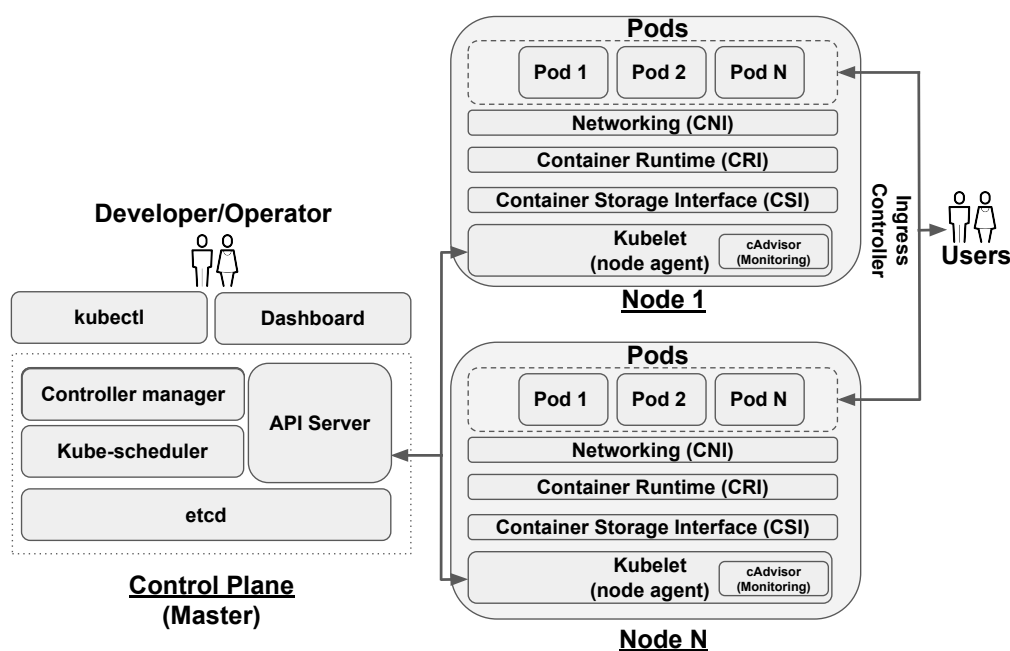


Figure 2.9: Architectural overview of Kubernetes

Kubernetes <sup>16</sup>, also referred as *k8s*, offers an easy way to automate and deploy containers on an elastic infrastructure with a container orchestration for automating application deployment, scaling, and management [87]. Kubernetes started in 2014 by three engineers McLuckie, Joe Beda and Brendan Burns that wanted to recreate *Borg* [173] and *Omega* [150] as open source projects. *Borg* is a cluster manager that runs services such as Google Search, Gmail or Google Maps. *Omega* is a flexible, scalable scheduler for large compute clusters.

Figure 2.9 shows an overview of Kubernetes architecture. Kubernetes uses a control plane (*i.e.*, master) and a distributed reliable key-value store for keeping the cluster state consistent (*i.e.*, etcd <sup>17</sup>), and a number of cluster nodes for providing the compute resources (*i.e.*, nodes).

15. <https://github.com/openstack/keystone/>

16. (κυβερνήτης, Greek for "governor", "helmsman" or "captain")

17. <https://github.com/etcd-io/etcd>

The control plane is mainly composed of three components: *API Server*, *Controller Manager*, *Kube-Scheduler* (see Figure 2.9). The control plane has to run on at least one node but it can be replicated for providing fault tolerance.

**API Server** is the entry point of the Kubernetes cluster management system.

The *API Server* supports the authentication and authorization of k8s. It also manages the orchestration life-cycle (*e.g.*, scaling up or down) of the hosted applications. The *API Server* is used by the *kubectl* (*i.e.*, the command-line interface) and dashboard (*i.e.*, graphical user interface ) to manage Kubernetes resources.

**Controller Manager** is a daemon that embeds the core control loops. It monitors the current state of the cluster and applies the changes to reach the desired state (*e.g.*, scaling an application up or down, adjusting endpoints).

**Kube-scheduler** is tracking the available capacity of each node (*i.e.*, host). Also, it determines the node where a pod (*i.e.*, container) shall be created in the infrastructure in order to fulfill the consumer needs and constraints (*e.g.*, resource limitations, affinity and anti-affinity).

In most k8s deployments, the cluster nodes are composed of five elements: the kubelet and its embedded monitoring cAdvisor, Container Runtime Interface (CRI), Container Network Interface (CNI), Container Storage Interface (CSI), and the pods (see Figure 2.9).

A pod is the smallest deployable unit. A pod represents a single instance of a running process. However, a pod can contain one or several containers when these containers are highly coupled (*e.g.*, same IPC namespace, or shared volume). There are various types of pods in k8s (*e.g.*, *ReplicaSet*, *Deployment*, *StatefulSet*, *Daemonset*) with various objectives. For example, a *Daemonset* implies that each node of the cluster will run an instance of a pod.

k8s works with a wide range of containerization technologies and network solutions. To achieve that without recompilation, the CRI, CNI, and CSI are plug-gable interfaces that enable to change easily the underlying implementations. Docker <sup>18</sup> or rkt <sup>19</sup> are examples of CSI, flannel <sup>20</sup> a CNI, and CephFS <sup>21</sup> a CSI.

---

18. <https://github.com/docker/docker-ce>

19. <https://github.com/rkt/rkt>

20. <https://github.com/coreos/flannel>

21. <https://github.com/ceph/ceph-csi>

The *kubelet* is a key service deployed on each node. The kubelet is the k8s agent that is in charge of implementing the interface between the nodes and the cluster logic. The *kubelet* is also embedding cAdvisor. cAdvisor collects measurements related to resource consumption such as CPU, memory, network or I/O. The kubelet manages the container runtime (e.g., Docker) and checks that the defined pods are created, healthy, or stopped when necessary. The kubelet also calls the CNI to create network interfaces for the new containers.

Kubernetes provides a way to divide cluster resources between multiple teams, or projects using the *Namespaces* concept. k8s proposes by default three Quality of Service for containers (i.e., Guaranteed, Burstable, and Best-Effort). When Kubernetes creates a container, it assigns one of these QoS classes. Note that Kubernetes makes a clear distinction between *Compressible resources* that can be throttled (i.e., the applications is slowed down proportionally to the throttling, but will otherwise proceed normally), and *Incompressible resources* that cannot be throttled without causing failure (e.g., for memory). k8s works with a wide range of containerization technologies such as Docker [132] or rkt [143]. Finally, k8s is able to monitor failed pods and restart them automatically.

Kubernetes and OpenStack are complementary technologies that can be combined. Kubernetes is a tool tailored for fine container orchestration technologies. In contrast, OpenStack is a framework for deploying a complete IaaS. Thus, OpenStack provides a complete multi-tenancy implementation which is not the case of Kubernetes. In addition, Kubernetes can be hosted within OpenStack virtual machine in order to benefit from the strong security of virtual machines [152]. However, running Kubernetes clusters on bare-metal servers appears to improve performance [152].

The modular development of solutions allows the integration of resource optimization strategies. Resource optimization is often provided by dedicated modules, allowing flexibility to adapt these solutions in order to integrate the contributions of this thesis.

## 2.1.5 Discussion

This thesis focuses on one solution enabling efficient available resources re-use in the Cloud context. The reclamation of the Cloud unused resources can be

achieved at all levels of the Cloud stack (*i.e.*, IaaS, PaaS, SaaS, and CaaS). However, in this thesis we focus mainly on IaaS and CaaS levels as they are directly controlling the physical resources (*i.e.*, compute, network and storage), thus enabling the design of efficient strategies to recycle available resources.

Cloud unused resource reclamation techniques can be applied to all Cloud models. However, when resources are shared with different organizations' security and legal considerations are essential. Indeed, when an organization is reclaiming its resources for its own needs within a private Cloud, risks are lower compared to a community Cloud where sensitive data or code could be exposed to potential attackers. Besides, in a community Cloud, legal issues have to be taken into account. As an example, an organization could be prosecuted due to network traffic which initiated racial hatred due to a third party using its resources. We focus on OS virtualization because they have lower overheads in terms of compute resources compared to virtual machines.

## 2.2 An introduction to Machine Learning

Machine learning investigates automatic techniques to make accurate predictions based on past observations [10]. Datasets contain a set of attributes called features, used to build a prediction model for some specific output response metrics. I/O access patterns (random/sequential) and operation types (read/write) are examples of features while the throughput is the output response. Datasets can be either quantitative (*e.g.*, throughput) or categorical (*e.g.*, spam/not spam). The general questions that ML can answers are: How our Cloud infrastructure is really used? Why and when our Cloud has malfunctioned? Which components should be replaced? Can we predict that the machine will break down next week? Which parts need to be improved?

There are three different categories in machine learning: supervised, unsupervised and reinforcement learning. In *supervised learning*, the algorithm uses features and their corresponding response values in order to model relationships between features and responses. It includes two types of problems, *Classification*: for categorical response values (*e.g.*, an email is spam or not), and *Regression*: for continuous-response values (*e.g.*, I/O throughput). In *unsupervised learning*, the algorithm only relies on the input features as the corresponding responses

are not available. The goal is to let the learning algorithm find by itself how data are organized or clustered. In *reinforcement learning*, the algorithm interacts dynamically with its environment in order to reach a given goal such as driving a vehicle.

Choosing the right learning algorithm for a specific problem is a challenging issue. Many state of the art studies such as [63] have discussed the way to select the appropriate learning algorithm(s) depending on the datasets and the type of problem to solve. Classical algorithms such as *linear discriminant analysis* and *nearest neighbor techniques* have been criticized on some grounds [31]. For instance, they cannot handle categorical variables and missing data. Other algorithms such as *support vector machines* (SVM) depend on the careful selection of hyperparameters and the implementation details [86]. Neural networks suffer from higher computational burden, proneness to over-fitting, the empirical nature of the model development, and the fact that they are hard to debug [170]. In [85], the authors described characteristics of different learning algorithms that we have summarized in Table 2.2, where we extracted a list of seven algorithms used in this thesis.

Table 2.2: Some characteristics of the learning methods used [85].

Characteristic	Key: ▲= good, ○=fair, and ▼=poor.						
	DT	MARS	AdaBoost	GBDT	RF	SVM	NN
Robustness to outliers in input space	▲	▼	○	▲	▲	▼	▼
Handling of missing values	▲	▲	▲	▲	▲	▼	▼
Computational complexity	▲	▲	▼	▼	▼	▼	▼
Prediction accuracy	▼	○	▲	▲	▲	▲	▲

The accuracy of a model depends strongly on the dataset and the learning algorithm used. It also depends on the algorithm tuning parameters, called hyperparameters. These parameters impact the complexity of the learning model, and they are selected so as to minimize the error. For convenience, we used the following notation:

- Inputs (features)  $x_i$  ( $i = 1, 2, \dots, n$ ) is a vector (where  $n$  is the total number of data samples available,
- Responses  $y_i$  ( $i = 1, 2, \dots, n$ ) is the output response.

In this thesis, we used two methods in order to configure the hyperparameters. The first one consists of simply using the configuration recommended by the

authors (of the algorithm) when available. The second one consists of using the  $K$ -fold cross-validation method [113]. Then we chose the configuration giving the best prediction. The idea of the  $K$ -fold cross-validation method is to divide the training set into  $K$  roughly equal-sized parts. For the  $k^{th}$  ( $k = 1, \dots, K$ ) part, we fit the model to the other  $K - 1$  parts and calculate the prediction error of the fitted model. We combine the  $K$  estimates of prediction error.  $\hat{f}^{-k}$  denotes the fitted model (for  $k = 1, \dots, K$ ), the cross-validation estimate of the error is:

$$CV(\hat{f}^{-k}) = \frac{1}{K} \sum_{k=1}^K \sum_{i \in k^{th} \text{ part}} (y_i - \hat{f}^{-k}(x_i))^2 \quad (2.1)$$

So, the hyperparameters of the model  $\hat{f}^{-k}$  are estimated such as to minimize the criterion (2.1). We used for our simulations  $K = 5$  which is the value recommended in [30].

## 2.2.1 Learning algorithms

In this section, we will describe each of the seven machine learning algorithms used in this thesis and give some elements about hyperparameters configuration.

**Decision trees (DT)** This method was developed at the University of Michigan by Morgan et al. in the early 1960s and 1970s ([65, 133]). *DT* partitions the feature space into a set of rectangles, and then fits a simple model in each one. In this thesis, we used the method CART (Classification and Regression Trees) [31] which is a popular method in decision trees. It encodes a set of *if-then-else* rules in a binary tree which are used to predict output variables given the data features. These *if-then-else* rules are created using the training data which aim to maximize the data separation based on a loss function related to classification or regression scores.

CART method can be evaluated as a linear combination of the indicator function of sub-regions  $R_m$  that form a partition of the feature space:

$$f(x) = \sum_{m=1}^M c_m I(x \in R_m), \quad (2.2)$$

where  $I$  is the indicator function having 1 for  $x$  of  $R_m$  and 0 for  $x$  not in

$R_m$ . The weights  $c_m$  and the regions  $R_m$  are learned from data in order to minimize the loss function.  $M$  is the maximum depth of the tree. We increase  $M$  such that the nodes are expanded until all leaves contain less than a certain minimum number of samples. The  $DT$  is composed of two main stages, creating a tree to learn from all the training samples and then pruning it to remove sections that are non-significant variables that would decrease the accuracy.

**Multivariate adaptive regression splines (MARS)** This method was introduced in 1991 by Friedman [68]. To approximate nonlinear relationship between the input features and the response values [69]. To achieve that, MARS is using piecewise linear basis functions of the form  $\max(0, x-t)$  and  $\max(0, t-x)$  as shown in example of Figure 2.10. Each function is piecewise linear, with a knot at the value  $t$  also called linear splines [69].

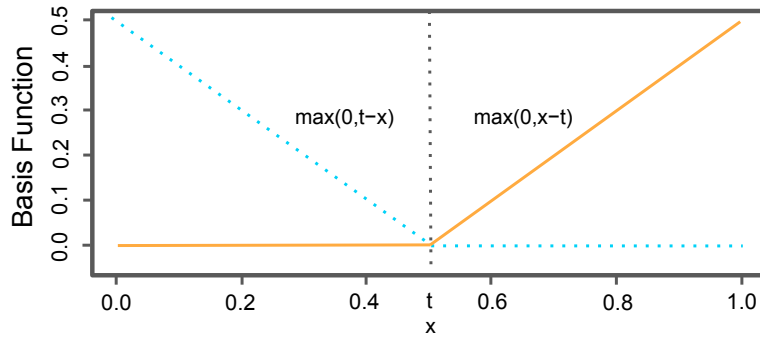


Figure 2.10: The basis functions  $\max(0, x-t)$  and  $\max(0, t-x)$  used by MARS

MARS model has the following form:

$$f(x) = c_0 + \sum_{m=1}^M c_m h_m(x), \quad (2.3)$$

where  $h_m(x)$  ( $m = 1, \dots, M$ ) takes one of the following two forms:

- a spline function that has the form  $\max(0, x_i - t)$  or  $\max(0, t - x_i)$ , where  $x_i$  is a feature and  $t$  is an observation of  $x_i$ . MARS automatically selects  $x_i$  and  $t$ .
- a product of two or more spline functions.



To build the model (in Equation 2.3), we have two main phases: First, the forward phase is performed on the training set, starting initially with  $c_0$ . Then, for each stage, the basis pair which minimizes the training error is added to a set  $\mathcal{M}$ . Considering a current model with  $m$  basis functions, the next pair added to the model has the form:

$$c_{m+1}h_\ell(x) \max(0, x_i - t) + c_{m+2}h_\ell(x) \max(0, t - x_i),$$

where  $h_\ell \in \mathcal{M}$ . Each  $c_m$  is estimated by the least-squares method. This process of adding basis functions continues until the model reaches the maximum number ( $M$ ) fixed.

Finally, the backward phase improves the model by removing the less significant terms until it finds the best sub-model. Model subsets are compared using the less computationally expensive method of Generalized Cross-Validation ( $GCV$ ). This criterion is defined as:

$$GCV = \frac{1}{1 - \frac{r+0.5d(r+1)}{N}} \sum_{i=1}^N (y_i - \hat{f}(x_i)) \quad (2.4)$$

in which  $r$  is the number of basis functions,  $d$  is a penalty for each basis function included in the developed sub-model,  $N$  is the number of training datasets, and  $\hat{f}(x_i)$  denotes the MARS predicted values.

**Random Forests** (RF), introduced in [29], they enhance decision trees by building a large collection of de-correlated trees, and then averaging them. RF are a combination of CART (Classification and Regression Trees) models, which are binary trees, such that each model depends on the values of a random vector sampled from training data independently with the same distribution for all trees in the forest. In *CART*, the split aims to maximize the accuracy score by splitting the training data with the best feature on each node of the trees. RF are very accurate and their hyperparameters are simple to tune [82].

RF has three main hyper-parameters: the number of trees  $T$  and the following hyper-parameters for each tree:

- $m$  : the number of features to consider when looking for the best split

- $n_{\min}$  : the minimum number of samples required to split an internal node

### Gradient Boosting Decision Trees (GBDT)

The main idea of glsgbdt is to train iteratively a decision tree such that the ensemble of these decision trees may be more accurate than any decision tree. In this thesis, we used *GBDT* proposed by Friedman [67]. GBDT has three main hyperparameters:

- $M$  : the number of regression tree models
- $\ell$  : the size of trees
- $\nu$  : the learning rate

**Boosting Method - Adaboost** The basic idea of these methods is that they combine the outputs of many "weak" estimator into a single estimator that, hopefully, will be much more accurate than any of the "weak" ones. A weak estimator is one whose error rate is only slightly better than random guessing. Freund and Schapire [66] proposed the most popular boosting algorithm for a binary classification problem which is called *AdaBoost.M1*. Zhu et al. [194] extended this algorithm to the multi-class case without reducing it to multiple two-class problems. Drucker [56] extended the *AdaBoost* algorithm to regression problems which is the algorithm used in our thesis. In AdaBoost, the weak learners are decision trees with a single split, called decision stumps. The glsadaboost model for regression (adaboost.R) has the form:

$$f(x) = \text{weighted median}\{h_t(x), t = 1, \dots, T\} \quad (2.5)$$

where  $h_t$  is a weak regression algorithm. adaboost.R uses multiple iterations to produce a stronger rule, the weights adjust themselves to improve the estimator performance. The algorithm scales the contribution of each regressor by a factor  $0 \leq \nu \leq 1$  called the learning rate. There is a trade-off between the number of weak regression machine and the learning rate.

**Long Short Term Memory (LSTM)** Recurrent Neural networks (RNN) [20] were designed to capture dependencies within an input sequence and not only a single feature vector compared to glsrf and glsgbdt. To achieve that, RNN uses hidden states that act as internal memory to keep information about

previous inputs. In this way, RNN are useful for capturing temporal dependencies by tracing previous information.

Traditional RNN suffers from vanishing or exploding problem during the back-propagation of the gradient weights on long sequences [20]. In [88], the authors have proposed glslstm to address this issue.

## 2.2.2 Machine learning workflow

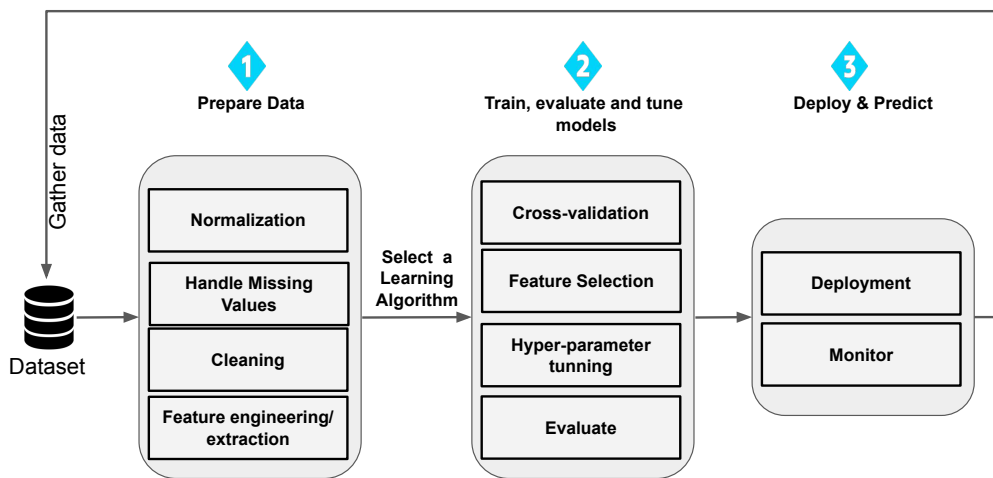


Figure 2.11: Cloud Computing service models

A challenge in model building is to have representative data to build an accurate prediction. To gather data, three choices are available: relying on existing data, generating data (*e.g.*, by executing workloads) or using both. In both cases, the dataset needs to be representative of some real usages, otherwise this model will face biases. Figure 2.11 shows an overview of the three main stages in a machine learning workflow:

1. **Prepare data:** First, the goal of this stage is to prepare the data for the learning stage. To achieve that, many steps can be performed to suit the needs of the learning algorithms such as standardization/normalization, handling of missing value, cleaning. Also, the data could be transformed for improving the performance (*i.e.*, accuracy) of the machine learning models also known as feature engineering.

2. **Train, evaluate, tune models:** Then, one needs to train the learning algorithm on the prepared data, select the most relevant features, tune the hyper-parameters and compare the results of the model predictions to the real values.
3. **Deploy/Predict:** Finally, the prediction has to be accessible to the end-users. This involves exposing and serializing the information that represent the trained model. On the medium-long-term, we need to monitor the prediction results to verify its accuracy, and when necessary consider refining the model.

### 2.2.3 Machine learning frameworks and libraries

Building machine learning models is a challenge with many stages and steps. Some of these tasks can be handled by machine learning frameworks and libraries. In this section, we present two open-source frameworks that we used in this thesis to help us build machine learning models.

**Scikit-learn** is a general machine learning library built on top of many Python libraries such as SciPy, or NumPy [139]. Scikit-learn includes many machine learning algorithms such as support vector machines, random forests, as well as tools for data pre/post-processing.

**Tensorflow/Keras** is an open-source library developed by Google. TensorFlow is a library for numerical computation using data flow graphs. Keras is a high-level API built on Tensorflow which makes the code simpler and clearer. It includes a library dedicated to deep learning, and more specifically to neural networks models (*e.g.*, LSTM).

## 2.3 Summary

This chapter introduced the knowledge and a set of methods used in this thesis. We summarized Cloud computing main concepts (*e.g.*, service and Cloud models, virtualization) and Cloud resource optimization and management. The MAPE-K model and its five steps and two industrial solutions (*i.e.*, OpenStack and Kubernetes). We also presented an overview of machine learning. In the next chapter, we will review state of the art related to our contributions.

# STATE OF THE ART

---

This state of the art chapter provides an overview of a wide range of studies having all in common the development and provisioning of tools and methods towards leveraging Cloud unused resources and deploying applications at a cheaper price while achieving [SLA](#).

We start this chapter by introducing the overall approaches and solutions for leveraging unused resources. We then introduce state of art for each problem covered in this thesis.

## Overall approach

The idea of recycling computer unused resources for research projects or for selling low-cost Cloud resources has been studied extensively in the scientific literature. Early research attempted to exploit idle workstations for parallel computation [3]. In the mid-1990s, volunteer computing platforms have been developed for providing resources to research projects that require huge amount of processing capabilities [13]. In these platforms the computer owners (*i.e.*, volunteer desktops, laptops, and mobile phones) donate their spare computing resources. Then, with the growth of Cloud computing services, some state of the art studies took advantage of Cloud unused resources in a reactive manner by leasing them with limited SLA guarantee. Other studies focused on detecting or resizing idle resources with the aim to make them available for reuse [135, 191].

In [126], authors make available any underutilized resource in an opportunistic way to improve resource utilization. Others proposed to mitigate the impact of volatility using fault-tolerance techniques [184, 179]. Since 2009, most Cloud providers (*i.e.*, Amazon, Google, Microsoft) are selling their dormant and unallocated resources in the form of an economy class to their customers, with limited

SLA. On the contrary, some proposed to use predictive models in order to achieve SLA [41] by forecasting mainly the CPU. In [41] authors have proposed to claim unused Cloud capacities to offer a cheaper class (*i.e.*, limited SLA) with long-term availability by forecasting available resources for the next 6 months. This led to a benefit increase of 60% for Cloud providers [39].

## Platforms available for reclaiming unused resources

This section presents two volunteer computing platforms and four commercial solutions to enable the utilization of unused resources.

### Volunteer computing

**SETI@home** is a scientific experiment started in 1999 in radio astronomy that uses unused processing capacities of millions of computers connected via Internet for discovering extraterrestrial intelligence. The SETI@home distributed computing software operates mainly as a screensaver [14].

These solutions are based on volunteer desktops and include no SLA-related guarantee, or a minimum duration of the execution.

### Commercial solutions that are reclaiming Cloud unused resources

**Amazon EC2 Spot Instances:** In 2009, Amazon started EC2 Spot instances. EC2 Spot instances sell Amazon EC2 spare compute capabilities at a significantly lower price than the on-demand ones. Compared to on-demand resources, spot instances can be interrupted by Amazon after a 2-minute notification (*i.e.*, when Amazon needs them for their regular customers). The spot prices are updated according to the supply and demand of available EC2 compute capabilities and are specific to different regions and availability zones. Amazon also offers EC2 fleet solution that orchestrates and manages spot instances along with on-demand resources. Amazon spot Instance may save up to 90% off the on-demand price but without SLA guarantee [18].

**Google Preemptible VMs:** In 2015, Google started Google Preemptible VMs, a similar solution to Amazon EC2 Spot Instances. Google solution is, however, limited to 24 hours, the price is fixed and the notification is sent only 30 seconds before the instance is shutdown.

**Azure batch VMs:** In 2017, Microsoft launched a similar solution called Azure batch with two offers: an 80% discount on Linux Low Priority VMs and a 60% discount on Windows instances. The price is also fixed.

**Spotinst Elastigroup:** Spotinst Elastigroup offers a solution that saves up to 90% of costs on compute infrastructure on top of all major Cloud providers but with SLA guarantees [4]. To achieve that, Spotinst uses machine learning to predict several metrics (*i.e.*, capacity trends, pricing, and interruptions rate). Elastigroup is able to prevent interruptions by predicting them and smartly migrating the allocated resources. In case there is no available spare capacity, the instance about to be interrupted can use on-demand instances to ensure SLA.

Table 3 summarizes available commercial solutions. Most of the solutions are not providing SLA apart from Elastigroup which uses predictive models and fall-back to on-demand instances. None of the available solutions reclaims allocated but unused resources (see Figure 2.2) that requires resources over-commit (see Background 2.1.2)

	EC2 Spot Instances	Azure bath VMs	Google Preemptible VMs	Elastigroup
Pricing	Variable	Fixed	Fixed	Variable
Notification	2 minutes	30 seconds	30 seconds	15 minutes
Time limit	None	4 hour	24 hour limit	6 hour
Revocability	When underbid	Higher priority	Higher priority	No, Fallback on-demand
Reclaimed Resources	Dormant/Unallocated	Dormant/Unallocated	Dormant/Unallocated	Dormant/Unallocated
SLA	No	No	No	Yes

Table 3.1: Summary of economy class solutions

## Problems addressed in this thesis

We focused in this thesis on four key problems (see Introduction chapter). Figure 3.1 recalls the problems addressed in this thesis.



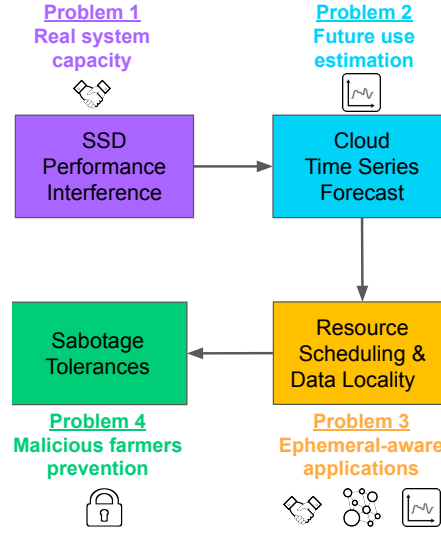


Figure 3.1: A map of problems and associated approaches

**Problem 1 (SSD and performance interference):** For addressing problem 1, we study state of the art work addressing interference in virtualized environments, especially SSD interferences. SSD storage devices are indeed massively adopted for their higher performance and low energy consumption [27]. Interference has an impact on real system capacities over time, and thus on SLA guarantees for both regular and ephemeral customers.

**Problem 2 (Cloud time series forecast):** After estimating the real capacity, we study research work investigating how to accurately estimate future used resources. The goal is to maximize the leasing of unused resources which, in turn, will maximize potential cost savings for the CP.

**Problem 3 (Resource scheduling and data locality):** In order to unleash all the benefits of Cloud unused resources, applications must be adapted to be ephemeral-aware. In this chapter, we study work in this direction and especially, the ones investigating big data workloads processing since they require a considerable amount of computing resources.

**Problem 4 (Sabotage tolerance mechanisms):** Finally, when applications are running efficiently on ephemeral resources, it is of utmost importance to study security issues. In this chapter, we are interested in studies conducted to provide secure remote computation.

### 3.1 Performance modeling and I/O interference

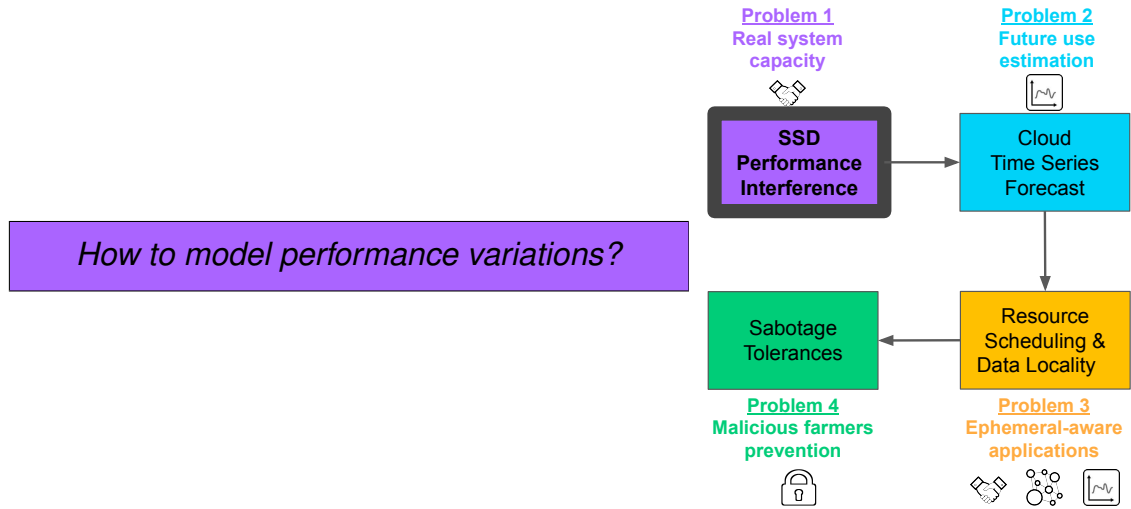


Figure 3.2: Problem 1 (Real system capacity estimation)

Efficiently sharing resources is challenging in order to guarantee SLA. Several studies have shown that among the shared resources I/Os are the main bottleneck [7]. As a consequence, Solid State Drives (SSDs) were massively adopted in Cloud infrastructure to provide better performance. However, they suffer from high performance variations due to their internals and/or the applied workloads (Problem 1, see Figure 3.2).

In this section, we present studies proposed in the literature for modeling I/O interference on SSDs. The first step is to get a performance model that can capture I/O interference. Another approach is to modify the system behavior or the SSD behavior to limit the risk of interference. But first, we present an overview of SSD internals and performance.

**A brief overview of SSD internals and performance** Flash memory is structured hierarchically: a chip is composed of one or more dies, each die is divided into multiple planes which are composed of a fixed number of blocks, each of which encloses a fixed number of pages (see Figure 3.3). Current versions of flash memories have between 128 KB and 2048 KB blocks (with pages of 2, 4, or 8 KB) [27]. A page consists of a data space and a metadata Out-Of-Band (OOB) area used to store page state, information on Error Correction Code (ECC), etc.

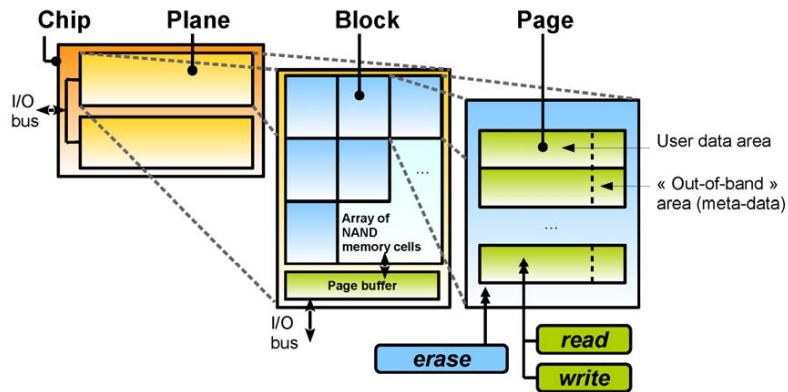


Figure 3.3: Simplified architecture of a NAND flash memory chip from [27]

Three operations can be carried out in flash memory: read and write, which are realized on pages, and erasures, which are performed on blocks.

The main flash memory constraints that affect SSD internal mechanism design is the erase-before-write rule and the limited number of erase cycles a flash memory cell can sustain [27]. In effect, a page cannot be updated without prior erase operation. Data updates are performed out-of-place with a *mapping scheme* to keep track of data position. These mapping schemes are different from one SSD to another and may induce large performance differences. Out-of-place updates also make it necessary to have garbage collection (GC) mechanisms to recycle previously invalidated pages. GC also has a great impact on performance, especially in case of bursts of random writes as those operations delay applications I/O request completion. On the other hand, the limited lifetime of flash memory cells makes it crucial to use *wear leveling* techniques. In addition, SSDs make use of *parallelism* within flash chips/dies/planes through advanced commands in order to maximize the throughput.

The complexity of SSD architectures and their wide design space have two major impacts with respect to performance. First, the performance may vary dramatically from one SSD to another, and second, for a given SSD, performance also varies according to the interaction of a given I/O workload, with other workloads, with system-related mechanisms, and with SSD internal mechanisms. These variations may induce a significant impact on SLA.

**Flash-based storage devices** Common performance modeling studies have targeted hard drives using analytic modeling [145, 24, 153], simulation [109, 33], benchmarking [169, 6], and black-box approaches [187, 174]. Many analytic and simulation approaches were based on understanding internal organization of storage devices. However, the internal design employed by SSDs are often closely guarded intellectual properties [70]. To overcome this issue, black box approaches have been used [187, 174, 91]. In [91], Huang *et al.* proposed a black box modeling approach to analyze and evaluate SSD performance, including latency, bandwidth, and throughput.

**Improving system behavior to limit the I/O interference** To better predict the SSD behavior, some state of the art studies have tried to tackle this problem at different levels, mainly at low-level SSD controller and system level. The first class of solutions tries to implement some low-level techniques to minimize the interference at the flash memory chip level, for instance by physically storing container data in specific flash chips [108]. Myoungsoo et al. [101] proposed to create a host interface that redistributes the GC overheads across non-critical I/O requests. The second class of solutions operates at the system level, Sungyong Ahn et al [7] have modified the Linux cgroup I/O throttling policy by assigning an I/O budget that takes into account the utilization history of each container during a specific time window. The third class of solutions proposes an application-based solution. In [55] and [114] the authors propose to avoid I/O interference by coordinating the applications I/O requests. Finally, Noorshames et al [136] present an approach for using machine learning to capture I/O interference.

## Discussion

Many studies have been conducted to tackle I/O interference issues [147]. These solutions are mainly preventive and are designed at different levels. At the device level, the authors of [108, 101] have proposed optimizations related to SSD algorithms and structure such as isolating VMs on different chips. Unfortunately, to the best of our knowledge, this type of SSD is not commercialized and no standard implementation is proposed. At the system level, some studies [159, 7] have attempted to modify the I/O scheduler and the Linux cgroup I/O throttling

policy. Nevertheless, these optimizations are not standard enough and are not supported by sufficient kernels to allow for a simple usage. Finally, at the application level, in [136], the authors focused on HDDs and did not consider SSDs and their specific I/O interferences. These conclusions are summarized in Table 3.1.

References	SSD	low-level	system-level	application-level	container-based
[136, 55]	No	No	No	<b>Yes</b>	No
[108, 101]	<b>Yes</b>	<b>Yes</b>	No	No	No
[7]	<b>Yes</b>	No	<b>Yes</b>	No	No
<b>Targeted solution</b>	<b>Yes</b>	No	No	<b>Yes</b>	<b>Yes</b>

Table 3.2: Summary of performance modeling and I/O interference

## 3.2 Cloud time series forecast strategies

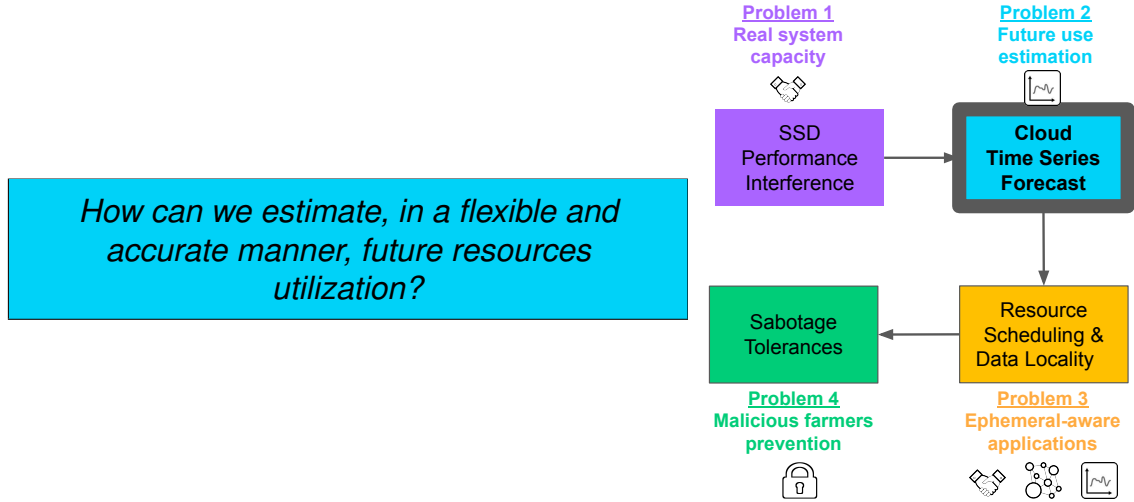


Figure 3.4: Problem 2 (Future use estimation)

In this section, we focus on state of art work that are looking to provide an accurate estimation of the future used Cloud resources (*i.e.*, Problem 2, see Figure 3.4).

Many studies such as [11] have discussed about how to select the appropriate learning algorithm(s) to forecast time series with learning algorithms such as Autoregressive (AR), Integrated Moving Average (ARIMA) and other more complex algorithms such as RNN, SVM, LSTM, and RF.

In [54] authors used *AR*, *MA*, *ARMA* and *ARIMA* models and other variants to forecast the average load from 1 to 30 seconds in the future. The time series built in *AR*, *MA* models assumes that models are stationary processes. This means that the mean of the series of these models and the covariance among its observations do not change over time. In case the time series is non-stationary a transformation to a stationary series has to be performed first. The main drawbacks of these models is the poor accuracy compared to machine learning methods [125].

Yang *et al.* [181] proposed several homeostatic and tendency-based one-step-ahead forecasting methods. The idea of homeostatic strategy is that the future CPU value will retain the mean CPU value, *i.e* if the current CPU value is greater (lower) than the mean of the history then the next value will likely decrease (increase). The tendency-based method forecasts the future CPU value under the assumption that the pattern is stable which is not our case. Beghdad *et al.* [21] proposed to use neuro-fuzzy and Bayesian inferences for the problem of CPU load forecasting. Gmach *et al.* [73] studied the workload analysis for enterprise data center applications. In this case, the workload analysis demonstrates the burstiness and repetitive nature of enterprise workloads. The workload demand pattern is decided by using a Fourier transformation then they classified workloads according to their periodic behavior using k means clustering algorithm. Finally, they generated synthetic traces to represent the future behavior of workloads. Song *et al.* [158] applied *LSTM* to forecast the mean host load in data centers of Google and other traditional distributed systems. They compared *LSTM* method with the following methods: *AR* model [176], artificial neural networks (*ANN*) [58], Bayesian model [53], the *PSR+EA-GMDH* method [182] and the echo state networks (*ESN*) [183]. They have shown that their method achieves state of the art performance with higher accuracy in both data centers. Kumar *et al.* [115] used *LSTM* networks to build a workload forecasting model. They showed that the accuracy of their forecasting model has reduced the mean square error up to  $3.17 \times 10^{-3}$ . Islam *et al.* [93] considered the case of resource provisioning in the Cloud. They used neural networks, linear regression algorithms and a sliding window technique. Their approach supposes a linear fashion of the workload pattern which is not our case. Indeed, applications deployed in data centers may generate nonlinear workloads such as unpredictable workloads [120].

## Discussion

In a Cloud context predicting the future resource demand is a challenging task due to many variables that may influence the resource usage such as the users, the number of applications, virtual machines or containers. The most important characteristics of demand prediction for reclaiming Cloud unused resources are the following:

- **Granularity:** The level at which the estimation is performed (*e.g.*, data center, cluster, or host level). The finer is the granularity, the more complex is the estimation, and the better the usability of the model.
- **Flexibility:** The estimation needs to be flexible enough to give the CP the opportunity to find the best trade-off between the amount of resources to reclaim and the risk of SLA violations.
- **Exhaustivity:** An estimation should be based on several resource metrics to achieve SLA requirements for reclaiming the maximum amount of unused resources. For example, if there are free CPU resources without available memory, this may lead to SLA violations.
- **Robustness:** estimations should be robust to workload changes as deployed workloads have vastly different runtime characteristics [151].
- **Applicability:** estimation techniques should not have high overheads in terms of time and computing resource requirements as compared to the potential reclaimable resources.

Most state of the art resource prediction studies [11] have focused on the estimation of the mean load. This makes those solutions poorly **flexible** as the mean load highlights only one aspect of the distribution of a variable (*e.g.*, CPU) without considering peak values that may cause SLA violations.

In addition, most studies [11] do not rely on an **exhaustive** set of metrics. Indeed, they are mostly based on two resource metrics: CPU and/or RAM consumption. Unfortunately, relying on one or two single metrics in a data center is not realistic since applications often require multiple computing resources. In effect, network and storage resources play a major role in SLA violation avoidance [11]. The authors of [41] provided an interesting investigation about forecasting

the unused capacity in order to provide SLA over spare resources. Even though they provide a **robust** model, the **granularity** was too large. The cluster level (*i.e.*, aggregation of all hosts) was chosen. As it was detailed in the introduction chapter (see Section 1.2), the resource usage distribution among hosts in a given data center is not homogeneous, which makes it hard to design a scheduler strategy able to deploy applications among the hosts given the cluster level spare resource prediction. The authors have focused on the forecast of aggregated CPU consumption (*i.e.*, all hosts in the cluster). This does not provide the required level of granularity.

These conclusions are summarized in Table 3.2.

References	Granularity (Host-Level)	Flexibility	Exhaustivity	Robustness	Applicability
[41]	No	No	No	<b>Yes</b>	<b>Yes</b>
[158, 115]	<b>Yes</b>	No	No	<b>Yes</b>	<b>Yes</b>
[54, 21, 73, 176, 53, 93]	Yes	No	No	No	No
<b>Targeted solution</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>

Table 3.3: Summary of Cloud time series forecast strategies

### 3.3 Improving Hadoop efficiency in volatile and heterogeneous Cloud environments

In this section, we discuss some studies that have already explored MapReduce in dynamic and heterogeneous environments (see Figure 3.5).

Many studies have already shown that heterogeneity significantly impacts Hadoop performance. In [188] authors proposed to improve data locality by delaying the task scheduling by a small amount of time for short jobs, which significantly improved performance. In [189] authors proposed to prevent from incorrect execution of speculative tasks by defining a fixed threshold in which the scheduler is able to select tasks to speculate. Their approach has improved the response time of MapReduce jobs by a factor of 2 in large heterogeneous clusters. In both cases they have improved execution time of jobs in heterogeneous environment.

Some approaches considered the case of an open shared system and proposed a placement strategy that dispatches data chunks to hosts based on their availability rate [100]. The drawback of such approaches is their assumption about



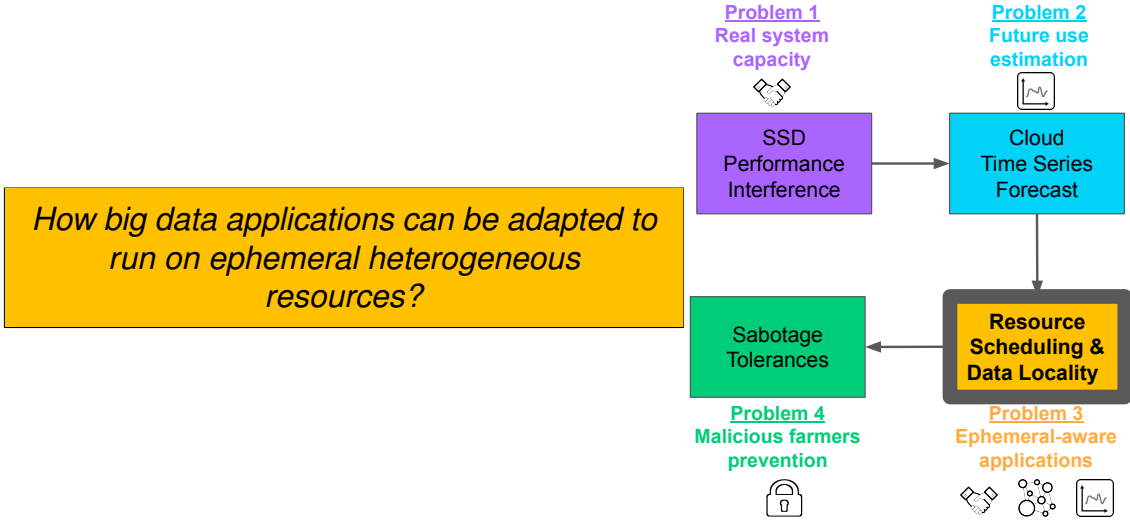


Figure 3.5: Ephemeral-aware applications adaptation

the inter-arrival time of interruptions (*i.e.*, due to node failure or volatility) to be independent and identically distributed across the system. There are some studies that handled volatility in open systems by using a small set of dedicated nodes in order to ensure the minimum amount of resources required to execute MapReduce jobs. In [121] authors have proposed hybrid approach to handle resources volatility. They used a small set of dedicated nodes in order to ensure the minimum amount of resources required in order to efficiently execute MapReduce jobs. While we target unused resources of Cloud data centers connected through the internet (*i.e.*, low bandwidth and high latency), the authors' targeted environment is composed of personal computers connected through a local network (*i.e.*, high bandwidth and low latency). In the same way [184] uses reserved resources alongside ephemeral ones to increase the reliability of executed tasks.

In [162] authors considered the case of running Hadoop application on pervasive grids which volatility represents the main challenge to overcome. In their work they use *collector* module that monitors nodes capacities only in terms of the number of processors and memory capacity in order to feed Hadoop scheduler. This information is used to make decisions and adapt resources allocation. In [45], authors have investigated the use of volatile spot instances as accelerators in public Clouds without considering any SLA constraints, while In [16] authors propose the use of hybrid infrastructures (*i.e.*, a mix of public or/and pri-

vate Cloud with volunteer computing) for big data applications processing. Their work has shown that hybrid infrastructures can provide operational continuity in environments with up to 25% of unstable nodes without loss of performance.

Finally, some studies used a predictive approach to improve data locality and Hadoop performance. In [131] authors propose a predictive scheduler with data prefetching to improve the data locality in MapReduce Cluster. It uses a linear regression to predict the execution time of map tasks. This prediction is used for pre-fetching the input data into nodes that will execute the respective tasks. In their case the predictor observes the execution of previous map tasks paying attention to two parameters: data input size and number of simultaneous map tasks. They assume that, there is a linear relation between the two previous parameters and the execution time. Therefore, the predictor trains a linear regression model to determine the correlation between them, which is used to schedule future tasks. In [156] authors propose a discrete event simulator for estimating MapReduce job execution time. The simulator allows one to vary the input data size, the type and number of machines in the cluster and uses linear regression to predict the job execution time. Their goal is to help organizations to better plan their budget with data analysis. Another approach [164] proposes to use the Naïve-Bayes-classifier method to predict node availability and thus improve map-reduce scheduler performance. A timeout-based approach, inspired by the classic binning scheme, is used to measure the availability of nodes and generate traces that are used to predict future availability. Worker nodes with low availability and stability stop accepting data chunks, which pushes the master node to distribute input data to more stable nodes.

## Discussion

There have been prior work to customize Hadoop so that it can run more efficiently on volatile and heterogeneous environments. Some studies only considered heterogeneity and did not consider the volatility of nodes. Others assumed an inter-arrival of interruption (*e.g.*, preempted by priority tasks) that needs to be dependent and identically distributed, which is not always the case. In contrast, other studies require a minimum amount of reserved resources to minimize the impact of volatility, which one does not necessarily have. Some focused on local

network environments while most Cloud data centers are connected through the internet (*i.e.*, low bandwidth and high latency). Some used a predictive model to provide SLA but their approaches suppose a linear workload pattern which is not our case. Moreover, available solutions are not providing the flexibility needed. Finally, there is no solution that simultaneously considers heterogeneity, volatility, users SLA guarantee, and data locality. Besides that, there is no approach that guarantees SLA without interfering with other workloads sharing the same physical resources. These conclusions are summarized in Table 3.3.

References	Heterogeneity	Volatility	Users SLA guarantee	Data Locality
[184]	No	<b>Yes</b>	No	No
[192]	No	<b>Yes</b>	No	<b>Yes</b>
[179, 121]	<b>Yes</b>	<b>Yes</b>	No	No
[15]	<b>Yes</b>	No	No	<b>Yes</b>
[189]	<b>Yes</b>	No	No	No
<b>Targeted solution</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>

Table 3.4: Summary of opportunistic mapreduce on ephemeral and heterogeneous Cloud resources

### 3.4 Sabotage-tolerance mechanisms

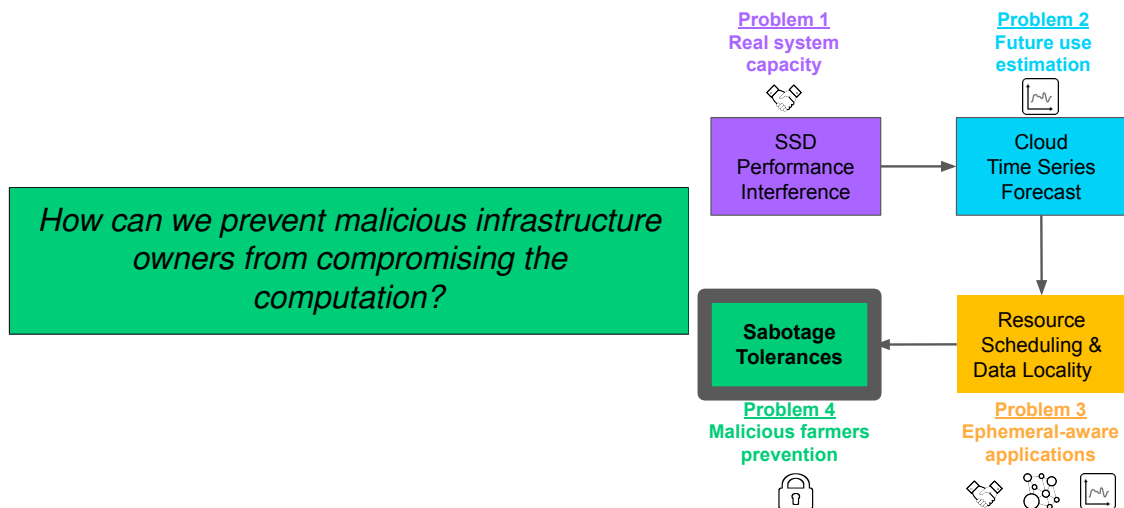


Figure 3.6: Problem 4 (Malicious farmers prevention)

Numerous studies have already evaluated the correctness of an execution in a trustless environment (Problem 4, see Figure 3.6). In this section, we first focus on the work that tries to validate the computation results. Then, we discuss studies that try to ascertain the correct execution of applications.

### **Ensuring the result correctness and detecting sabotage**

In [148], the objective is to execute the same work unit  $N$  times before eventually comparing them - each result being a vote - until converging towards a result. This method has the advantage of ensuring a very high level of certainty with regards to the correctness of the result of a given work unit. However, this comes with two major drawbacks. First, it has a high overhead:  $N$  times the initial execution cost with  $N$  being the number of votes. Second, the time this method requires to ascertain the falseness of a given execution can be excessively long as many rounds of voting may occur, postponing the decision every time.

In [57, 193], a different strategy was used. Contrary to the previous method, the goal here is to minimize the overhead of checking the correctness of an execution by submitting the various resource providers to test. The kinds of tests that can be used may rely on four different techniques: naive, quiz, ringers, and spot-checking (see [148, 75]).

### **Application identification and detection**

The fingerprinting approach tries to generalize application behavior given its execution traces into a model. This approach has been used in different contexts to identify and detect applications. In [5, 186], the authors show the benefits of fingerprints to automatically detect hardware Trojan and in [122], Lin *et al.* used packet size distribution of the connections to create an application fingerprint.

Side-channel analysis is composed of two steps, commonly referred to identification and exploitation. The identification consists in understanding the leakage and building suitable models. The exploitation consists of using the identified leakage models to extract an information. To build the model, several approaches have shown that it can be approximated in a profiling phase using machine learning techniques. In [80], Gulmezoglu *et al.* show that the use of cache access profiles could be efficient to classify applications. Zender *et al.* [190] show the ef-

efficiency of unsupervised machine learning to automatically classify network traffic and application. In [149], Schuster *et al.* show that by monitoring an encrypted network traffic, a convolutional neural network can accurately distinguish movies streamed using network traffic bursts. Unfortunately, the assumption of relying on a single metric (*i.e.*, network) and one type of application is not adapted for Cloud environments where numerous applications can be deployed.

## Discussion

Many studies have been conducted to provide secure remote computation [17, 148]. Most of the traditional approaches such as replication voting, ringers, and spot checking - whether with or without blacklisting - have a high overhead on the compute resources (it may double the used resources) to verify each application execution or require a dedicated hardware such as Intel SGX with 60% of the native throughput and about 2x increase of the application code size [17]. For reselling Cloud unused resources efficiently, we are looking for the following properties:

- **Backward compatibility:** proving a non-invasive/non-intrusive solution on the application code and not limited to a type of application or hardware.
- **Online execution:** proving a continuous verification of the correct execution of the application.
- **Efficiency:** proving a small overhead to verify each application execution

These conclusions are summarized in Table 3.4.

References	Backward compatibility	Online execution	Efficiency
[17]	No	No	<b>Yes</b>
[148]	<b>Yes</b>	No	No
[57, 193, 75]	No	No	No
[190, 149, 80]	No	<b>Yes</b>	No
<b>Targeted solution</b>	<b>Yes</b>	<b>Yes</b>	<b>Yes</b>

Table 3.5: Summary of sabotage-tolerance mechanisms

### 3.5 Summary

In this part, we introduced various studies having all in common the development and provisioning of tools and methods toward leveraging Cloud unused resources and deploying applications at a cheaper price while achieving [SLA](#).

First, we presented several approaches targeting problem 1 (*real system capacity estimation*) for robust capacity estimation. Then, we showed available strategies for forecasting unused resources related to problem 2 (*future use estimation*). After that, we discussed the work done for running Hadoop efficiently on top of ephemeral resources with [SLA](#) guarantees, *i.e.*, problem 3 *ephemeral-aware applications adaptation*. Finally, we studied the work to ensure the execution of an application in a trustless environment *i.e.*, problem 4 *malicious farmers prevention*.

These studies have improved the real capacity assessment, but also efficient applications running in ephemeral Cloud environments, and finally security. However, the available studies suffer from some limitations:

- **problem 1 (real system capacity estimation):** While the first class of solutions (*i.e.*, improve low-level [SSD](#) implementations) is hardly usable for Cloud providers using off-the-shelf [SSDs](#), the second (*i.e.*, improve system-level implementations) could be cumbersome to implement efficient solutions that fit for different devices. Moreover, these approaches have focused on [HDDs](#) and did not consider [SSDs](#) and their specific I/O interferences. They also did not investigate on containers while these are increasingly used by [CPs](#). The lack of solutions that handle [SSD](#) specific challenges that can be used independently from low-level optimizations makes it difficult to deploy applications on top of unused resources while achieving [SLA](#).
- **problem 2 (future use estimation):** Studies related to predicting the future use have focused on the estimation of the mean load. This makes those solutions poorly flexible as the mean load highlights only one aspect of the distribution of a variable (*e.g.*, CPU). In addition, most studies do not rely on an exhaustive set of metrics. These problems may lead to [SLA](#) violations.
- **problem 3 (ephemeral-aware applications adaptation):** Some state of the art studies require a minimum amount of reserved resources to minimize

the impact of volatility, which one does not necessarily have. Other studies did not consider simultaneously heterogeneity and volatility. Finally, most of state-of-art work did not provide methods or tools to avoid interference with the provider's regular customers' workloads which makes it difficult to deploy applications with [SLA](#) guarantees.

- **problem 4 (malicious farmers prevention):** Finally, many studies have been conducted to provide secure remote applications execution. However, most of the approaches have a high overhead on the compute resources (*e.g.*, it may double the used resources) or require dedicated hardware such as Intel SGX. Finally, these studies are not able to continuously verify the correct execution of the application which could lead to [SLA](#) violations.

This chapter described the main research efforts to leveraging Cloud unused resources and deploying applications at a cheaper price while achieving [SLA](#). One of the ultimate goals of the thesis is studying the deployment of applications in a Cloud made of volatile resources and heterogeneous hardware while providing security, and guarantee performance for ephemeral customers while avoiding interference on regular customers. In the next chapters, we describe in detail our contributions to achieve these goals.

PART II

# **Contributions and Validations**

---



# PHD OVERVIEW

---

In this chapter, we present the overall solution that we want to defend in this thesis. We claim that Cloud unused resources can be utilized to deploy applications at a low cost without compromising on system quality of service and security for both regular and ephemeral customers. We also claim that among the three types of Cloud unused resources the allocated but underutilized resources type is the most relevant for optimizing Cloud infrastructures (see Section 2.1.1). Indeed, it enables to reclaim the whole Cloud unused resources (*i.e.*, allocated but underutilized, unallocated, and dormant). We believe that OS virtualization provides a standard and lightweight way for (re)using Cloud unused resources (see Section 2.1.2). We also believe that resources overcommitment (see Section 2.1.2) is mandatory for reclaiming allocated but underutilized resources. We propose to combine three approaches:

First, we think that machine learning algorithms can accurately predict the real system capacity (**Problem 1**) and future resources availability (**Problem 2**) for providing SLA guarantees. Moreover, resources estimation has to be done at the host level which provides an accurate overview on available machines. This approach then helps to design strategies to deploy applications among the hosts based on cluster level spare resources.

Second, applications need to be adapted to run efficiently on ephemeral heterogeneous resources through specific smart placement (*e.g.*, data locality, allocation and scheduling) and fault-tolerance mechanisms. An applications' catalog with such adaptations should be made available to the customers. Also, we need to introduce a new class of QoS dedicated to Economy instances. This class would enable to prevent any interference on regular workloads and support the development of a strategy that automatically reclaims unused resources for allocating them to ephemeral customers work-

loads. This class would support flexible allocation policies (e.g., cgroups see Background) to limit and prioritize resources usage (e.g., CPU, block I/O, network, etc.) for each container. It needs to be combined with a safety margin to manage unpredictable workloads and avoid interferences between the ephemeral customers' applications and regular customers' workloads (**Problem 3**).

Finally, we defend in this thesis that by analyzing and characterizing a set of metrics, we would be able to create predictive fingerprint recognition models that enable verification on the correct execution of the requested applications on remote untrusted machines (**Problem 4**).

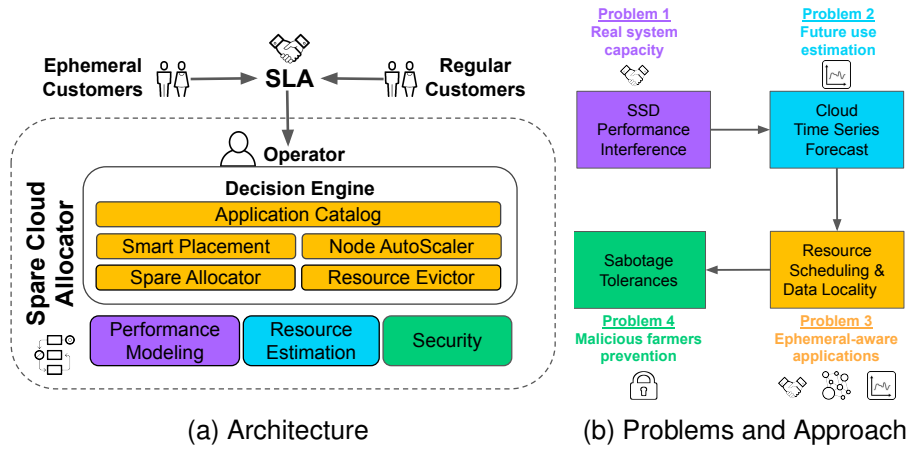


Figure 4.1: PhD Overview

An overview of the solution is depicted in Figure 4.1. Figure 4.1a shows an overview of our proposed architecture to leverage Cloud unused resources to deploy applications with SLA guarantees. Figure 4.1b shows the link, by the use of similar colors, between associated problems and the architecture. The architecture relies on four main modules:

**Decision Engine (DE)** is responsible for handling customer requests which consist in executing the designated applications while guaranteeing SLA for ephemeral and regular customers. This module has four sub-modules:

- The **Application Catalog** is the user's gateway to the available applications that are ephemeral-aware.

- The **Spare allocator** is a module that assigns and shares Cloud unused resources between applications of *ephemeral customers*. This module also handles how the cluster responds to the evicted resources.
- The **Resource Evictor** implements a mechanism that reacts to under-estimation of used resources from the *Resource Estimation module*.
- The **Smart Placement** controls all applications ephemeral placement and scaling activities (e.g., data locality, scheduling).
- The **Node Autoscaler** implements a mechanism that replace any lost nodes automatically and maintain the desired amount of nodes.

**Performance Modeling** this component builds a realistic system maximum performance model for each node.

**Resource Estimation** this component aims to predict resource volatility.

**Security** This component is in charge of the security of ephemeral customers workloads by using a fingerprint recognizer that is able to ascertain the correct execution of the requested applications.

We present, in this thesis, four contributions towards this overall project. We first present our contribution for determining/modeling a realistic system maximum performance (see Introduction 1.4 chapter). This contribution is used in *Performance Modeling* module (see Figure 4.1a). After, we present our contribution for forecasting Cloud unused resources to mitigate the impact of their volatility. This contribution is used in *Resource Estimation* module (see Figure 4.1a). We also present two contributions aiming to leverage Cloud unused resources for big data applications (see Introduction 1.4 chapter). This contribution is used in *Application Catalog*, *Smart Placement*, and *Resource Evictor* modules. Finally, we present our sabotage-tolerance contribution which is especially relevant for community Cloud models (see Introduction 1.4 chapter). This contribution is used in *Performance Modeling* module (see Figure 4.1a).

# ESTIMATING REAL SYSTEM CAPACITY BY CONSIDERING SSD INTERFERENCES

---

## Problem 1 (Real system capacity estimation)

How to model performance variations?

### 5.1 Introduction

In a container-based system, applications run in isolation and without relying on a separate operating system, thus saving large amounts of hardware resources. Resource reservation is managed at the operating system level. For example in Docker, Service Level Objectives are enforced through resource isolation features of the Linux kernel such as cgroup [129]. Efficiently sharing resources in such environments is challenging in order to ensure SLOs. Several studies have shown that, among the shared resources, I/Os are the main bottleneck [7, 177, 185, 140]. As a consequence, Solid State Drives are massively adopted in Cloud infrastructure to provide better performance. However, they suffer from high performance variations due to their design and/or to applied workloads (see Section 3.1). Moreover, the co-located jobs and/or the hardware may interfere and result in unwanted performance glitches (see Introduction Problem 1.4).

We define three types of I/O interferences on a given application I/O workload in SSD based storage systems. First, an I/O workload may suffer interference due to SSD internal mechanisms such as Garbage Collection (GC), mapping, and wear leveling [79]. We have measured that, for a given I/O workload, depending on the SSD initial state, the performance can dramatically drop by a factor of 5 to

11 on different SSDs because of the GC (see Section 5.1). Second, an application I/O workload may also undergo I/O interference related to the kernel I/O software stack such as page cache read-ahead, and I/O scheduling. For instance, in [159], the authors showed that by using different I/O schedulers (CFQ and deadline) on two applications running in isolation, the throughput may drop by a factor of 2. Finally, the workload may also suffer I/O interference related to a neighbor application's workload. For instance, workload combination running within containers may decrease the I/O performance by up to 38% [177].

In this chapter, we present the investigations achieved about the use of machine learning for building predictive I/O performance models on (SSDs) to anticipate I/O interference issues in container-based Clouds. We evaluated five learning algorithms based on their popularity, computational overhead, tuning difficulty, robustness to outliers, and accuracy: DT, MARS, AdaBoost, GBDT, and RF. Finding the adequate algorithm for modeling a given phenomenon is a challenging task which can hardly be achieved prior to investigation on real data. Indeed, the relevance of the chosen algorithm depends on several criteria such as the size, the quality or the nature of the modeled phenomenon. We have investigated six I/O-intensive applications: multimedia processing, file server, data mining, email server, software development and web applications. The used dataset represents about 16 hours of pure I/Os (removing I/O timeouts) on each of the four tested SSD. We evaluated the relevance of the tested algorithms based on the following metrics: prediction accuracy, model robustness, learning curve, feature importance, and training time. We share our experience and give some insights about the use of machine learning algorithms for modeling I/O behavior on SSDs.

In this chapter our methodology is described in Section 5.2. Section 5.3 details the experimental evaluation performed. Section 5.4 discusses some limitations of our approach. Finally, we conclude in Section 5.5.

## Motivation

We performed some experiments to observe I/O interference due to SSD internals, and neighbor applications.

Concerning SSD-related interference, we focused on the SSD initial state impact. In fact, varying the initial state makes it possible to trigger the GC execution.

We designed microbenchmarks using *fio* [19] relying on the Storage Networking Industry Association (SNIA) specification [168]. This specification includes a secure erase, a workload-independent preconditioning to attain the so-called **SSD Steady State**. We performed intensive random writes with a 4KB request size.

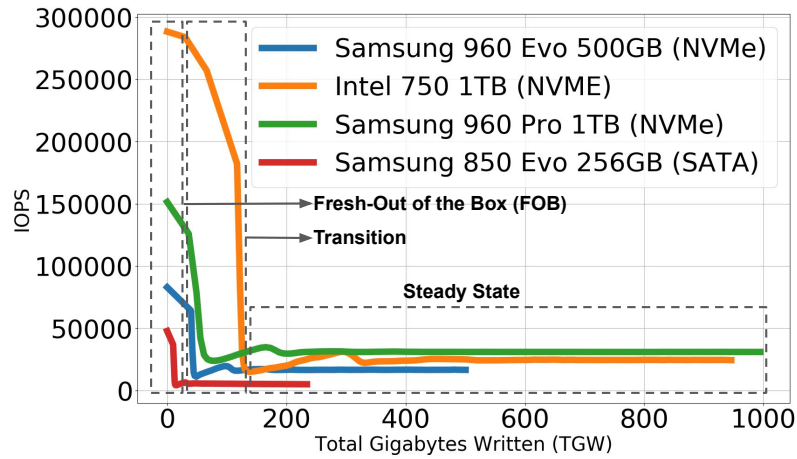


Figure 5.1: I/O performance of random writes for 4 SSDs

Figure 5.1 shows the measured IOPS. One can observe three states for each device: fresh out of box, transition and steady state. More importantly, we can observe from 5x to 11x performance drop when the system sustains continuous bursts of random writes (far below values reported in datasheets). This is due to **GC** latency as it takes more time to recycle blocks when the volume of free space is low. In reality, the system keeps on oscillating between the three states according to the sustained I/O traffic and the efficiency of the **GC**.

We have also performed some experiments to identify the I/O interference due to neighbor workloads (on the same **SSD**). We ran three different containers in parallel and observed the throughput for one specific reference container that runs sequential write operations. For the other two containers, we built up four scenarios ; random write/read, and/or sequential write/read. The volume of generated I/O requests was the same for each experiment. As described in Section 2 of this chapter, *cgroup* v1 cannot limit properly asynchronous I/O traffic. So, containers were not bound by *cgroup* in terms of I/O performance.

Figure 5.2 shows the performance of the reference container for the four scenarios on four different **SSDs**. We observe that the performance drop between the maximum and minimum throughput obtained for the reference container rep-

resents 22% in the best case (SATA disk with which CFQ can be used) and up to 68% in the worst case with a small dispersion (*i.e.*, an interquartile range of 0.04125 in case of the Evo 850 SSD for two executions). This value represents the variation due to the neighboring containers only.

As a consequence, placing a set of containers on a set of SSDs is a real issue that needs to be investigated to avoid high SLO violations.

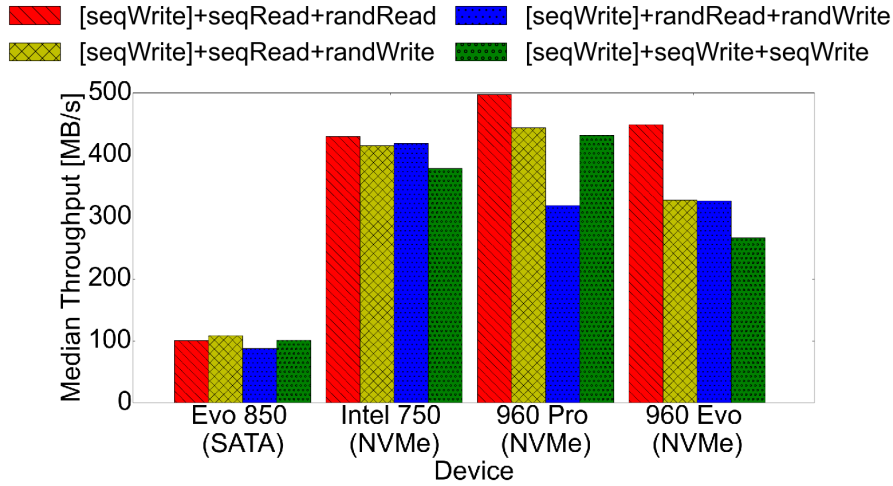


Figure 5.2: I/O Interference of mixed workloads

To conclude, we have illustrated two types of I/O interference, first the sensitivity to the write history which induces I/O interaction with the GC (SSD internals), and second the I/O interactions between I/O workloads which may strongly impact the performance. This motivated us to investigate ways to model I/O throughput taking into account these interactions in order to avoid SLO violations.

## 5.2 Modeling SSD I/O performance: a machine learning approach

### 5.2.1 Approach scope and overview

To build a predictive model that is able to forecast SSD I/O performance in container-based virtualization environment, one needs to fix the scope of the model. From the application point of view, we used six data-intensive applications and a micro-

benchmark: video processing, file server, data mining, email server, software development and web applications, this is detailed in Section 5.2.2. From a system point of view, our study does not focus on I/O variations related to system configuration change (such as read-ahead prefetching window size or I/O scheduler). System-related parameters (e.g., kernel version, filesystem, docker version, etc.) were fixed for our experiments and are detailed in the Evaluation section. Finally, from the storage device point of view, we have experimented with 4 SSD models, both SATA and NVMe, to explore the difference in predictive models as compared to the used technology/interface.

Figure 5.4 describes the overall approach followed that consist of three different steps (see Background Chapter):

- Dataset generation step: A challenge in model building is to use representative data to build an accurate predictive model. In our study, we created datasets by monitoring containers running real applications and benchmarks, see Section 5.2.2.
- Learning step: we built the I/O performance model based on a subset of collected I/O traces/data (supervised learning) using five machine learning algorithms discussed in the Background chapter. In the learning step, one needs to pre-process the data (I/O traces collected) in order to extract the input features and the responses from the traces. Then, one needs to split the data to decide about the part that will be used to train the model and the one used to evaluate it, see Section 5.2.3
- Evaluation step: In this step, we evaluated the accuracy of the trained model, see Section 5.3.

We seek at developing a framework to enable container placement in a heterogeneous cloud infrastructure in order to satisfy users SLO and avoid I/O performance glitches. To achieve this, we devise a self-adaptive container-based *MAPE-K* (Monitor-Analyze-Plan-Execute-Knowledge) [95] loop, an extensively used reference architecture for cloud computing optimization [137, 127, 138] (like the OpenStack Watcher project we have previously developed <sup>1</sup> that was specific to virtual machines).

The *MAPE-K* loop is composed of four main steps depicted in Figure 5.3:

---

1. <http://github.com/openstack/watcher>



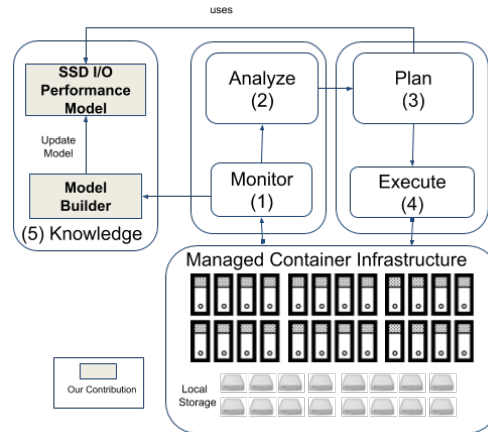


Figure 5.3: MAPE-K

1. **Monitor:** our framework collects containers' I/O requests using a previously designed block-level I/O tracer <sup>2</sup>.
2. **Analyze:** containers' I/O traces are continuously analyzed and preprocessed for the next step.
3. **Plan:** the framework relies on one hand on the containers I/O patterns from the analyze step and current containers placement, and on the other hand on the I/O SSD performance model (see the knowledge part) in order to issue a container placement plan. This performance model is updated continuously when needed according to the monitored I/Os. This may be done either by performing Online learning or by updating the model whenever new applications (new I/O interferences) are run or new storage devices are plugged in.
4. **Execute:** the proposed container placement is scheduled and executed on the real system by calling the adequate APIs of the used infrastructure manager, such as Kubertenes [87].
5. **Knowledge:** in our framework, the knowledge part is related to the SSD I/O performance model built and that drives the overall placement strategy (of the plan phase).

---

2. <https://github.com/b-com/iotracer>

This paper focuses on the **Knowledge** part of the loop. The SSD I/O performance models are built thanks to the **Model builder** component that relies on machine learning algorithms. This paper details our methodology for designing such a component and gives a return of experience about our investigations.

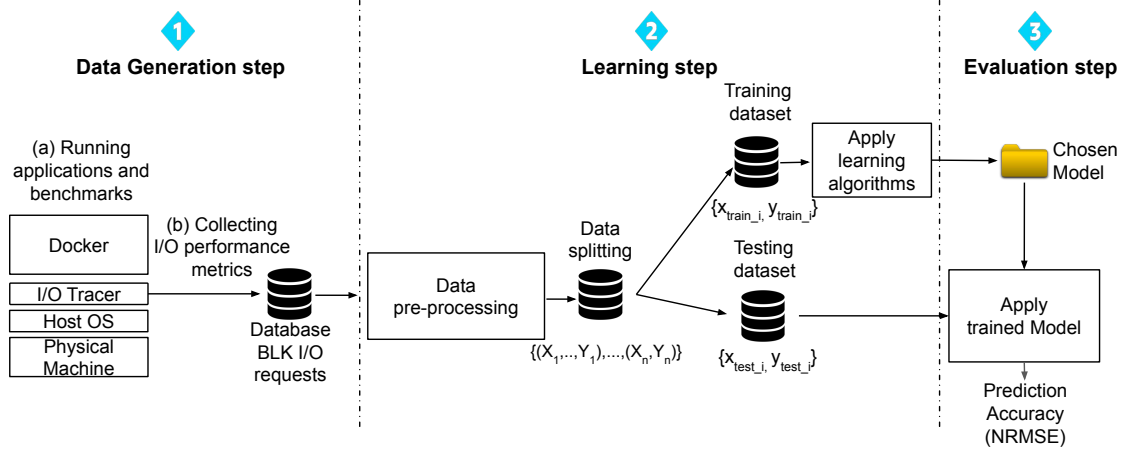


Figure 5.4: Overall Approach

## 5.2.2 Dataset generation step

In the dataset generation phase, we have mainly two steps (see Figure 5.4): generating the workload by executing different applications and collecting the I/O traces. For the sake of our study, we have generated our own datasets. Indeed, we did not find any dataset available off the shelf that represent typical combinations of I/O requests issued from container environments.

We selected six data intensive applications that were deployed in a container-based environment covering various use-cases and I/O interferences. Those applications behave differently from an I/O point of view. We also used micro-benchmarks as defined by [169] to generate more I/O interference scenarios. Table 5.1 summarizes the benchmarks used.

We used four different scenarios to induce different I/O interferences for the 6 applications:

1. Each application was run alone within a container. This was done to determine each application's performance reference without interference (due to

Table 5.1: Applications and benchmarks used

Name	Category	Description
<i>web</i>	Server application	N-tiers web application
<i>email</i>	Server application	Email server
<i>fileserver</i>	Server application	File server
<i>video</i>	Multimedia processing	H.264 video transcoding
<i>freqmine</i>	Data mining	Frequent itemset mining
<i>compile</i>	Software development	Linux kernel compilation
<i>micro-benchmark</i>	Synthetic Benchmark	I/O workload generator

others I/O workloads).

- Up to 5 instances of the same application were run at the same time, each instance was ran within a dedicated container on the same host. This was done in order to generate I/O interference between the same applications. We decided to limit the number of instances to 5 in order to be able to allocate in a fair manner the processor time across the containers. In addition, according to [1], 25% of the companies run less than 4 containers simultaneously per host with a median of 8 containers.
- Applications were run in a pairwise fashion to test all possible combinations. This means that with six applications, we executed 15 combinations (e.g., file server with data mining, file server with web application, etc.). This was done to deliberately generate I/O interference per pair between the applications.
- The six applications were run at the same time in six containers.

These scenarios were executed three times in order to be representative. Additionally to these applications, we used micro-benchmarks to enrich the I/O interference scenarios.

## Generating workload phase

The used applications are briefly described per category (see Table 5.1). To generate the dataset, we used the tools Nginx [141], MySQL [134], and WordPress [28] for the *web* application, FileBench [165] for the *email* and *fileserver*, ffmpeg [64] for the *video* application, Parsec benchmark suite [23] for the *freqmine* application, GNU Compiler Collection [161] for the *compile* application.

**Server application:** We chose three typical enterprise server applications: an n-tiers **web application** (WordPress), file and email servers (Filebench).

WordPress is an Open Source content management system based on Nginx, PHP, and MySQL. In the case of a WordPress website, we varied the number of concurrent readers/writers between 1 and 50. Varying the number of users has a direct impact on the storage system by issuing multiple MySQL connections, and performing multiple table reads/writes. Moreover, MySQL generates many transactions with small random I/O operations. The tool that generates the traffic was run on a separate host.

We used Filebench to evaluate **email** and **file servers** to generate a mix of open/read/write/close/delete operations of about 10,000 files in about 20 directories performed with 50 threads.

**Media processing:** ffmpeg is a framework dedicated to audio and video processing. We used two videos, a FullHD (6.3 GB) and an HD (580MB) video. For the transcoding of the H.264 video, we varied the PRESET parameter between *slow* and *ultrafast*. This parameter has a direct impact on the quality of the compression as well as on the file size. We encoded up to 5 videos within 5 containers simultaneously. Writing the output video generated a high number of write operations at the device level and may generate erase operations when files are deleted at the end of video transcoding.

**Data mining:** This application employs an arrays-based version of the *FP-growth* (Frequent Pattern-growth) method for frequent itemset mining. It writes a large volume of data to the storage devices.

**Software development:** Linux kernel compilation uses thousands of small source files. Its compilation demands intensive CPU usage and short intensive random I/O operations to read a large number of source files and write the object files to the disk. For the sake of our study we compiled the Linux kernel 4.2.

## Collecting containers I/O metrics

Timestamp [milliseconds]	Container ID	Access Type	Address	Accessed Data Size [bytes]	Access Level
1503293862000	340e0a2d67aa	W	83099648	524288	BLK
1503293863000	340e0a2d67aa	W	83100672	524288	BLK

Table 5.2: Sample of I/O requests stored in the time series database

In order to collect the I/O data, we used a block-level I/O tracer <sup>3</sup> which has a small overhead. It is a kernel module running on the host that automatically detects and monitors new containers I/Os. We chose the block level to build a performance model of the storage system, and so only I/Os satisfied by the SSD were considered. Table 5.2 shows a sample of the traced I/O requests. All traced I/Os were inserted in a time series database, see Figure 5.4.

### 5.2.3 Learning step

In [85], the authors described characteristics of different learning algorithms that we have summarized in the Background chapter, we extracted a list of five algorithms that could fit our needs (*i.e.*, DT, MARS, AdaBoost, GBDT, and RF).

In addition to the prediction accuracy criteria, we selected these five algorithms based on the following criteria:

- Robustness to outliers: In storage systems, we are concerned about outliers as on average most I/O requests do not use the whole available performance of the devices.
- Handling of missing values: The large number of possible combinations of I/O workloads require a learning algorithms that can handle missing values.
- Computational complexity: We cannot train the algorithms on every combination once and for all, so we need to be able to recompute the model quickly online to reduce the number of SLA violations.

### Data Pre-processing

The goal of the pre-processing step is to create the matrix of input features noted  $x$  and the vector of the observed responses noted  $y$  (*i.e.*, throughput) from the I/O traces stored in the time series database.

The observed response  $y$  (throughput) is calculated from the captured I/O trace. We need to define a time window that would represent one data sample to be used by the learning algorithm. The objective is to have a time window during which we correlate I/O activities of every single container with regards to

---

3. <https://github.com/b-com/iotracer>

the others. One needs to compromise on the size of this time window. If it is too large, it would capture too many events and I/O interactions, thus the learning phase will lose in precision. A time window that is too small would generate a too large dataset with a large proportion of samples that do not contain relevant information. We chose a middle-ground and used a time window of 10 seconds. We computed the response vector  $y = (y_i)_{i=1}^n$  as follows:

$$y_i = \frac{1}{10^4} \sum_{T_i \leq t \leq T_i + 10^4} d_t \quad (5.1)$$

where  $y_i$  is the throughput in MB/s obtained by one container and  $T_i$  is the starting time in milliseconds of the monitoring period. The variable  $i$  is the number of time windows within a single sampling window.

The selection of the input features  $x$  is a key step to build a good predictive model. One needs to consider the variables that have an influence on the I/O performance for the learning algorithms to find the (hidden) relationships between  $x$  and  $y$  (see Chapter 2). We have selected 9 features listed below based on [168, 177, 136]:

As previously discussed, We have three types of I/O interferences that may affect the throughput: (a) interference due to SSD internals, (b) interference related to the kernel I/O software stack, and (c) interference due to the co-hosted application workloads. One needs to extract the features from the traces in order to represent such interferences.

- Interference (a): this interference is related to the impact of internal mechanisms of SSDs on performance, especially the GC. The more the SSD sustains write operations, the more the GC is initiated, the higher this impact. As a consequence, we chose to capture this feature with the write operations history of the SSD. Indeed, this history gives indications about the state of the SSD.
- Interference (b): As previously mentioned, system related parameters were fixed in this study. However, as we trace at the block level layer, the impact of the page cache and the I/O scheduler is already taken into account.
- Interference (c): they are inferred from the traces as we get the performance of each container knowing what the other collocated containers are doing

and the overall performance sustained by the SSD.

For each  $y_i$  we computed the corresponding row of  $x_i$  that captures the I/O interference as follows:

- *Device write history*: This feature represents the cumulative volume of data written on the device in bytes. We used it to capture SSD internal write operations. Indeed, the more the SSD sustains write operations, the more the GC is initiated, the higher the impact on the application I/Os.
- *Device throughput*: Overall data transfer rate of the device.
- *Device I/O requests*: Number of I/O requests satisfied by a given device.
- *Container I/O requests*: Number of I/O requests per second for each running container.
- *Container random write rate*: Rate of random write I/O requests for each running container.
- *Container written bytes*: Number of bytes written for each running container.
- *Container random read rate*: Rate of random read I/O operations for each running container.
- *Container read bytes*: The number of bytes read for each running container.
- *Container block size distribution*: Block size distribution for each running container.

Table 5.3: Pre-processed data,  $X$ : Inputs (features) and  $Y$ : Output

$X$									$Y$
Device Write History Volume	Device Throughput in MB/sec	Device I/O requests	Container I/O requests	Container Random Write Rate	Container Written bytes	Container Random read Rate	Container Read Bytes	Container Block Size in Bytes	Throughput
20156469248	298.14	152652	152652	0	625262592	0	0	4096	298.14
322122547200	319	652	505	0	264765440	0	0	524288	248.37

Table 5.3 shows a sample of the pre-processing result for a time window of 10 seconds.

## Data splitting

The aim of *Data splitting* step is to divide the data into two distinct datasets, one for training and one for testing purposes.

We randomly used 75% of the data to train our model through the learning algorithms and the remaining 25% were used for validation purpose as recommended by [30]. We ran this selection step 100 times in order to evaluate the robustness of the tested algorithms. The accuracy of the model may change according to the data splitting performed. A robust algorithm is the one that provides a good model regardless of the data splitting being performed.

## 5.3 Evaluation

This section describes the results of our experiments. Through this experiment, we try to answer four research questions:

- **RQ1:** What is the accuracy and the robustness of the tested algorithms?
- **RQ2:** How does the accuracy change with regard to the size of the training dataset (learning curve)?
- **RQ3:** What are the most important features in building the model?
- **RQ4:** What is the training time overhead?

### 5.3.1 Evaluation metric

One of the most common metrics to evaluate the quality of a model is the Root Mean Square Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Where  $y_i$  is the measured and  $\hat{y}_i$  the modeled throughput. The RMSE indicator penalizes large deviations between predicted values and observed values. In order to be able to compare SSDs with different performance. We used a Normalized Root Mean Square Error (NRMSE), given by:

$$\text{NRMSE} = \frac{\text{RMSE}}{y_{\max} - y_{\min}}$$



### 5.3.2 Experimental setup

All experiments were performed on a server with an Intel(R) Xeon(R) E5-2630 v2 CPU clocked at 2.60GHz with 130GB of RAM. Concerning the storage system, we used four SSDs: one with a SATA interface (*Samsung 850 Evo 256GB MLC*) and three others with NVMe interfaces (*Intel 750 1.4TB MLC*, *Samsung 960 Pro 1TB MLC* and a *960 Evo 500GB TLC*).

We used the Ubuntu 14.04.4 LTS GNU Linux distribution with a kernel version 4.2.0-27. We used the *ext4* file system for all the experiments. The virtualization system used was Docker version 1.12.2.

For our tests, we have used the *AUFS* storage driver for managing Docker image layers. However, each container mounts a host directory as a data volume on a locally-shared disk for data-intensive workloads. These data volumes are dependent on the filesystem of the underlying host (*ext4*) which are recommended for I/O-intensive workloads [62]. Finally, all containers get the same proportion of block I/O bandwidth.

We made use of the *xgboost* [44] version 0.6 and *scikit-learn* [139] version 0.18 libraries which provide state of the art machine learning algorithms.

### 5.3.3 Datasets characteristics

This section provides an overview of the used datasets characteristics. For each SSD, the dataset is composed of the six data-intensive applications and a micro-benchmark with the different scenarios presented in Section 5.2.2.

At the block level, 75% of the size of the traced I/Os is between 20KB and 110KB with a median of 64KB which represents most of the typical enterprise block sizes according to the SNIA [59]. The read/write ratios of the tested workloads also covered most of the enterprise applications, see Table 5.4.

In addition, we made sure that the volume of the data written to the disks exceeded by far the size of the disks in order to span the different SSD performance states shown in Figure 5.2.

Table 5.4: Measured workload characteristics

Name	Read/Write Ratio [%]	Seq/Rand Ratio [%]	Block sizes for 80% of I/Os
<i>web</i>	76/24	10/90	8KB, 16KB, 32KB
<i>email</i>	10/90	1/99	4KB, 8KB, 12KB, 16KB
<i>fileserver</i>	83/17	30/70	4KB, 8KB, 12KB
<i>video</i>	40/60	92/8	512KB
<i>frequmine</i>	2/98	99/1	4KB, 8KB, 512KB
<i>compile</i>	9/91	65/35	4KB, 8KB

### 5.3.4 Prediction accuracy and model robustness

As explained in Section 5.2.3, we ran each algorithm 100 times by randomly selecting each time 75% of the dataset (comprising all the applications) to build the model and the remaining 25% to evaluate its accuracy. For each execution, we used 6000 training samples each consisting of 10 seconds of workload (more than 16 hours of pure I/Os excluding I/O timeouts). The accuracy is evaluated through the median NRMSE while the robustness is given by the dispersion.

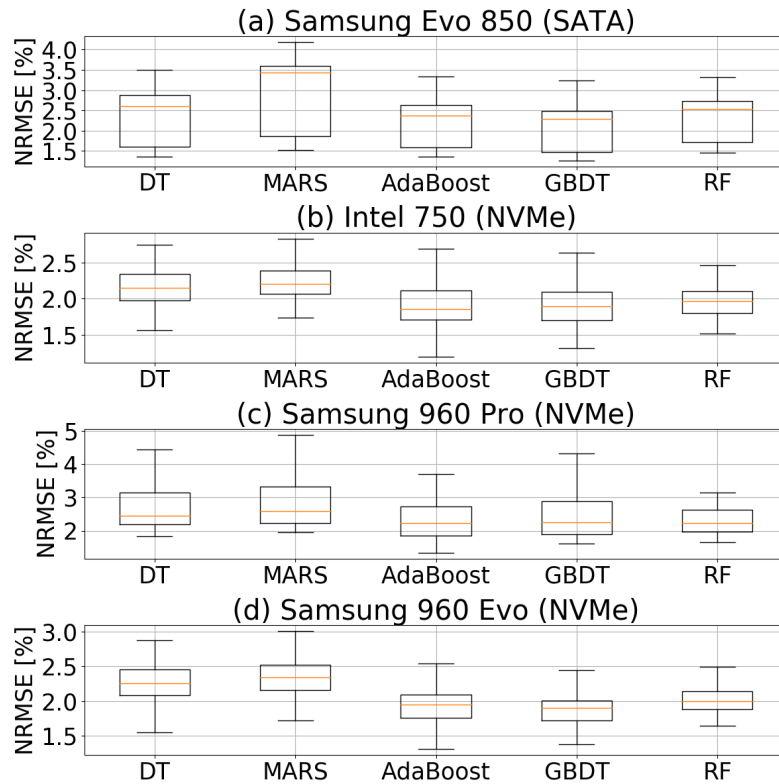


Figure 5.5: Box-plot of NRMSE for each algorithm on all SSDs.

Figure 5.5 shows the boxplots for each learning algorithm according to the

storage device used. A first observation is that the more accurate models (median NRMSE represented by the red line within each box) were achieved with *AdaBoost*, *GBDT*, and *RF* with an NRMSE median error of about 2.5%.

A second interesting observation is that the ranking of the learning algorithms is the same regardless of the SSD being used. This is a very interesting result that means that *different SSD behaviors can be captured with the same learning algorithms*.

A third observation is that *AdaBoost*, *GBDT*, and *RF* also provide a smaller dispersion compared to the other algorithms. Indeed, the models built with those algorithms are less sensitive to the data distribution between the training set and the testing set. This means that the models built with these algorithms are more inclined to be resilient to any I/O pattern change, which is a very interesting property.

Note that *RF* and *DT* gave their results with fixed hyperparameters rather than using cross-validation (see Chapter 2).

Overall one can observe that most of the algorithms used to provide an NRMSE lower than 5% when using the 6000 training samples.

### 5.3.5 Learning curve

The learning curve shows the evolution of the model accuracy (*i.e.*, NRMSE) according to the number of training samples [10, 85].

In order to build our learning curve, we performed a progressive sampling by increasing dataset sizes  $N_{training} = 150$  to  $N_{max}$  with an increase step of 100 samples (where  $N_{max}$  is the total number of samples available). At each step, we ran 100 times the algorithm by randomly selecting the data for each iteration. Note that the minimum size of the training set was fixed to 150 samples which is the size recommended in [85] in order to obtain a good performance when using 5-fold cross validation to estimate the hyperparameters.

In Figure 5.6 we show the accuracy of the algorithms according to the training set size. First, we observe as expected that for each algorithm, the accuracy improves with the increase of the training set size. Second, the best algorithms *Adaboost*, *GBDT* and *RF* have a similar convergence slope. Another interesting result is that the best algorithm ranking is the same for small and large training

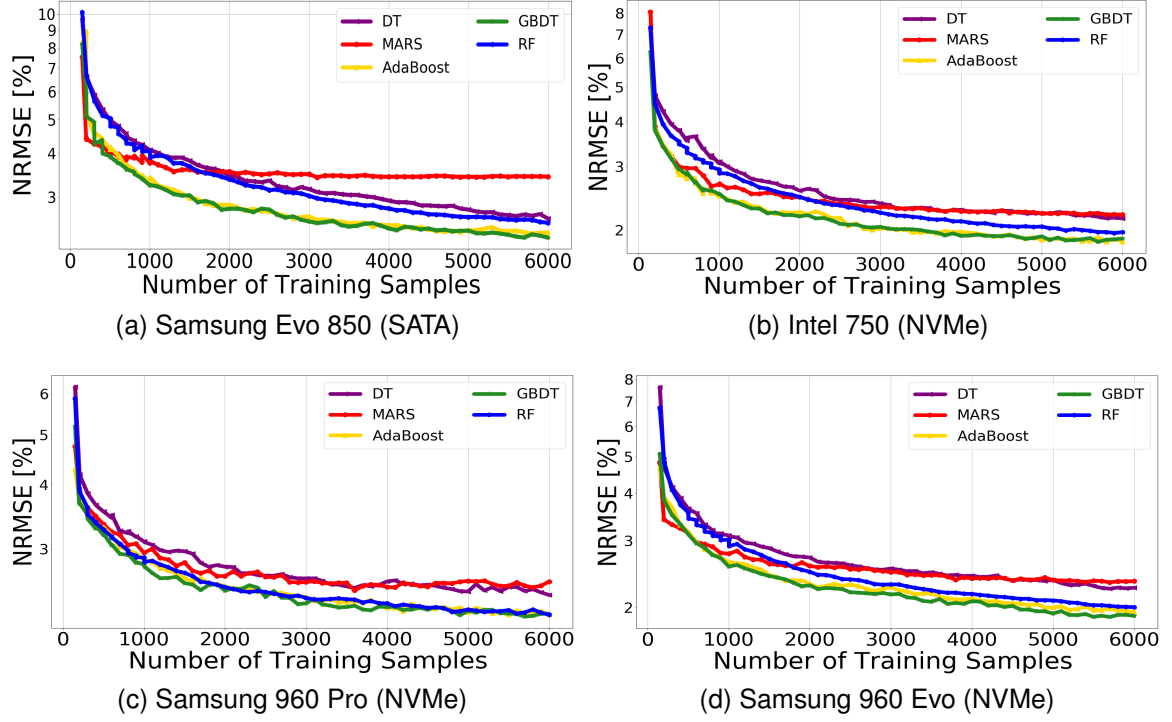


Figure 5.6: Learning curves on the testing set as a function of the number of training samples

samples sets used. This ranking corresponds to the one established in the previous section regardless of the used SSD.

Third, we observed that for the first training samples ( $<1000$ ), the accuracy was not good. This could be explained by the fact that with such a small dataset it is hard to avoid over-fitting but also that the outliers are more difficult to avoid, especially with *MARS* which is not robust to outliers. We can conclude that we need at least about 3 hours (*i.e.*, about 1100 training samples) of pure I/Os to reach a good level of accuracy (*i.e.*, NRMSE).

In a production infrastructure, one may define an off-line SSD warm up period using micro-benchmarks, macro-benchmarks or simply by running target applications on the SSD before integrating the disks into the system. This makes it possible to generate the training samples in order to reach a minimum acceptable accuracy. In addition, in our study, the model is continuously updated according to the workload using a feedback loop. This allows to continuously refine the learning samples.

### 5.3.6 Feature importance

In this section we want to assess the share of each feature in the model we have developed. The feature importance technique can be used to assess the contribution of the selected features to the predictability of the response (*i.e.*, throughput) [10, 85]. The features reaching the highest score are the ones contributing the most to build the model.

Among the five selected learning algorithms, we used *RF* to compute the feature importance as it proved to be one of the most accurate ones for the tested datasets.

As described in [31] with *RF*, one of the ways to compute the feature importance is by measuring the prediction accuracy level set when we randomly swap the values of a given feature. If the accuracy variation is low, then the feature is not important. Figure 5.7 shows the median feature importance of our predictive models.

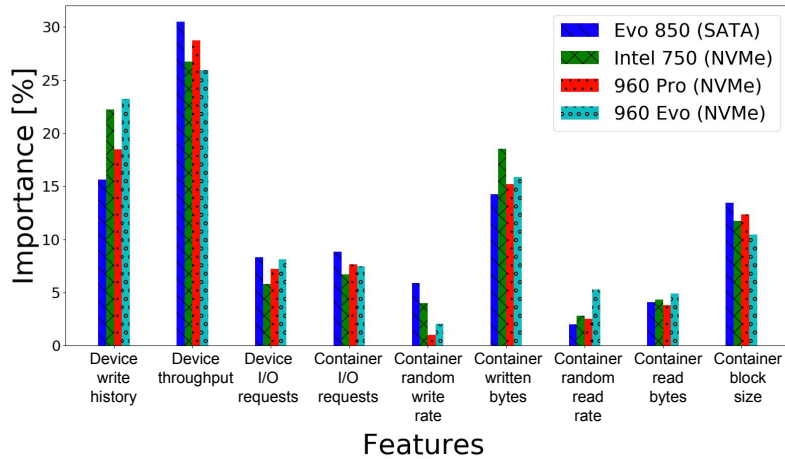


Figure 5.7: Feature importance

We notice in Figure 5.7 that about 46% of the throughput prediction is based on the *device write history* and the *device throughput* features. We also observe that about 27% of the prediction is due to the *container written bytes* and *container block size* features.

In addition, *device I/O requests* and *container I/O requests* contribute to about 17 %. Finally, *container random write rate*, *container random read rate* and *container read bytes* are the less significant features. This means that one may create

a model which is accurate enough without considering those features.

Overall, we notice that regardless of the SSD used, we obtained the same ranking concerning the importance of the features, especially the ones having a high percentage.

### 5.3.7 Training time

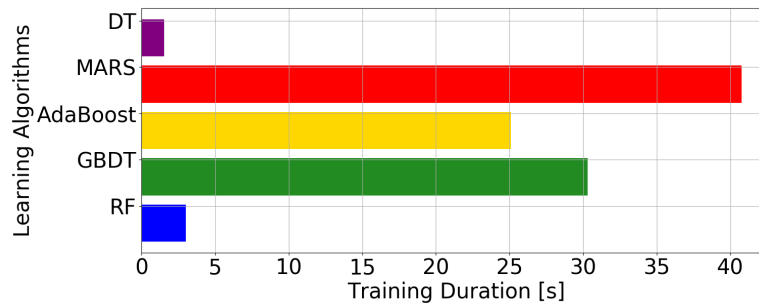


Figure 5.8: Median computation time used for the training of different learning algorithms

The training time is an important metric if one needs to recompute the models for some reason. This may be done either to perform online learning, or to update the model after some new applications are run or some new storage devices plugged in.

Figure 5.8 shows the median computation time taken to train each of the five learning algorithms. It turns out that *MARS* took the longest time for training, with a median time of about 40 seconds (for 6000 training samples). This is due to the complexity of *MARS* which is  $O(m^3)$ , where  $m$  is the number of basis functions (see Background chapter). Then, with a training time of 30 seconds *GBDT* is slower than *Adaboost* (25 seconds). *DT* and *RF* executed in less than 4 seconds.

The duration is highly related to the choice of hyperparameters (*e.g.*, number of folders in  $K$ -fold validation, fixed parameters, etc.).

As compared to the time spent to build the model, the amount of time necessary to make the prediction based on the latter was much shorter (*i.e.*, less than 40 milliseconds).

## 5.4 Limitations

There are some potential issues that may have an impact on the results of this study and that could also be considered for future research:

- System parameters such as file system, kernel version, prefetching window size, continuous or periodic trimming of SSD devices were not varied. These parameters might have an impact on the model building.
- We did not consider CPU, Memory and networks related metrics in our approach. In [166] authors show that processors' caches are shared between all virtual machines, which may compromise performance isolation. Also, in [110] authors show that passing from one to two cores may increase the throughput performance from 870K IOPS to 1M IOPS with a local Flash disk. So, these variables may have an impact on I/O performance in case the CPU is overloaded and cannot satisfy I/O requests for instance.
- The used I/O tracer does not monitor file system metadata [142], this could make our model underestimate the issued I/Os.
- The size of invalid space in a flash memory may have an impact on the performance. In addition to the write history, one may use the number of invalid blocks, for example using *Smartmontools* [9], as a new feature.

## 5.5 Summary

According to our study, it turned out that machine learning is a relevant approach to predict SSD I/O performance in a container-based virtualization. We evaluated five learning algorithms. The features used for the regression were extracted from six data-intensive applications and micro-benchmarks. We experimented with 4 SSDs. We draw several conclusions that may help Cloud providers to design a machine learning based approach to avoid SLO violation due to I/O performance issues:

### **Prediction accuracy and models robustness (RQ1 Findings)**

- *GDBT*, *Adaboost* and *RF* gave the best performance with an NRMSE of 2.5% using 6000 training samples. From the three algorithms, *RF* was the most accurate.

- The ranking of the tested algorithms was the same regardless of the SSD used.
- *Adaboost*, *GDBT* and *RF* provided the smallest dispersion proving there robustness to a changing I/O pattern.
- We used fixed hyperparameters to tune *RF* and *DT*. This makes these algorithms simpler to use.

### **Learning curve (RQ2 Findings)**

- The prediction accuracy is enhanced for every algorithm as we add more training samples.
- The ranking of the algorithms accuracy remained the same regardless of the number training samples (RF, GDBT and Adaboost).
- We need at least a dataset of about 3 hours of pure I/Os to reach a good level of accuracy and a minimum of 150 samples to run the algorithms.

### **Feature Importance (RQ3 Findings)**

- The importance of the features was not balanced. The most important ones were the *device write history*, *device throughput*, *container written bytes* and *container block size*. These features are available off the shelf. Surprisingly, the random write rate did not prove to be very important for the experiments performed.
- The ranking of features importance was the same for all SSDs, especially the most important ones.

### **Training Time (RQ4 Findings)**

- The training time of *glstrf* and *glstdt* was the shortest one.
- The training time of all algorithms was small enough to be used in runtime to update the model. This is a good property if we have to recompute the model for a new device, a new system configuration, or a new I/O pattern.



# ESTIMATING FUTURE USE TO PROVIDE AVAILABILITY GUARANTEES

---

**Problem 2 (Future use estimation)**

How can we estimate, in a flexible and accurate manner, future resources utilization?

## 6.1 Introduction

One way to improve Cloud data center resource utilization and thus reduce the total cost of ownership (TCO) is to reclaim unused resources [126] and sell them. However, reclaiming resources needs to be done without impacting customers' requested QoS. In case of violations of these agreements, penalties are applied. The goal of CPs is to maximize the amount of reclaimed resources while avoiding risks of violations due to resources overcommitment (see Background chapter). Google and Amazon proposed to take advantage of unused resources by leasing them at a lower price compared to regular ones (*e.g.*, dedicated resources). In [11], the authors proposed a similar approach. However, reclaimed resources are coming with limited to no SLA guarantees, which reduces the number of applications that can be deployed [41].

Once the capacity is estimated (see Chapter 5), one way to provide availability guarantees is to predict future use. Predicting a time series is possible since in most cases there is a relationship between the past and the future. However in a Cloud context, predicting the future resource demand is a challenging task, due to many variables that may influence the resource usage such as the users, the number of applications, virtual machines or containers.

In this chapter, we focus on investigating how to provide an accurate estima-

tion of the future used resources (see Introduction Problem 1.4). Our goal is to maximize the leasing of unused resources which, in turn, will maximize potential cost savings for the CP. To tackle these challenges our idea is to use quantile regression to make our model **flexible** for the CP, rather than using the simple mean regression of resource usage. This makes it possible for a CP to make relevant and accurate trade-off between the volume of resources that can be leased and the risk in SLA violations. We used six resource metrics (*i.e.*, CPU, RAM, disk read/write throughput, network receive/transmit bandwidth) for the forecast to be **exhaustive** enough and allow more accurate allocations. We used three learning algorithms: Gradient boosting decision tree (*GBDT*), Random Forest (*RF*) and Long short term Memory (*LSTM*).

For **robustness** concerns, we evaluated our approach using six months of traces about resource usage from four different data centers (*i.e.*, two private companies, one public administration and one university). We evaluated several metrics such as the prediction accuracy per host and for a collection of quantiles. In addition, for **applicability** issue, we measured the training and forecast time to determine the overheads.

We have also evaluated the economic impact of our contribution for comparison. Our results show that the use of quantile regression may provide an increase of the potential cost savings by up to 20% with *glsstm* and about 8% for *GBDT* and *RF* as compared to traditional approaches (see Section 6.3).

Our contributions can be summarized as follows:

- A technique that relies on machine learning and quantile regression that makes it possible to trade-off between the amount of reclaimable resources and SLA violations.
- A comparative study of three machine learning algorithms (*RF*, *GBDT* and *LSTM*) with six quantile levels.
- An evaluation on real traces from four data centers for a six-month time period.

Our methodology is described in Section 6.2. Section 6.3 details the experimental evaluation performed. Finally, we conclude in Section 6.4.

## 6.2 Methodology

Our goal is to provide a solution that maximizes the leasing of unused resources on a set of heterogeneous Cloud infrastructures (*e.g.*, OpenStack). Among the challenges discussed in the introduction, predicting the future use of hosts resources is an important issue. This forecasting has to be robust, fine-grained, flexible and exhaustive. In this section, we present our methodology that aims in building such a prediction model based on different learning machine algorithms. We introduce quantile regression as a technique to limit the risks of SLA violation at the cost of limiting the resources to be sold. We applied our methodology by replaying six months of four real data centers traces.

### 6.2.1 Quantiles

**Quantiles** are data values that divide a given dataset into adjacent intervals containing the same number of data samples [22]. They are useful to gain insight about the distribution of a random value (*e.g.*, CPU utilization) noted  $Y$  as compared to its mean value. **Conditional quantiles** investigate the behavior of  $Y$  by considering another vector of variables noted  $X$  that provides additional information. For example, the time (hour, minute) or historical values of CPU are variables that may be useful to describe CPU behavior (some VMs may be switched off during some period of time each day). The main advantage of conditional quantiles is to give a more comprehensive analysis of the relationship between  $X$  and  $Y$  at different points in the conditional distribution of  $Y$  given  $X = x$ . **Quantile regression** [112] seeks to estimate conditional quantiles. Rather than estimating the mean value of the CPU at a given time stamp, this regression method allows to estimate the  $\tau$ th quantile (*e.g.*, the 0.75th quantile or the CPU utilization value for which 75% of the values are lower).

In Fig. 6.1, we have forecasted the conditional mean of the CPU usage (see Fig. 6.1a) and a collection of quantiles (*i.e.*, 0.05, 0.25, 0.5, 0.75 and 0.95) (see Fig. 6.1b) for a given time window. Quantile regression offers several levels of quantiles that gives us the opportunity to select the one that finds the best trade-off between SLA violations and available unused resources as compared to the conditional mean. As the quantile level increases, the amount of spare (reclaimed)

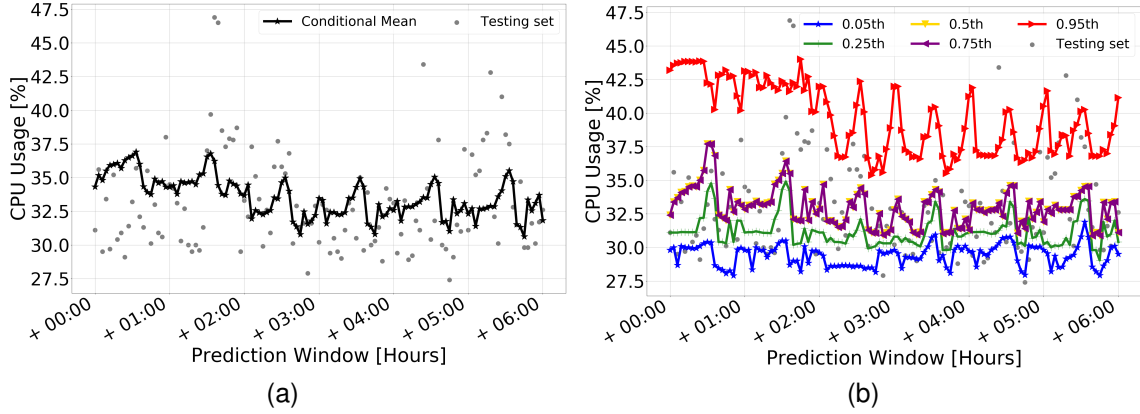


Figure 6.1: Forecasting of six hours of CPU with: (a) The conditional Mean curve in black, (b) Five different quantile regression curves.

resources decreases, the lower the risk of SLA violation. This is the main reason for the use of quantile regression for reclaiming unused resources in our study.

## 6.2.2 Approach overview

Our approach is composed of three steps as shown in Fig 6.2:

- **A forecast strategy step:** we chose to investigate three machine learning algorithms with their corresponding quantile approach. Our objective is to build a forecast model that infers future responses (*e.g.*, CPU, disk) from a set of past traces with different quantile levels. As a consequence, our problem fits in the supervised learning category. Since we want to forecast six metrics (*e.g.*, CPU, RAM), we used regression-based algorithms. We have evaluated RF, glsghdt and glslstm (see Section 6.2.3).
- **A data pre-processing step:** we prepared the extracted datasets from the data centers by applying the following operations: down-sampling, normalization, missing value handling, and features extraction (see Section 6.2.4).
- **An evaluation step:** we replayed six months traces from four data centers by extracting all test windows of 24 hours and their associated training set per host. Then, we built prediction models with six quantile levels (*i.e.*, 0.5, 0.6, 0.7, 0.8, 0.9, 0.99<sup>th</sup>). We evaluated the accuracy, the training time and

the potential economic savings induced by reclaiming resources (see Section 6.3).

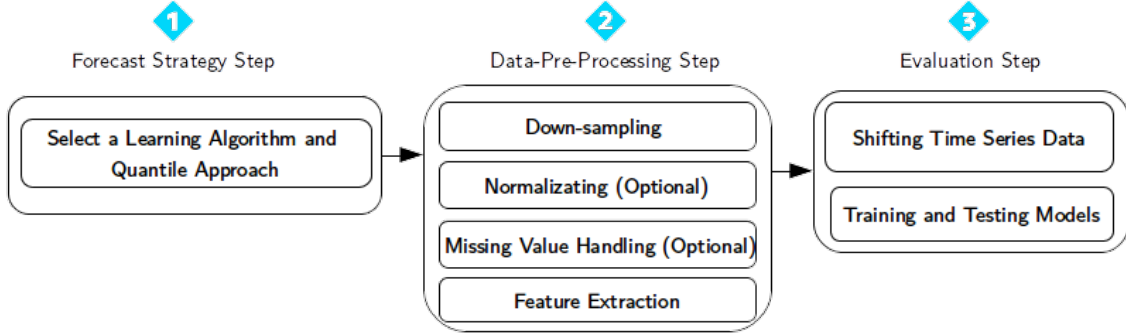


Figure 6.2: Overall Approach

### 6.2.3 Forecast Strategy step

In a Cloud infrastructure, forecasting future resources demands is a challenging task, especially for long periods of time [11]. Many variables may influence resources usage such as the deployed applications, the user behavior and the period of the day [151].

#### Time series

Most CPs store their cluster resource usage indicators in time series. A time series is a sequence of  $N$  measurements  $\{y_1, y_2, \dots, y_N\}$  of an observable metric (e.g., CPU, RAM), where each measurement is associated with a *time stamp*. As confirmed in [163] time series forecasting methods can reliably be used for Cloud resources demand prediction. Several strategies have been proposed to forecast a time series (e.g., multi-step-ahead, iterated-one-step-ahead, recursive-multi-step-ahead, direct-multi-step-ahead). In this study, we used two strategies to forecast time series: (1) **a static strategy** that seeks to find a relationship between values of different time series. **A dynamic strategy called Multiple-input and Multiple-output (MIMO)** which can predict the whole sequence of values [26]. These strategies were used in the context of quantile regression.

## Conditional quantile

There are two approaches to estimate the  $\tau$ th conditional quantile. To summarize these two approaches, we used the following notation:

- $X$  a vector of  $p$  features (e.g., working hours)
- $Y$  an output variable (e.g., CPU usage)
- $y_1, y_2, \dots, y_n$  sampled values from  $Y$
- $x$  one observation of  $X$
- $F(\cdot|x)$  the conditional Cumulative Distribution Function (CDF) of  $Y$  given  $X = x$
- $\mathbb{E}$  the mathematical expectation

The **direct** approach consists in minimizing a sum of asymmetrically weighted absolute residuals [74] based on:

$$q_\tau(x) = \arg \min_{\mu(x)} \mathbb{E} (\rho_\tau(Y - \mu(x)) | X = x)$$

where  $\rho_\tau$  is the following loss function introduced by Koenker and Bassett [112] and  $\tau$  is the quantile level:

$$\rho_\tau(u) = \begin{cases} \tau u & u \geq 0 \\ (\tau - 1)u & u < 0 \end{cases} \quad (6.1)$$

This loss function is asymmetric, except for  $\tau = 0.5$  (median).

The **indirect** approach is performed in two steps, the first one estimates the conditional CDF. Then, the  $\tau$ th conditional quantile of  $Y$  given  $X = x$  is obtained via inversion of the estimated conditional CDF [111] based on:

$$q_\tau(x) = F^{-1}(\tau|x)$$

## Machine learning algorithms

We have investigated three algorithms:

- **RF** and **GBDT** for the static forecasting strategy, these algorithms were recognized to be the best potential choices according to [69], [11] and [38]

- **LSTM** algorithm for the *MIMO* forecasting strategy, where it proved its efficiency in the context of workload prediction as in [115], [158]

The interesting characteristics of these algorithms are summarized in the Background.

## 6.2.4 Data pre-processing step

The goal of the pre-processing step is to create the matrix for input vector features, noted  $X$ , and the vector of the observed responses, noted  $Y$  from past traces. To achieve that, three operations have to be done: standardization/normalization, handling of missing values, and preparing the data for the learning.

The first step is standardization/normalization of the datasets. It turns out that depending on the dataset and thus the company, the sampling rate of the metric collection was not the same. The sampling rate has an impact on the accuracy and the processing time. A too low frequency would provoke the loss of system dynamism and thus may lead to **SLA** violations, but a too high frequency would cause an increase of the processing time. We down-sampled the measurements in order to aggregate a time range into a single value at an aligned *timestamp*. We chose a data sampling rate of 3 minutes as a good trade-off between the processing time and the possibility to capture fine grained behaviors. In addition, as recommended for **LSTM** we scaled the input features between zero and one [118].

The second step handles the missing values that are common in real deployments, the data can be corrupted or unavailable. To achieve that, we filled the missing values by propagating the last valid measurement.

The third step consists in preparing the data by extracting the features  $X$  and the output response  $Y$  from the datasets. This extraction has to be done according to the characteristics of the learning algorithms.

Concerning **RF** and **GBDT**, for each  $y_i$  (*i.e.*,  $Used_{(t,mtr)}$ ), we extracted the row of  $x_i$  as follows:

- We extracted the day, hours and minutes features to investigate the *timestamp* information. We selected these features to allow learning algorithms to find the relationship between these features and resource usage. The

feature month and year were not used since we trained our models using a one-month data.

- We extracted, from the datasets, the holidays and working hours features (*i.e.*, the feature is set to 1 for working hours, and 0 for hours of week-ends or holidays).

For [LSTM](#), the training data required to use a sliding window in order to transform the time series (*i.e.*,  $Used_{(t,mtr)}$ ) into a supervised learning problem.

### 6.2.5 Evaluation step

We evaluated our approach by replaying the six months traces from four data centers. One requirement to consider in order to evaluate a time series forecast compared to traditional supervised learning is to split the training and testing set sequentially in order to maintain the temporal dimension.

To achieve that, the six months of data were shifted into multiple sequential 24 hours windows. Each window is composed of a training window of 1 month and a testing part (*i.e.*, forecast window) of 24 hours. The test window is starting after the end of a training window. As we fixed the forecast window to 24 hours on six months, this gives 183 windows per host. Then, each window was evaluated with *Normalized Mean Quantile Errors* (NMQE).

## 6.3 Evaluation

This section describes the results of our experiments, by which we try to answer four research questions (RQ):

- **RQ1 (Flexibility)**: What are the potential cost savings for [CP](#) with regard to different quantile levels ?
- **RQ2 (Exhaustivity)**: What differences can we observe in [SLA](#) violations when considering several resource metrics as compared to only CPU as in state of the art work.
- **RQ3 (Robustness)**: What is the accuracy of the tested algorithms and how does the accuracy change according to the evaluated workloads ?



- **RQ4 (Applicability):** What is the training overhead of the learning algorithms and its impact on the reclaimed resources ?

### 6.3.1 Experimental setup

To answer these four research questions, we led four experiments, each using production traces from four data centers. In this section, we introduce the elements used in common within these experiments:

- the experimental scenario used to calculate the potential cost savings for CPs in particular the leasing model, the pricing model and the penalty model.
- the metrics used to evaluate the learning phase,
- the experimental environment.

#### Experimental Scenario

**Potential cost Savings :** To calculate the potential cost savings for CPs, we defined three models. First, a leasing model to determine the period during which the customer rents the unused resources and their amount (resource granularity). Second, a pricing model to determine the fee that the CPs would receive from the customer for the provided service. Finally, a penalty model that fixes the amount of discount on the customer bill in case of [SLA](#) violation. We assume that all reclaimed resources are leased. The cost savings estimations do not take into account the cost generated by the leasing such as the wear out of the hardware and energy consumption.

**Leasing Model :** For simplicity, we used a unique model based on the declared capacity of the hosts in the datasets. The leasing granularity is a container runtime provisioned for a period of 24 hours with 2 virtual CPU cores, 8 GB of memory, and 100 Mbp/s for the storage throughput and network bandwidth.

**Pricing Model :** We used a fixed price based on a pay-as-you-go model since it is the dominant schema according to [\[151\]](#). The price was fixed to 0.0317\$ per hour for one leasing model as used by Google Preemptible VMs [\[77\]](#).

**Penalty Model :** There are three types of penalties [72]: (1) a *fixed penalty* where each time the SLA is violated a discount is applied, (2) a *delay-dependent penalty* for which the discount is relative to the CP response delay, and (3) a *proportional penalty* where the discount is proportional to the difference between the agreed upon and the measured capacity.

Public Clouds such as OVH, Amazon and Google use a hybrid approach (*Fixed penalty* and *Delay-dependent penalty*). Table 6.1 shows the discount applied when SLA are not met in case of our experiments.

Table 6.1: Discount applies in case violations for a 24-hour window

Violation Duration [Minutes]	Discount
> 15 to ≤ 120	10%
> 120 to ≤ 720	15%
> 720	30%

## Evaluation Metric

To evaluate the robustness of the learning algorithms and the potential cost savings, we used the *NMQE* and *Interquartile Range IQR* metrics.

**NMQE** A common metric to evaluate the quality of quantile regression is the Mean Quantile Error (*MQE*):

$$MQE = \frac{1}{n} \sum_{i=1}^n \rho_{\tau}(y_{i,mtr} - \hat{q}_{mtr}(\tau, x_i))$$

In order to be able to compare accuracy with different metrics, we used a Normalized *MQE* (*NMQE*), given by:

$$NMQE = \frac{MQE}{y_{\max} - y_{\min}}$$

**IQR** The interquartile range (*IQR*) is a measure of statistical dispersion (*i.e.*, *NMQE*'s) given by:

$$IQR = Q_3 - Q_1$$

where  $Q_3$  and  $Q_1$  are respectively the 0.75 and 0.25 quantiles.

## Experimental environment

We made use of Python with the following packages: *scikit-garden*<sup>1</sup> in case of *RF* with version 0.1 and *scikit-learn*<sup>2</sup> for *GBDT* version 0.18 and *keras*<sup>3</sup> for *LSTM* libraries which provide state of the art machine learning. We also used Apache Spark [160] version 2.0.2. Beside, all training and forecasts were performed on DELL PowerEdge FX2s servers with an Intel(R) Xeon(R) E5-2630 v2 CPU clocked at 2.60GHz with 130GB of RAM. Note that the training and inference are not distributed (*i.e.*, only one server at a time is used for a 24-hour model). However, several models with different parameters (*e.g.*, learning algorithm, quantile level) are trained in parallel to reduce the overall experimentation duration.

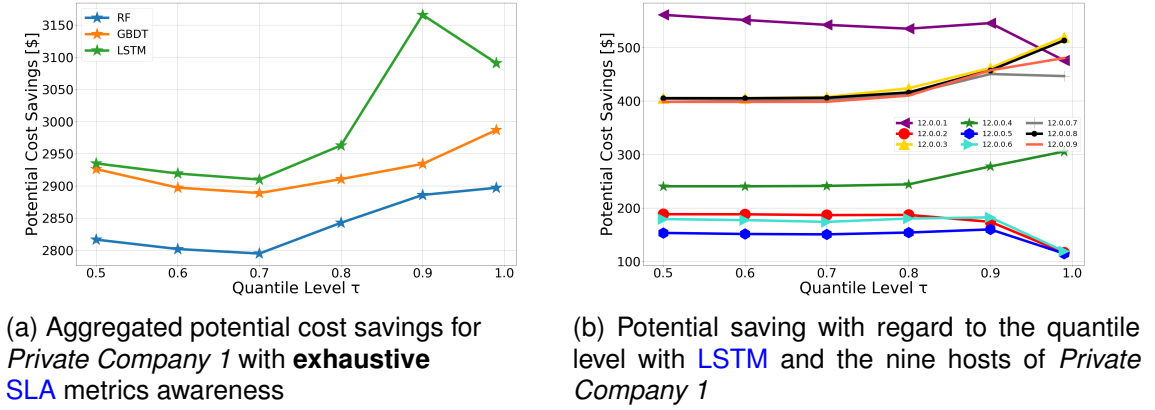


Figure 6.3: Aggregated Potential Cost Savings

### 6.3.2 Flexibility: potential cost savings (RQ1)

To evaluate the benefits of quantile regression for reclaiming unused Cloud resources while achieving the SLA, we compared the potential cost savings for *CP* with regard to different quantile levels. We conducted three experiments:

- **Exp1.** Using the *Private Company 1* dataset, we computed the reclaimable resources amount using the three different learning algorithms. Based on

1. <https://github.com/scikit-garden/scikit-garden>  
2. <https://github.com/scikit-learn/scikit-learn>  
3. <https://keras.io>

these three models, *i.e.*, the leasing model, the pricing model and the penalty model introduced in the previous subsection, we then calculate the potential cost saving in dollars according to the quantile level.

- **Exp2.** We compared the behavior of the quantile regression depending on the host resources usage profile to investigate if the optimal quantile level changes according to the host resource usage.
- **Exp3.** We generalized to the other data centers.

### Exp1. (Aggregated Potential cost savings)

Fig 6.3a shows the potential cost savings in dollars according to the quantile levels for *Private Company 1* and for the three machine learning algorithms.

A first observation one may draw is that the potential cost savings increase with the increase of the quantile level up to  $\tau=0.99$  for both *GBDT* and *RF* and up to  $\tau=0.9$  for *LSTM*.

A second observation is that for each learning algorithm, there is an optimal  $\tau$  level, which corresponds to the trade-off between *SLA* violations and the amount of reclaimable resources (*i.e.*,  $\tau=0.99$  for *GBDT* and *RF* and  $\tau=0.9$  for *LSTM*). This shows that with *GBDT* and *RF*, the decrease of reclaimable resources (increase of  $\tau$ ) is compensated by the reduction of *SLA* violations. However, in case of *LSTM* when  $\tau>0.9$ , this is not the case anymore: the reduction of unused resources is higher than the decrease of *SLA* violations. A third observation is that the best amount of potential cost savings is obtained with *LSTM* with 3166\$ for six months and 9 physical machines.

We conclude that for all learning algorithms studied, quantile regression brings a clear added value: (1) improvement in cost savings as compared to a median-estimation based approach ( $\tau=0.5$ ), and (2) a flexibility to adapt to the optimal level of  $\tau$  according to the selected algorithm. This result can be generalized to all tested data centers as discussed farther.

### Exp2. (Potential cost savings at the host level using LSTM)

Fig. 6.3b shows the potential costs savings at a host level using *LSTM* for *Private company 1*. We notice two behaviors of the cost saving according to the quan-

tile level: (1) the hosts where cost savings increase up to  $\tau=0.99$ ; and (2) those where savings decrease starting from  $\tau=0.9$ . We notice that all the the hosts obeying the first behavior are those with a low usage, such as 12.0.0.3, 12.0.0.4, 12.0.0.8. This can be explained by the fact that an increase in the quantile level does not imply a strong decrease of reclaimable resources, as the peak utilization of resources reaches a maximum of 40% for the CPU and 45% for the RAM. Even when  $\tau$  increases, the loss of cost savings is less significant as compared to hosts with a high utilization (peaks that reach 100%) and with larger resource usage dispersions.

As expected, when comparing the cost savings with the measured resource usage, we notice that hosts with a high usage, such as 12.0.0.2, 12.0.0.5 and 12.0.0.6 generate less savings, except for the host 12.0.0.1 which has extra memory of 170GB compared to the others.

We conclude that: (1) quantile regression makes it possible to adapt to resource usage heterogeneity in data centers and (2) the host **granularity** is relevant for reclaiming resources in a data center.

Table 6.2: Potential Cost Savings with regards to  $\tau$  for all datasets

Dataset	$\tau$	0.5	0.6	0.7	0.8	0.9	0.99
University	RF	2122	2124	2122	2155	2185	<b>2198</b>
	GBDT	2672	2651	2652	2568	2727	<b>2786</b>
	LSTM	2163	2134	2122	2155	<b>2259</b>	2236
Public Administration	RF	4628	4616	4635	4786	5024	<b>5034</b>
	GBDT	4715	4676	4670	4691	4794	<b>4926</b>
	LSTM	4708	4687	4698	4789	<b>5142</b>	4838
Private Company 1	RF	2816	2801	2795	2842	2885	<b>2897</b>
	GBDT	2926	2897	2889	2910	2934	<b>2987</b>
	LSTM	2935	2919	2910	2963	<b>3166</b>	3090
Private Company 2	RF	6659	6650	6655	6887	<b>7414</b>	6803
	GBDT	6670	6728	6763	6995	<b>7210</b>	7153
	LSTM	6428	6441	6528	6736	<b>7722</b>	6857

### Exp3. Generalization on 4 data centers

Table 6.2 shows the aggregated potential cost savings on the four data centers with regards to quantile levels and the three learning algorithms. We observe that for all datasets, *LSTM* is the best choice except for the *university* where *GBDT* gives better cost savings.

Compared to the traditional approaches that use conditional mean (*i.e.*, equivalent to  $\tau=0.5$ ), our approach based on the use of quantile regression performs

better with an increased amount of savings of 8% for *private company 1*, 20% for *private company 2*, 9% for *public administration* and 4% for the *university*.

Overall one can observe that the use of quantile regression is useful for the three algorithms and four datasets and provides the required **flexibility** to reduce SLA violations.

### 6.3.3 Exhaustivity: impact of relying on a single resource (RQ2)

To illustrate the need to apply metrics-exhaustive models, we calculated the cost savings by taking into account only the CPU. We then subtracted the calculated savings to the results previously calculated by our six metrics model.

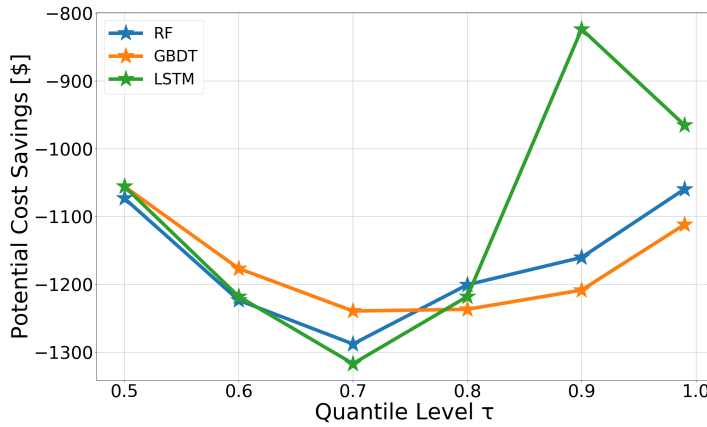


Figure 6.4: Aggregated cost violations for *Private Company 1* when there is no exhaustive SLA metrics awareness (*i.e.*, only CPU)

Fig 6.4 shows the cost of SLA violation when taking into account only CPU. With  $\tau=0.5$ , one can observe that a non-exhaustive choice of metrics leads to a violation of about 1050\$. Taking into account only CPU leads to an increase in SLA violations. Indeed, these violations get higher, up to -1317\$ with  $\tau=0.7$  and then decrease down to  $\tau=0.99$  due to the reduction of reclaimable resources.

To conclude, we observe that a non-exhaustive choice of metrics may leads to no savings as the penalties may be higher compared to the benefits. In addition, the use of quantile regression in a non-exhaustive way has increased the amount of violations.

### 6.3.4 Robustness: resilience to workload change (RQ3)

To evaluate the accuracy of the tested algorithms and observe its evolution along the various deployed workloads, we used  $NMQE$  and  $IQR$  indicators. These are used on all forecasting models and all hosts of *private company 1* with a quantile equal to 0.9.

Table 6.3: Median ( $M$ ) and interquartile range ( $IQR$ ) of  $NMQE$  for all forecast models and all hosts with 0.9 quantile level with *private company 1* dataset.

Metric	Indicator	RF	GBDT	LSTM
CPU	$M$	<b>0.37</b>	0.48	0.57
	$IQR$	<b>0.71</b>	0.91	0.97
RAM	$M$	<b>0.00002</b>	0.14	0.15
	$IQR$	<b>0.09</b>	0.38	0.18
Disk read rate	$M$	<b>0.13</b>	0.21	0.27
	$IQR$	<b>0.62</b>	0.68	0.91
Disk write rate	$M$	<b>0.05</b>	0.1	0.12
	$IQR$	<b>0.11</b>	0.2	0.14
Netreceived	$M$	<b>0.01</b>	0.025	0.018
	$IQR$	<b>0.09</b>	0.138	0.136
Nettransmitted	$M$	<b>0.009</b>	0.014	0.011
	$IQR$	<b>0.04</b>	0.08	0.077

Table 6.3 shows the resilience of the learning algorithms when facing various workloads for six months evaluated using  $NMQE$  and  $IQR$  indicators.

We observe that all the forecast models have quite a good accuracy. We observe that **RF** has the best accuracy regarding the median of  $NMQE$ . also provides the smaller dispersion given by  $IQR$  compared to the other algorithms. This means that **RF** is more inclined to be resilient to workload pattern change, which is a very interesting property. In addition, we observed that *CPU* and *disk read rate* were the metrics with the highest dispersion with a  $IQR$  of 0.71 and 0.62.

When comparing with the potential cost savings, we would expect **RF** to give the best results. However, as shown in Table 6.2 **LSTM** did. This can be explained by the fact that the calculation of the potential savings only penalizes the negative errors (*i.e.*, when the available unused resources are overestimated). Underestimation is not penalized directly compared to the indicator  $NMQE$ , which penalizes both positive and negative errors.

### 6.3.5 Applicability: training overhead (RQ4)

Training time is important as it is directly related to the amount of reclaimable resources. Indeed, the resources used for training and forecast would not be avail-

able for leasing. In this experiment, we evaluated the overhead (median computation time) to train and forecast each of the three learning algorithms for six metrics and a forecast horizon of 24 hours.

Table 6.4 shows the raw results. It turns out that *LSTM* was the slowest with a training/forecast time of about 500 seconds, as it has a high number of parameters to optimize. Then, with a duration of 130 seconds *RF* is slower than *GBDT*. This could be due to the fact that *RF* is estimating the quantile with an indirect approach that requires two steps.

Table 6.4: Median computation time used for the training and forecast 24 hours for one host.

Algorithm	Median Processing Time [Seconds]
<i>GBDT</i>	2
<i>RF</i>	157.11
<i>LSTM</i>	424.20

This means that for a data center composed of 100 hosts the learning phase would take about 12 hours each 24 hours with *LSTM* if we used a similar equipment to the one we experimented. In comparison, *RF* would take about 4 hours and *GBDT* 3 minutes. Note that the duration is highly related to the implementation of the learning algorithms and the choice of hyperparameters. When looking from the point of view of training/forecast computation time *GBDT* seems to be a good choice and *LSTM* the worst.

### 6.3.6 Threats to validity

Our experiments show the benefits of using quantile regression for reclaiming unused Cloud resources while achieving *SLA*. However, as in every experimental protocol, our evaluation has some bias which we have tried to mitigate. All our experiments were based on the same case study regarding the leasing model, the pricing model and the penalty model. We have tried to mitigate this issue by using models close to those of real Cloud providers.

One external threat to validity is our choice of data centers raw data. Further work is needed to reproduce our case study on other datasets, and we cannot guarantee that our results will apply to all data centers. We have tried to mitigate this issue by using datasets from different real *CPs* and different business cases.



Finally, there is a threat that the choice of hyperparameters are incorrectly set for the learning phases even if we relied on strong state of the art work. If this happens to be the case, then all experiments introduce a similar level of imprecision, and a relative comparison of these may still be valid.

## 6.4 Summary

The use of quantile regression is a relevant approach to reclaim unused resources with [SLA](#) requirements. We described our technique that makes it possible to select the quantile level that gives the best trade-off between the amount of reclaimable resources and the risk of [SLA](#) violations. We evaluated three machine learning algorithms with regards to five properties **granularity**, **flexibility**, **exhaustivity**, **robustness** and **applicability** by replaying six months of four data centers traces (*i.e.*, one *public administration*, two *private companies*, one *university*).

We drew four main conclusions:

- **Flexibility (RQ1)**: Our results show that quantile regression provides the required **flexibility** that makes it possible to find the optimal quantile level that maximizes cost savings.
- **Exhaustivity (RQ2)**: the most **robust** learning algorithm was given by *RF* with a median NMQE of 0.37 for *private company 1* hosts. However, traditional accuracy metrics used in machine learning fail to determine the best algorithm that maximizes the potential cost savings while limiting **SLA** violations. Using our approach, it turned out that *LSTM* performs better on robustness for three data centers, with potential cost savings increasing up to 20 %.
- **Robustness (RQ4)**: as expected we need to be as **exhaustive** as possible to avoid **SLA** violations by taking into account a higher number of metrics. We measured that considering only CPU and omitting disk read/write, network and RAM leads to no savings, as the violation amount reaches 1317\$ in the worst case (*i.e.*, a difference of about 145% on the cost savings between an exhaustive prediction and only CPU).
- **Applicability (RQ4)**: for **applicability** concerns, **GBDT** was the one with the smaller computational overhead while **LSTM** had the highest overheads.

# LEVERAGING CLOUD UNUSED RESOURCES FOR BIG DATA

---

**Problem 3 (Ephemeral-aware applications adaptation)**

How big data applications can be adapted to run on ephemeral heterogeneous resources?

## 7.1 Introduction

Advances in technologies such as smart-phones and the Internet of things led us to a data deluge. According to recent estimations [92] by 2025 the amount of data generated will be about 160 zettabytes. MapReduce [51] is a programming model proposed by Google for processing such large amounts of data while providing high performance and fault tolerance. Hadoop [154] is an Open-source implementation of MapReduce that runs across clusters of a large number of computing nodes. Although processing massive data requires a significant amount of computing resources, maintaining such a large-enough dedicated infrastructures to process multiple types of jobs is undoubtedly expensive.

Cloud computing provides on demand access to scalable, elastic and reliable computing resources (see Background chapter). Although these features make Cloud infrastructures good candidates for processing Hadoop workloads, a clear drawback is their operation cost. Furthermore, Cloud computing data centers are often over-provisioned in order to cope with workload variations [39] and nodes failure. This over-provisioning increases the Total Cost of Ownership (TCO) for Cloud providers and results in a low average resource utilization. In the introduction chapter, we have shown that the average CPU usage lies between 20% to 50% on several data centers. Some studies proposed to reclaim these unused

resources and offer them at a cheaper price [126] to increase resource utilization. This led to a benefit increase of 60% for Cloud providers [39]. Therefore, a promising alternative for optimizing the cost of processing data-intensive applications on Cloud infrastructures is to opportunistically exploit their allocated but unused computing resources.

In this chapter, we show that Cloud unused resources could be used to process Big data Hadoop application at a low cost. In order to do so, several challenges need to be tackled: heterogeneity and volatility of resources and isolation with regards to regular customer workloads (see Introduction Problem 1.4).

Our approach relies on three mechanisms. *i*) a Data placement planner to cope with Cloud heterogeneity, *ii*) a Forecasting builder to predict resource volatility (based on the previous chapter 6 contribution), and *iii*) a QoS controller to ensure users SLA guarantee by avoiding interference. The data placement planner relies on the Forecasting builder and decides about the distribution of Hadoop chunks to process according to resource availability. The Forecasting builder relies on quantile regression and machine learning algorithms to accurately predict the amount of unused resources and their availability (volatility) to feed the data placement planner. The QoS controller is used to avoid Hadoop interference on users workloads of the Cloud provider. It achieves that by increasing and decreasing the allocated resources to Hadoop containers on-the-fly.

We evaluated our approach using traces of three months of resources usage from three different data centers (*i.e.*, two private companies, and one university). We compared native Hadoop job execution time with our solution. The experimental results show that Cuckoo divides Hadoop job execution time between 5 to 7 times when compared to the standard Hadoop implementation.

The remainder of this chapter is organized as follows. Section 7.2 presents some information on MapReduce and Hadoop. Then, we describe our methodology in Section 7.3. Section 7.4 details the experimental evaluation we have performed. Section 7.5 presents some limitations of our approach. Section 7.6 concludes the chapter.

## 7.2 MapReduce and Hadoop

In this section, we introduce the key concepts of MapReduce paradigm through its Hadoop implementation.

### 7.2.1 MapReduce programming model

MapReduce is a programming model, inspired by Lisp programming language, and proposed for processing large data sets, potentially using hundreds or thousands of distributed machines [51]. The MapReduce model hides complex tasks, such as partitioning large data sets, scheduling and executing programs across distributed computers, dealing with failures, and handling inter-machine communication from users. This is done with a simple abstraction based on two phases, namely *map* and *reduce*. For each phase, the user writes a specific function (i.e., one map and one reduce function). The map function takes an input data set and outputs a set of intermediate  $\langle \text{key}, \text{value} \rangle$  pairs. After that, the intermediate pairs are grouped by the same key. Then, each set of values corresponding to a single key is forwarded to the reduce function. Finally, each reduce function merges the values trying to form a smaller set of values.

### 7.2.2 Hadoop framework architecture

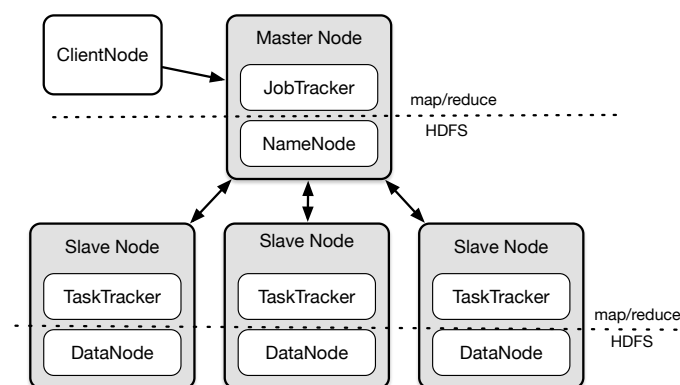


Figure 7.1: Hadoop architecture

Hadoop has a master/slave architecture organized into two main layers (see Figure 7.1). The first layer consists of a single master node called *Jobtracker* and

a set of slave nodes called *Tasktrackers*. The second layer consists of a master node called *NameNode* and slave nodes called *DataNodes*. *Tasktrackers* are in charge of executing map and reduce functions, while *DataNodes* store chunks of input data. Users interact with a Hadoop cluster by means of a *ClientNode*, which is used to send the input data, map and reduce functions to the cluster.

Specifically, a Hadoop user sends its map and reduce functions to *ClientNode*, which in turn, sends them to the *Jobtracker*. Concurrently, *ClientNode* fetches the *block allocation* information (*i.e.*, chunk-to-node mapping) from the *NameNode*. Then, the *ClientNode* splits the input file into even-sized data chunks and streams them to *DataNodes*, which are randomly replicated across the cluster for fault-tolerance. Hadoop runs map and reduce tasks simultaneously. Each task occupies one single processing slot, which is released once the task is completed. When a *Tasktracker* has an empty slot, it sends a *heartbeat* message to the *Jobtracker* requesting a new task. The *Jobtracker* scheduler keeps assigning new tasks to available *Tasktrackers* until all the tasks are done. Hadoop scheduling algorithm favors data locality and does not consider other factors such as system load and fairness. In Hadoop a task is considered as local when both the task and the data chunk to process are initially placed on the same node. Otherwise, it is a remote task.

## 7.3 Cuckoo: a Mechanism for Exploiting Ephemeral and Heterogeneous Cloud Resources

Our main goal is to provide a framework that leverages unused Cloud resources to run Hadoop jobs efficiently without interfering with the co-located workloads.

### 7.3.1 The Cuckoo Framework Architecture Overview

The Cuckoo framework relies on three modules, each of them addressing a specific challenge previously described in Section 7.1 (*i.e.*, resource volatility, heterogeneity, SLA guarantees).

- **Forecasting builder:** This module predicts (as accurately as possible) the future resource utilization at the host level, and therefore the amount of un-

used resources and their availability. The Forecasting builder considers both CPU and memory and its main goal is to estimate the volatility of these resources.

- **Data placement planner:** This module uses the predictions of the Forecasting builder and applies a placement strategy in order to distribute data chunks across the cluster hosts. The Data placement planner solves the heterogeneity of available resources by tuning data chunks allocation according to CPU availability and volatility.
- **QoS controller:** This module guarantees that running Hadoop jobs of *ephemeral customers* do not interfere with regular workloads of *regular customers* in order to ensure the SLA. The QoS controller continuously monitors the resource utilization to detect if *regular customers* could be impacted by *ephemeral customers*. If it is the case some corrective actions are triggered. It also has a preventive mechanism that consists in preserving a certain amount of unused resources to absorb workload variation. To this amount of preserved resources we refer to as *safety margin*.

Figure 7.2 presents both actors and modules and shows how they interact with each other. The *Customer* starts by submitting (1) a Hadoop job using the *ClientNode*. Then, the *JobTracker* sends (2) a request to the *Data placement planner* to check if the *Operator* is able to provide enough resources to process the job within a time window of 24 hours. In order to verify that, *Data placement planner* retrieves (3) the latest resource predictions, which are continuously updated by the *Forecasting builder* module, and creates the block allocation information, which maps chunks to nodes for that specific job. After that, the *JobTracker* replies (4) to the *ClientNode* with either an acceptance or a rejection message depending on the amount of available resources. Following, the *ClientNode* fetches (5) the block allocation map and sends (6) the chunks to the *DataNodes*. Finally, the *QoS Controller* monitors (7) the real-time utilization of unused resources in order to adapt the amount of resources allocated to containers that run *TaskTracker* nodes, in the case of interference.

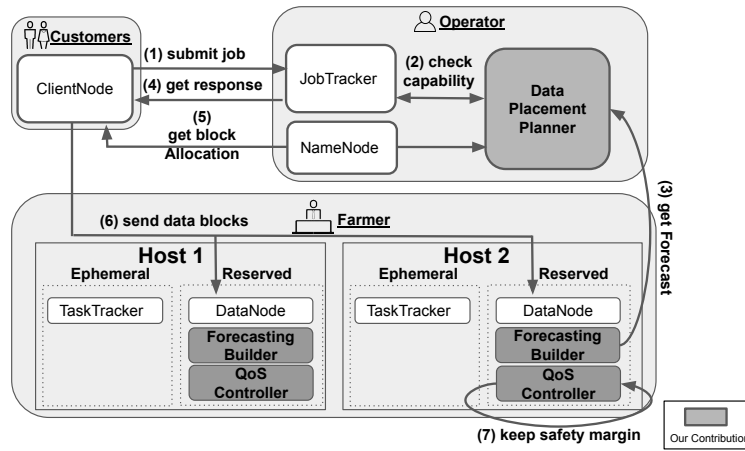


Figure 7.2: Overview of the Cuckoo architecture

### 7.3.2 Forecasting Builder

The objective of the *Forecasting Builder* module is to estimate the future amount of used resources for each host. By doing so, it can estimate the available resources for running Hadoop jobs.

In the previous chapter 5 we have shown that quantile regression is a relevant approach to reclaim unused resources with SLA requirements. This work has shown that quantile regression may increase the amount of savings by up to 20% compared to traditional approaches. We use quantile regression to implement our forecasting builder module.

Quantile regression provides the accuracy of machine learning algorithms with the flexibility of quantiles. Moreover, quantiles make it possible to reason about the trade-off between the amount of reclaimed unused resources and the potential SLA violations. In our work, we chose to use Gradient Boosting Decision Tree which gives the best trade-off between prediction accuracy and training time in order to forecast a 24-hour time window.

### 7.3.3 Data Placement Planner

The objective of the *Data Placement Planner* is to find the best data block mapping in order to minimize the overall execution time given the available resources by minimizing data transfers. To achieve that, we use a modified version of the



Weighted-Round-Robin (WRR) algorithm. WRR is designed to handle hosts with different processing capabilities by assigning a different weight to each [8].

In our approach, the weight was calculated by taking into account the predicted usage of resources and the processing capability for each host, estimated in *GFLOPS*<sup>1</sup> within a time interval of 24 hours. Then, data chunks are distributed proportionally to the weight assigned to each host. That is, hosts with higher weights receive more data chunks to process than hosts with lower values.

The safety margin value (described in Section 7.3.4) denoted by *sm* is also taken into account. This value is used to remove the corresponding proportion of resources from the pool of unused resources that is the input of the *Data Placement Planner*. As mentioned before, the *sm* value is used to absorb unpredictable workload behavior and forecasting errors. For instance, if the forecast builder estimates that 12 cores are used in a 32 cores machine, then 20 cores would be available during the next 24-hour time window. If *sm* value is 10%, then 2 cores (from the 20 available) are removed from the the pool and only 18 cores are considered.

The following algorithms describe *i*) the process of calculating the host weight (Algorithm 1), and *ii*) the data chunk placement strategy (Algorithm 2).

---

**Algorithm 1** Host weight calculation

---

```

1: function CALW(hostid, sm, ti, sp)
2:   weight  $\leftarrow$  0
3:   cap = getFlops(hostid)
4:   predUsage = ForecastingBuilder(hostid, ti, sp)
5:   for each load  $\in$  predUsage do
6:     load += sm
7:     if load < 100 then
8:       weight += (cap - ((load/100) * cap)) * sp
9:     end if
10:  end for
11:  return weight
12: end function

```

---

**Calculating the host weight:** Algorithm 1 has four input parameters: *host<sub>id</sub>*, a safety margin (*sm*), a time interval (*ti*) and a sampling period (*sp*). The parameter

---

1. Giga Floating Point Operations per Second

$ti$  is the time for which we calculate the weight. In our case, we used a  $ti$  of 24 hours during which we measured hosts resources usage with a sampling period ( $sp$ ) of 3 minutes, thus resulting in 480 measures every 24 hours.

First, we retrieve the maximum processing capacity ( $cap$ ) for the selected host  $getFlops(host_{id})$  at *line 3*. Then, we request the Forecasting Builder module to estimate the available amount of resources ( $predUsage$ ) for this host  $ForecastingBuilder(host_{id}, ti, sp)$  at *line 4*. The *Forecasting Builder* returns a set of  $ti/sp$  data prediction points. Then, we iterate over these  $predUsage$  data points to compute the weight of the selected host (*lines 5–10*). For each predicted data point, we add the safety margin to the predicted load (*line 6*). Then, if the total used load is under 100 % (*i.e.*, the host has some unused resources), the weight is calculated by subtracting from the total processing capacity of the host noted  $cap$  the  $predUsage$  and  $sm$ . We then multiply the result by the specified sampling period to integrate the duration (*line 8*). The higher the free CPU resource and capacity, the higher the weight.

**Data placement strategy:** Algorithm 2 starts by computing weights for each host using Algorithm 1 (*i.e.*, host weight calculation). For each job, first we initialize a matrix of Booleans for the chunk mapping called *blockAllocation* at *line 3* and we get the chunk replication factor used by Hadoop with function  $getReplication()$  at *line 4*. Second, for each chunk of the job (*lines 5–12*), we retrieve the estimated processing costs using  $getEstimatedTaskCost(chunk_{id})$  function at (*line 6*). In this work, we have used fixed costs (see Section 7.4) but map or reduce tasks costs could be estimated as proposed in [107]. Then, we select a host according to the assigned weights (based on WRR) at *line 8*. The hosts with higher weights are selected first. Next, we update the matrix *blockAllocation* to indicate that the chunk ( $chunk_{id}$ ) has been placed on the chosen host ( $host_{id}$ ) at *line 9*. Finally, we dynamically update the weight of the host by decreasing its value according to the used resources ( $cost$ ) and to updated predictions (*line 10*). We repeat these three steps for the default number of replicas (*i.e.*,  $nbReplicas$ ) initialized for Hadoop (*lines 7–12*). If there is not enough resource for processing a chunk, a rejection message is sent. When all the chunks of all the jobs have been placed, we send the allocation matrix *blockAllocation* to the *NameNode* (*line 14*) and an acceptance message. Finally, the block allocation matrix is retrieved by the

*ClientNode.*

---

**Algorithm 2** Data Placement Algorithm

---

```

1: weights = initWeights()
2: for each job ∈ JobTracker.all() do
3:   blockAllocation[nbChunks, nbHosts] = false
4:   nbReplicas = getReplication()
5:   for each chunkid ∈ job.chunks do
6:     cost = getEstimatedTaskCost(chunkid)
7:     repeat
8:       selectedH = selectHostid(hosts, weights)
9:       blockAllocation[chunkid, selectedH] = true
10:      weights[hostid] = updateW(selectedH, cost)
11:      nbReplicas − −
12:    until nbReplicas == 0
13:   end for
14:   send(job, blockAllocation)
15: end for

```

---

### 7.3.4 QoS Controller

The QoS controller implements a mechanism that reacts to under estimation of the used resources from the Forecasting builder. Indeed, as discussed before, prediction errors may exist due to an unexpected variation of the *regular customers* workloads. The reactive policy of the QoS controller checks if the *regular customers* workloads are using more than a predefined threshold of the safety margin (tuned to 50% in our experiments). In this case, the allocated resources for the *ephemeral customers* jobs must be reduced or completely released.

The QoS controller manages both the CPU and memory resources. In order to release resources the QoS controller proceeds as follows. The CPU control is done by adjusting dynamically the hard limits of the CPU cycles that a container is able to consume. In this way, Hadoop jobs cannot use more CPU than the amount of time set for the container. As a consequence, the map or reduce tasks will be slowed down without affecting regular customers workloads.

For the memory resource, Cuckoo has a more aggressive strategy and it acts as a system memory killer. Cuckoo kills proportionally the amount of map or reduce tasks necessary to free the safety margin related to memory occupation.

In case of CPU and/or memory starvation (*i.e.*, only the safety margin resources are available) the container is killed).

## 7.4 Experimental Validation

This section describes the experiments conducted to validate the efficiency of the Cuckoo framework.

### 7.4.1 Experimental Methodology

We tried to answer three research questions (RQ) in order to tackle the 3 challenges mentioned in the section 7.1 (*i.e.*, resource heterogeneity, volatility and QoS guarantee):

**RQ1:** What is the overall performance of Cuckoo compared to native Hadoop implementation ?

**RQ2:** How does the forecasting builder accurately model the volatility of resources ?

**RQ3:** What is the effectiveness of Cuckoo with regards to the number of remote tasks?

We have conducted several experiments to evaluate our solution. We used a 3-month production data set from three different data centers and compared Cuckoo to standard Hadoop implementation. By standard Hadoop implementation we mean that the data chunks are uniformly distributed across nodes and the selected job scheduler is FIFO. However, compared to a standard implementation we have an injection phase that consists in varying the CPU and Memory load over time according to data centers' traces.

In our evaluations, we used two configurations related to the number of map tasks that is 514 and 640, corresponding to two different data sets of 40GB and 32GB respectively in a network of 50Mbps. In both cases we used 40 reduce-tasks. The processing costs for each Map and Reduce task is equal to 3100 FLOPS/Byte and 6300 FLOPS/Byte for the map and 1000 FLOPS/Byte for the reduce tasks. We set the chunk size to 64 MB and configured Hadoop to host three replicas for each chunk (default configuration) and each *TaskTracker* to run 20 slots of map and reduce tasks. According to [146], usually each task needs

between 2 GB and 4 GB of memory which means that for a machine with 48GB of memory the Hadoop TaskTracker could run between 10 and 20 tasks in parallel.

The experimentation has three phases: *i*) infrastructure initialization, *ii*) deployment, and *iii*) injection. The infrastructure initialization phase configures the physical machines (*i.e.*, speed, number of cores, memory), the network (*i.e.*, topology, available bandwidth, latency) according to the three data centers. Then, the deployment phase consists in launching the Hadoop and Cuckoo modules (*e.g.*, *JobTracker*, *DataNode*, *Data Placement Planner*). Finally, the injection phase consists in varying the CPU and Memory load over time according to data centers traces. The injection is done by replaying the traces from three data centers. As we fixed the forecast window to 24 hours on three months, this gives 92 windows per host.

Our experiments were performed using Simgrid 3.20 simulation tool and a customized version of MRA++ MapReduce [15] for handling the three phases.

## 7.4.2 Data sets

Each data set corresponds to a specific data center. The largest data center is PC-2 (*i.e.*, private company 2) with 27 hosts providing a total of 3552 GFLOP/s and 3.8TB of RAM memory, followed by PC-1 and University. Table 7.1 shows the overall capacity of all data centers.

Table 7.1: Total capacity of each data center

Name	Number of Hosts	CPU [GFLOP/s]	RAM [TB]
PC-1	9	2208	1.2
PC-2	27	3552	3.8
University	10	1363	1.5

Moreover, these data centers are heterogeneous. The PC-1 has six different configurations among its nine hosts, PC-2 has 13 different configurations among its 27 hosts, and finally the University data center has 6 different configurations. The average resource utilization of each data center is 13% for PC-1, 4% for PC-2 and 6% for University. These results motivated us toward reclaiming unused resources.

### 7.4.3 Experimental Results

We evaluated the overall execution time of a job according to the safety margin value, the number of remote tasks and the number of rescheduled tasks.

#### RQ1-Job Execution Time

To evaluate the benefits of Cuckoo compared to Hadoop we compared the job execution time according to different configuration of safety margin (*i.e.*, 0%, 5%, 10%, 15%, 20%, 25%, 30%).

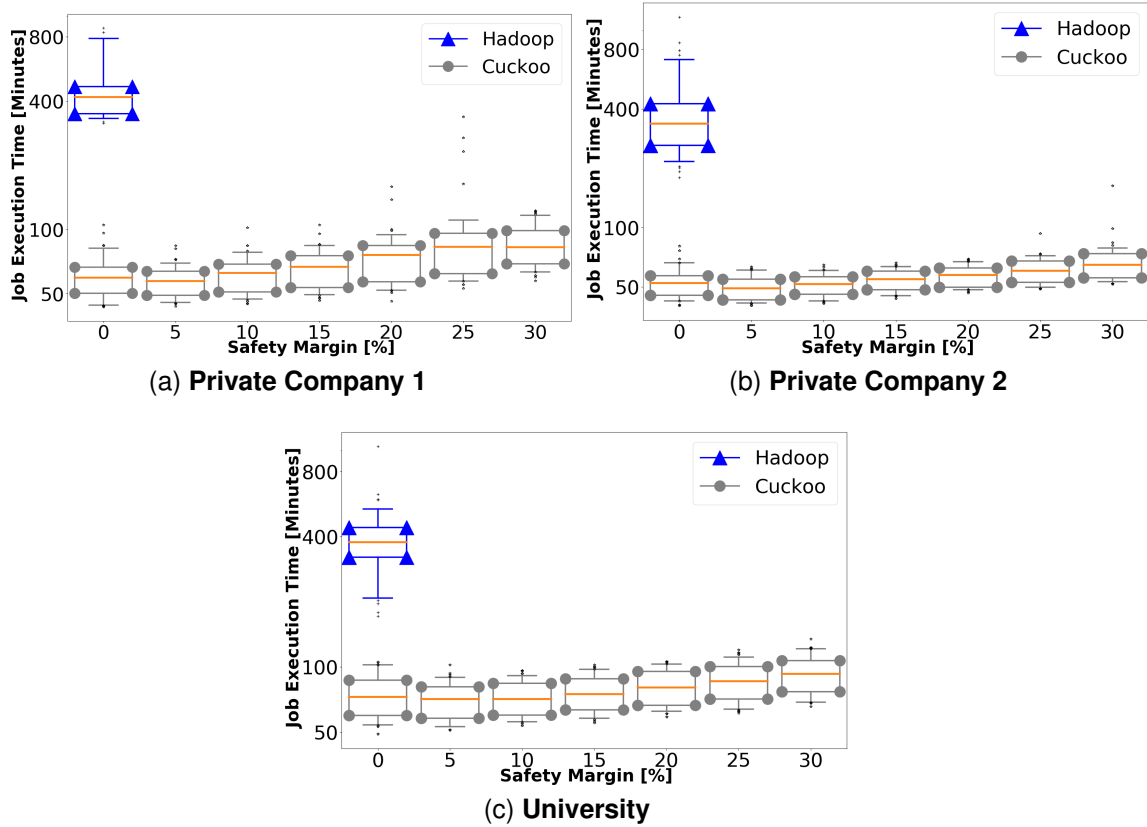


Figure 7.3: Job execution time for standard Hadoop and Cuckoo

Figure 7.3 shows the resulting job execution time. We observe that the minimum median completion time of Hadoop is 417 minutes for PC-1, 336 minutes for PC-2 and 377 minutes for University. On the other hand, the best median job completion time for Cuckoo is 57 minutes for PC-1, 49 minutes for PC-2 and 71 minutes for University with a safety margin of 5%. The smaller dispersion and

best completion time was observed in the case of a 5% safety margin for all data sets. Then, with a safety margin greater than 5%, we notice that the job execution time increases. This is due to the fact that increasing the safety margin size leads to a decrease on the amount of reusable resources and thus the slow down of the Hadoop jobs. In addition, we observed that the best safety margin is the same regardless the data set evaluated. This means that the prediction of the forecast builder is accurate. Cuckoo is 7 times faster than native Hadoop strategy for PC-1 and PC-2 and 5 times faster for the University.

### RQ2-Effetiveness of the Forecast builder

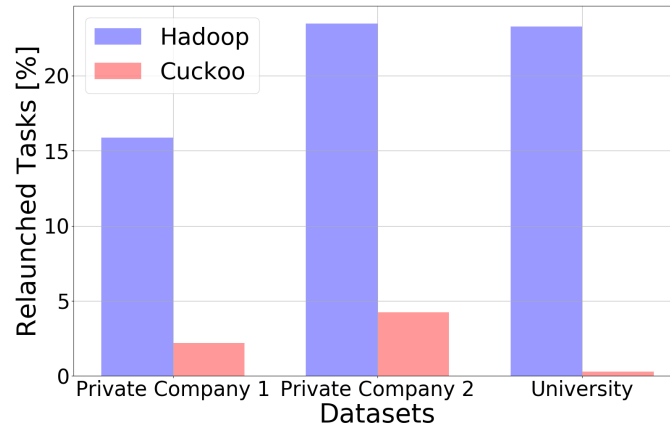


Figure 7.4: Median percentage relaunched tasks comparison between Hadoop and Cuckoo

As mentioned in Section 7.1, one way to increase data center utilization is to reclaim unused resources to run ephemeral containers that are executing map or reduce tasks. However, such containers could be evicted by the *QoS Controller* in case of interference with the regular customers workloads (see Section 7.3.4). It means that the task should be relaunched causing a resource waste. Figure 7.4 shows the number of relaunched tasks for the three data sets with a safety margin of 0%. The lower the number of relaunched tasks, the better the volatility taken into account by the Forecast builder.

We observe that with the standard implementation of Hadoop, more than 15% of the tasks were relaunched while only less than 5% are in case of Cuckoo for the three data sets (3 times less). This confirms that our prediction module helps

Cuckoo to handle the volatility of resources efficiently.

In Cuckoo, PC-2 has the higher percentage of relaunched tasks with about 5% while the University the slower, with less than 1%. The percentage of relaunched tasks represents the percentage of killed and rescheduled tasks due to violation. This means that PC-2 is wasting more resources when compared to University and PC-1. Moreover, a high number of relaunched tasks may lead to remote tasks, as explained in Section 7.4.3.

### RQ3- Effectiveness of Cuckoo and Remote Tasks

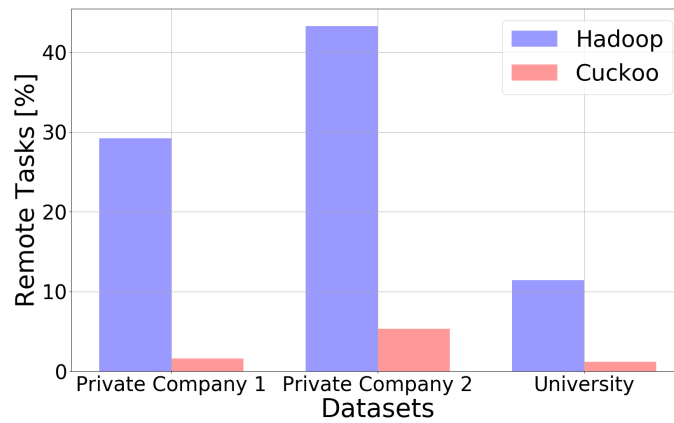


Figure 7.5: Median percentage remote tasks comparison between Hadoop and Cuckoo

In this experiment we evaluated the percentage of remote tasks generated by Cuckoo and we compare it to the ones generated by Hadoop standard implementation. We have measured data movement during the experiments (see Section 7.4.1). A task becomes remote when a *TaskTracker* has an empty slot and no more available local chunks. In this case, a chunk is downloaded to be processed. In heterogeneous environments this situation is more likely to happen. Indeed, Hadoop distributes data chunks uniformly across nodes and expects them to run tasks with the same execution time. Thus, running Hadoop in heterogeneous systems requires data chunks to be reallocated to feed available computing slots. This creates network traffic overhead, and therefore degrades the system performance. So, by estimating the number of remote tasks we evaluate the ability of Cuckoo to manage heterogeneity and volatility. Figure 7.5 shows the



percentage of remote tasks. One may observe that Cuckoo outperforms standard Hadoop for all data sets. In the case of PC-1, Cuckoo has reduced the percentage of remote tasks by about 7 times, while in PC-2 and University by 6 and 19 times respectively. This confirms that Cuckoo handles both heterogeneity and volatility of resources effectively. As discussed in Section 7.1, remote tasks take longer to execute as compared to local tasks due to the required data transfer time.

We also observe that PC-2 has more remote tasks. This can be explained by the fact that PC-2 is the largest data center with 27 hosts and since data chunks were distributed across hosts, more data transfers were generated. So for a given number of chunks to process, the higher is the replication factor, the lower the number of data movements are. In addition, the lower the number of hosts, the lower data movements for a given replication factor.

We conclude that for all three tested datasets, Cuckoo outperforms Hadoop in all the cases. This can be explained by the fact that our data placement and the predictive provisioning strategies together make Cuckoo volatility and heterogeneity aware, and therefore able to required less remote and relaunched tasks, when compared to Hadoop standard implementation.

## 7.5 Limitations

There are some potential issues that may impact on the results of this study. We will consider these issues in a future work. In the following we highlight some of them:

- We did not considered job and task scheduling, and memory, network and storage for the data placement decisions. These resources may have an impact on the overall performance. This limitation has been partially addressed in our recent contribution [83].
- The Map/Reduce tasks are single-threaded and cannot leverage the computing power of more than one core.
- The sampling period of the data center traces is 3 minutes which means that we cannot measure violations with smaller time sampling.
- We did not consider how co-located application workloads may interfere on

Map or Reduce tasks. We have considered only a fixed capacity per host, while the capacity may depend on all running applications [49].

## **7.6 Summary**

Data center resources are underused. They are heterogeneous and their usage is not balanced among hosts. We argue in this chapter that those resources could be used to process Big data Hadoop application at a low cost. In order to do so, several challenges need to be tackled: heterogeneity and volatility of resources and isolation with regards to regular customer workloads.

To tackle these issues, we have developed a heterogeneity and volatility aware data-placement strategy called Cuckoo. Volatility and heterogeneity are managed by our proposed forecasting and resource-aware data placement strategies. In addition, a QoS controller is used to avoid any interference with regular workloads by means of a safety margin.

Our results show that Cuckoo outperforms standard Hadoop implementation by a factor of 5 to 7, while avoiding any interference with regular customer workloads.

# PREVENTING MALICIOUS INFRASTRUCTURE OWNERS FROM SABOTAGING THE COMPUTATION

---

## Problem 4 (Malicious farmers prevention)

How can we prevent malicious infrastructure owners from compromising the computation?

## 8.1 Introduction

A promising alternative for optimizing the cost of processing applications on Cloud infrastructures is to opportunistically exploit their allocated but momentarily unused computing resources [50]. Many platforms (e.g., BOINC [14], Condor [123]) enable the leveraging of these unused resources for a variety of purposes (e.g., scientific computing, big data) and business models (e.g., free, reward). In the previous chapters (5, 6, and 7) we demonstrated the possibility of leveraging Cloud unused resources for big data without interfering with the co-located workloads.

However, any infrastructure owner (*i.e.*, *Farmer*) can join such platforms to provide/share his/her computation capacities. These farmers seek to reduce their TCO by making their unused computing resources available to other users. Allowing any farmer to join such platforms exposes an *Operator* (*i.e.*, interface organizations between the farmers and the customers) to malicious behavior. Malicious farmers can potentially produce erroneous or inaccurate results without effectively running the applications to obtain higher benefits from the Operator (e.g., while saving their computation capacities) [155]. In such a scenario, one needs to in-

investigate *How can we prevent malicious infrastructure owners from compromising the computation?* (see Introduction Problem 1.4).

Many studies have been conducted to provide secure remote computation [17, 148]. Most of the traditional approaches such as replication voting, ringers, and spot checking - whether with or without blacklisting - have a high overhead on the compute resources (it may double the used resources) to verify each application execution or requires a dedicated hardware such as Intel SGX with 60% of the native throughput and about 2x increase of the application code size [17].

In this chapter, we propose a different but complementary solution to state of the art work having the following properties:

- **Backward compatibility:** non-invasive/non-intrusive on the application code and not limited to a type of application or hardware.
- **Online execution:** continuous verification of the correct execution of the application.
- **Efficiency:** proving a small overhead to verify each application execution

Our approach relies on the use of classification techniques to build a fingerprint model of an application execution in a trusted environment using the Random Forest learning algorithm. In this work, we assume that performance metrics are continuously sent by the farmer as in [35, 2]. Then, the built trusted fingerprint model is continuously compared with the current workload metrics sent from the untrusted environment to detect an application execution sabotaging or the alteration of its behavior. To do so, three different cases can be observed:

- The **homogeneous hardware** case where the targeted hardware is both standardized and specified. This means here that the model is trained with this standardized hardware which is the same as the targeted one.
- The **heterogeneous hardware** case where the targeted hardware is specified but varies from machine to machine. This means that the model is trained with the same (heterogeneous) hardware mix as the targeted one.
- The **unspecified hardware** case where the targeted hardware is both unspecified and heterogeneous. This means that the model is not trained with the same hardware mix as the targeted one.

We have investigated five applications: multimedia processing, file server, 3D rendering, software development, web application.

Our experimental results show that our fingerprint recognizer is able to detect the correct execution of applications in a trustless environment with a median accuracy of 99.88% for *homogeneous hardware*, 98% when using *heterogeneous hardware* and 44% in case of *unspecified hardware* during the training phase (see Section 8.3).

The remainder of the chapter is organized as follows. Section 8.2 presents our methodology? Then, section 8.3 details the experimental evaluation performed. Finally, we conclude in Section 8.5.

## 8.2 Methodology

In order to monitor application resource usage, different metrics (*e.g.*, CPU usage, memory usage, throughput, etc.) are utilized. Analyzing and characterizing these metrics would enable one to create predictive fingerprint recognition models that make it possible to verify that the remote machines are effectively executing the requested applications. One assumption we made is that the farmers are providing those resource usage metrics online for the container used to execute the customer application.

To create such predictive fingerprint recognition models, we propose a framework that is able to control the correct execution of applications in a trustless environment. Our framework is made of three components (see Figure 8.1). This chapter focuses on the *Fingerprint Tracker* and *Fingerprint Builder*.

1. The **Decision Engine** is responsible for handling customer requests which consist in executing the designated applications (1). To do so, the decision engine first verifies whether the fingerprint recognition model for the requested application is available. If not, it requests the *Fingerprint Builder* to generate one for this new application (2). Then, the Decision Engine chooses a suitable farmer that will be in charge of executing the customer application (3) [48]. Finally, it requests the *Fingerprint Tracker* to verify the correct execution of this application (4).
2. The **Fingerprint Builder** is responsible for constructing the predictive fin-

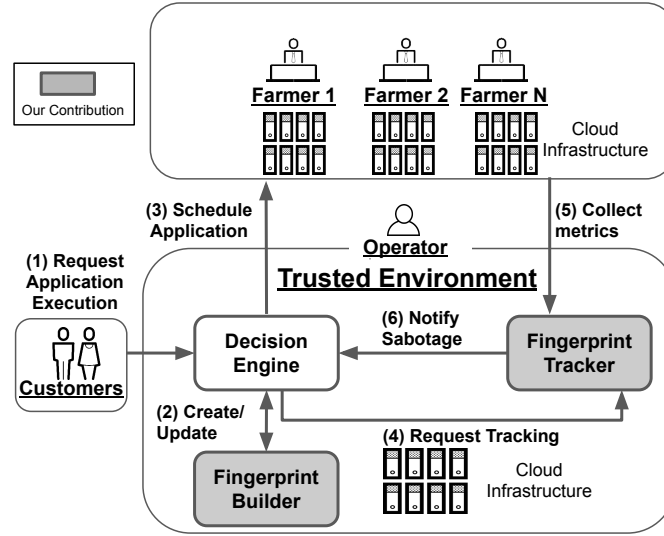


Figure 8.1: Overall approach

gerprint recognition models. To do so, this component uses an environment of trust in order to ascertain the correctness of such models (see Section 8.2.1).

3. The **Fingerprint Tracker** is in charge of controlling continuously the correct execution of applications in a trustless environment (*i.e.*, the farmer infrastructure) using the predictive fingerprint recognition models previously built by the *Fingerprint Builder*. In order to achieve that, it first collects the required execution metrics (5). Then, it identifies the application based on its fingerprint obtained via its resource usage. Finally, it compares this result with the expected application that was communicated by the *Decision Engine* to determine whether or not the application was correctly executed to trigger potential counter measurements when necessary (6) (see Section 8.2.2).

### 8.2.1 Fingerprint Builder: Building the fingerprint models in an environment of trust

This section details how the *Fingerprint Builder* constructs the fingerprint recognizer models. Figure 8.2 describes the overall approach followed with three differ-

ent steps performed in the trusted environment: Data generation step, Learning step, and Evaluation step.

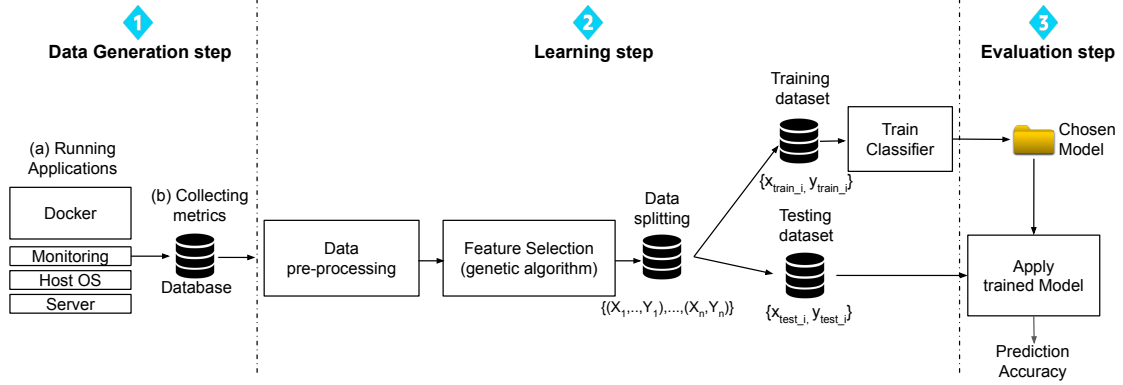


Figure 8.2: Training fingerprint models approach

### Data Generation step

In the dataset generation phase, we have mainly two steps (see Figure 8.2): generating the traces by executing different applications and collecting their respective container metrics. We have selected five applications that were deployed in a container-based environment covering various use-cases. Table 8.1 summarizes the benchmarks used.

Table 8.1: Applications and Benchmarks used

Name	Category	Description
<i>web</i>	Server application	N-tiers web application
<i>email</i>	Server application	Email server
<i>video</i>	Multimedia processing	H.264 video transcoding
<i>rendering</i>	Multimedia processing	3D rendering
<i>compilation</i>	Software build	Linux kernel compilation

To generate the dataset, we used the tools Nginx [141], MySQL [134], and WordPress [28] for the *web* application, FileBench [165] for the *email*, ffmpeg [64] for the *video* application, and blender <sup>1</sup> for the *rendering* application, GNU Compiler Collection [161] for the *compilation* application.

1. blender.org

**Server application:** We chose two typical enterprise server applications: an n-tiers **web application** (WordPress), and email servers (Filebench). WordPress is an Open Source content management system. In our setup, WordPress is deployed with Nginx, PHP, and MySQL. In the case of a WordPress website, we varied the number of concurrent readers/writers between 1 and 50. Varying the number of users has a direct impact on resource usage. The tool that generates the traffic was executed on a separate host. We used Filebench to evaluate **email** to generate a mix of open/read/write/-close/delete operations.

**Multimedia processing:** ffmpeg is a framework dedicated to audio and **video processing**. We used two videos, a FullHD (6.3 GB) and an HD (580MB) video. For the transcoding of the H.264 video, we varied the PRESET parameter between *slow* and *ultrafast*. This parameter has a direct impact on the quality of the compression as well as on the file size. Blender is a toolset for making **3D rendering**, visual effects, art, and interactive 3D applications. We used five 3D models.

**Software build:** Linux kernel compilation uses thousands of small source files. Its compilation demands intensive CPU usage and short intensive random I/O operations to read a large number of source files and write the object files to the disk. For the sake of our study, we compiled the Linux kernel 4.2.

## Data Pre-processing

The goal of the pre-processing step is to create the matrix of input features noted  $x$  and the vector of the observed labels noted  $y$  (*i.e.*, the running application) from the traces stored in the time-series database. The selection of the input features  $x$  is a key step to build a good predictive fingerprint model. One needs to consider the variables that have an influence on application fingerprints for the learning algorithms to find the (hidden) relationships between  $x$  and  $y$  (see chapter background).

## Feature Extraction

In a container environment, there are more than 50 collected metrics such as active files, CPU usage, I/O async, mapped file, pgpgout. This large number of



potential features does not allow for an exhaustive search [96]. According to [43] a good feature selection algorithm can be used based on the following considerations: simplicity, stability, number of reduced features, classification accuracy, storage, and computational requirements. According to [84], PTA(l,r), GPTA(l,r), Sequential Feature Selection (SFFS), and genetic algorithm perform well for such a task.

We used a genetic algorithm to derive a combination of features that maximizes the accuracy of the fingerprint recognition model. Genetic Algorithms (GA) are stochastic optimization methods that mimic the process of natural evolution [180]. In GA, a population is composed of individuals. An individual is a potential solution of the optimization problem (*i.e.*, the selection of the best features for detecting the running application). Individuals can be scored using at least one fitness function (FF). In our study, the individual is composed of a vector of 1 and 0 indicating whether or not the feature (*i.e.*, the metrics) is selected and the fitness function which is the accuracy classification score calculated by counting the number of correct classifications and divide it by the total number of samples. At each GA step, individuals from a generation of a population mutate using two-point crossover to generate new individuals who inherit from both parents on which random flip could be applied (*i.e.*, a previously selected feature can become unselected). Then, through the fitness function, the best children of the new generation (*i.e.*, those who maximize the classification score) are selected to produce the next generation. Finally, after 100 generations the selected features are then used in a classification process with a Random Forest (RF) classifier. We selected RF algorithm based on the following criteria [69]:

Computational complexity: estimation techniques should not have high overheads in terms of time and computing resource requirements as compared to the potential reclaimable resources.

Robustness to outliers: we are concerned about outliers as on average most Cloud applications do not use the whole available performance of the devices.

Handling of missing values: The large number of possible combinations of workloads require a learning algorithms that can handle missing values.

## 8.2.2 Fingerprint Tracker: tracking application executions

This section details how the fingerprint tracker leverages the fingerprint recognition models in order to ascertain both remotely and online whether or not a sabotage is taking place.

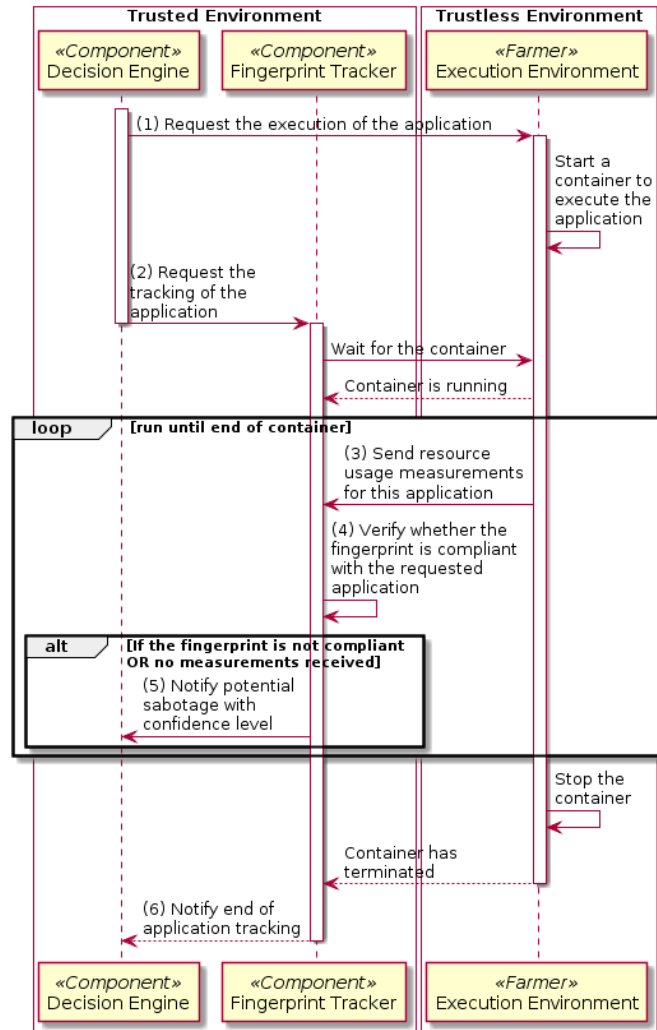


Figure 8.3: Sequence diagram of the interaction between the trusted environment and the trustless environment

In Figure 8.3 we highlight the interactions that occur between the trusted (*i.e.*, Operator) and the trustless environment (*i.e.*, Farmer) in order to track the correct execution of the customer applications. To do so, the methodology is the following. First, the operator requests the execution of an application to a farmer (*i.e.*, a trustless resource provider) (1), which subsequently triggers the creation of a

container that will host the execution of the application. Then, this designated farmer is asked to supply, every second, the resource usage measurements of the machine that is used to execute the customer application within a predefined time interval (3). These measurements are then ingested by the *Fingerprint Tracker* to detect the correctness of the execution of the application (4). If either the resource usage measurements are not delivered in a timely fashion or the *Fingerprint Tracker* detects that the fingerprint is not complying with the expected one for a duration of at least 2 minutes (this duration can be adjusted based on the desired confidence level as explained in Section 8.3), then the *Fingerprint Tracker* considers that there is a sufficiently high likelihood that a sabotage has taken place during this time frame. In such a case, the *Fingerprint Tracker* notifies the *Decision Engine* of a potential sabotage with the associated confidence level (5) so that it can trigger countermeasures, such as spot-checking with black-list [148]. Finally, upon completion of the application and after its hosting container has stopped, the *Fingerprint Tracker* ends its tracking and notifies it to the decision engine (6).

## 8.3 Evaluation

This section describes the obtained results. Through this experimental part, we try to answer 4 research questions (RQ):

- **RQ1:** What are the best features for tracking application fingerprint?
- **RQ2:** What is the accuracy of the fingerprint Tracker for the three use cases: *homogeneous*, *heterogeneous*, and *unspecified* hardware?
- **RQ3:** How does the accuracy change with regards to the size of the training dataset (learning curve)?
- **RQ4:** What is the minimum period of monitoring required?

### 8.3.1 Experimental setup

We used four heterogeneous physical machine. Table 8.2 describes the hardware characteristics of machines used by the farmers. We used two DELL server configurations that are very common in data center infrastructures and two uncommon configurations (a laptop and an embedded board).

We made use of Python with the *scikit-learn* [139] version 0.18 library which provides state of the art machine learning algorithms. Besides, all training and forecasts done by the Operator were performed on servers with an Intel(R) Xeon(R) E5-2630 v2 CPU clocked at 2.60GHz and with 130GB of RAM. In our experiments, we used five applications: video processing, 3D rendering, email server, software development, and web application, which are detailed in Section 8.2.1. We used the Ubuntu 14.04 LTS GNU Linux distribution with a kernel version 4.2 for M1, M2, and M3, and for M4 a kernel 4.14. The virtualization system used was Docker version 18.06. Finally, we have experimented with 4 heterogeneous physical machines in terms of CPU performance and architecture, and storage (*i.e.*, SSD or HDD) to explore the fingerprint accuracy as compared to the used hardware.

Table 8.2: Farmer physical machines

ID	CPU	Memory (GB)	Storage
<i>M1</i>	Quad core Intel Core i7-4900MQ	15	Samsung Evo 850
<i>M2</i>	Hexa core Intel Xeon E5-2630	130	Intel Solid-State Drive 750
<i>M3</i>	Hexa core Intel Xeon E5-2630	130	Samsung 960 Pro
<i>M4</i>	ARM Cortex-A53	1	Kingston microSDHC

#### RQ1-Selected features

Our approach uses a genetic algorithm to select a subset of the monitored metrics to be used to efficiently train the fingerprint models. For the 5 applications, it emerged that among the 48 metrics the GA method has selected a total of 5 features for homogeneous hardware (see Table 8.3) and 13 features for heterogeneous hardware (see Table 8.4) for all the applications. We observed that they are mainly related to CPU, memory and storage usage. These results show that a set of 48 metrics commonly used in Cloud technology could be utilized to classify a range of real applications without any assumption.

Table 8.3: Selected features for homogeneous hardware

Name	Description
active-anon	Anonymous memory that has been used more recently
pgpgin	Number of kilobytes the system has paged in from disk per second.
I/O write and sync	Number of I/O operations
write-bytes	Bytes written per second to disk

Table 8.4: Selected features for heterogeneous hardware and unspecified hardware

Name	Description
cpu-usage	Percentage of CPU utilization
active-anon	Anonymous memory that has been used more recently
inactive-anon	Bytes of anonymous and swap cache memory on inactive LRU list
pgpgin	Number of kilobytes the system has paged in from disk per second.
pgfault	Number of page faults the system has made per second
active-file	Bytes of file-backed memory on active LRU list.
I/O read, write and sync	Number of I/O operations
mapped-file	Bytes of mapped file (includes tmpfs/shmem)
read-bytes	Bytes read per second from disk
write-bytes	Bytes written per second to disk
writeback	Bytes of file/anon cache that are queued for syncing to disk.

## RQ2-Accuracy

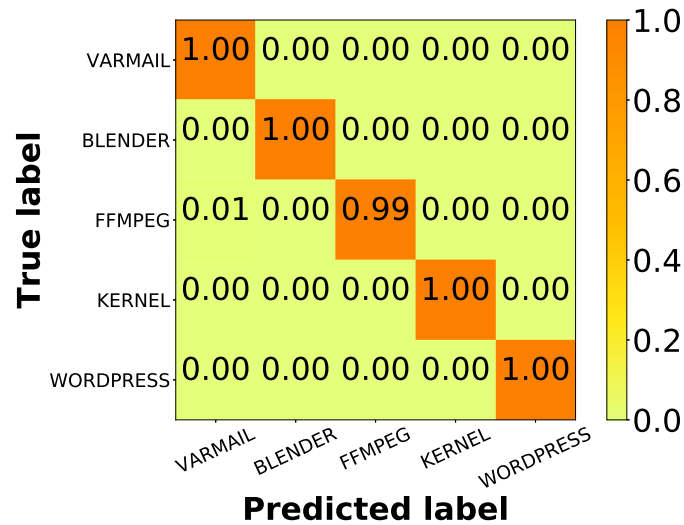


Figure 8.4: Confusion matrix with homogeneous hardware

The confusion matrix shown in Figure 8.4 was built as follows: we ran each application 50 times by randomly selecting each time 70% of the dataset (comprising all the applications) to build the model and the remaining 30% to evaluate its accuracy for a given hardware architecture. For each execution, we have fixed the hardware architectures and assumed that on the trustless side, the hardware

used was the same (*i.e.*, the homogeneous hardware case). We observed that the resulting predictive fingerprint recognition model was very accurate and succeeded in distinguishing between the 5 applications with an accuracy of 99.88%.

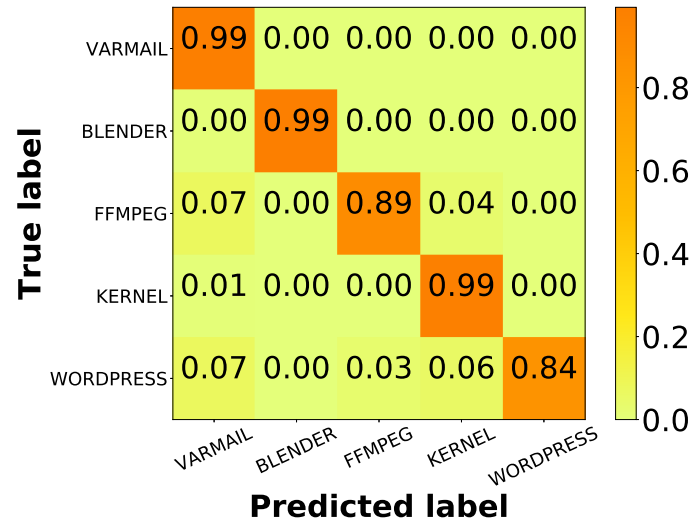


Figure 8.5: Confusion matrix with heterogeneous hardware

Figure 8.5 follows the same methodology as the previous experiment but for the *heterogeneous hardware* case (*i.e.*, four hardware architectures were combined in a unique dataset so that an application could be run on very different hardware on the trustless side). We remark that WordPress was the most inaccurate application to track with an accuracy of 84%.

Figure 8.6 shows the confusion matrix for five applications in case of *unspecified hardware*. The accuracy is evaluated as follows: during the training the hardware M1, M2 and M3 are used and then M4 is used for testing. We have chosen the M4 hardware for the test because it is the most different in terms of hardware characteristics. The goal is to be able to evaluate the impact on the fingerprint recognizer accuracy in the (extreme) unspecified and different (*i.e.*, ARM processor) hardware case. We observe that compared to the heterogeneous hardware case, the accuracy drops to about 40%. This result means that the application fingerprinting technique may not be relevant when the hardware used for the training is too different from the one used for the test.

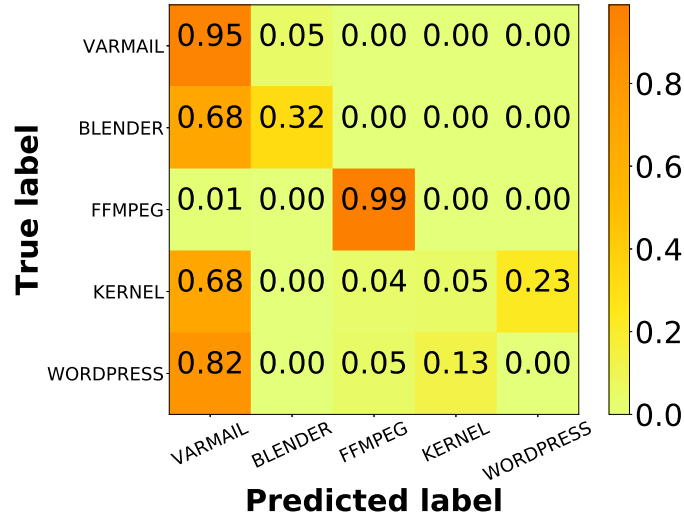


Figure 8.6: Confusion matrix on unspecified hardware

### RQ3-Learning curve

The learning curve shows the evolution of the model accuracy according to the number of training samples [10, 85]. In order to build our learning curve, we performed a progressive sampling by increasing the dataset sizes  $N_{training} = 1$  to  $N_{max}$  with a step of 1 second.  $N_{max}$  is the total number of samples available.

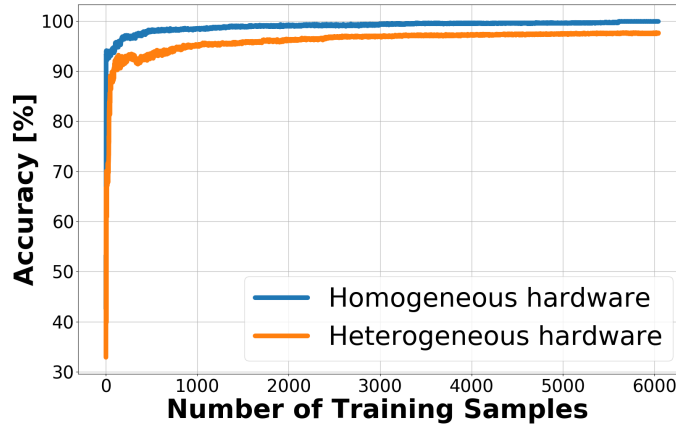


Figure 8.7: Learning curves on the testing set as a function of the number of training samples

In Figure 8.7 we show the accuracy of the algorithms according to the training set size. First, we observe, as expected, that the accuracy improves with the increase of the training set size. In case of homogeneous hardware, we observe

that with 3000 samples (*i.e.*, 5 minutes) the accuracy reaches 99.95% and 100% with 5500 samples. As compare to heterogeneous hardware, we notice that we need about 3600 samples (*i.e.*, 60 minutes) of application trace to be able to distinguish between the 5 five applications with an accuracy of 97% and it reaches 98% with 5700 samples. Moreover, after about 100 minutes the accuracy does not increase anymore.

#### RQ4-Monitoring Interval

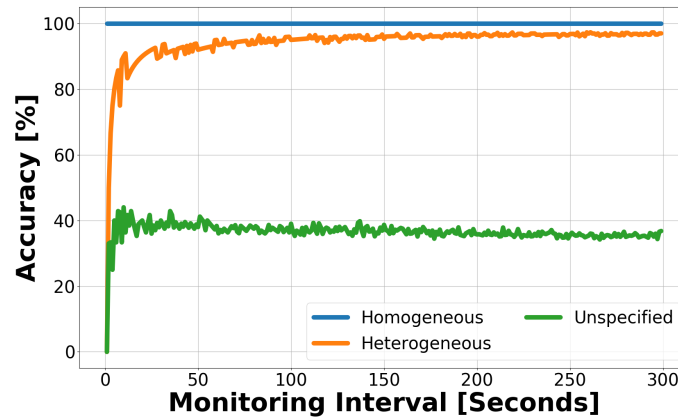


Figure 8.8: Accuracy curves as function of the number of testing samples

In Figure 8.8, we show the size of the monitoring interval in seconds used to predict the running application with regards to the accuracy for the three use cases: homogeneous, heterogeneous and unspecified. We observe that only 1 second is required to achieve 100% accuracy in case of homogeneous hardware. In contrast, we notice that 60 seconds are needed to reach an accuracy of 98% for heterogeneous hardware. Finally, in the unspecified hardware, we observe that the accuracy is capped to 40% and does not improve after about 40 seconds. This means that in case of unspecified hardware, training a model with a set of metrics shows some limitations. It would be interesting to look at metrics that are not hardware sensitive such as system calls issues from applications.



## 8.4 Discussions

This application fingerprinting mechanism based on consumed resources can be considered as a second level of security that can be coupled with other existing approaches already proposed in the state of the art studies. Indeed, it may be difficult for a malicious user to cheat on both the output format of the application and the associated stream of measurements that lead to this incorrect result.

In addition, to prevent a saboteur from saving the application usages (*e.g.*, cpu usage) and then sending these metrics to the operator, three actions could be implemented:

1. First, the decision engine scheduler may try to avoid to schedule the same application to the same farmer several times.
2. Second, we propose to use the technique of *Proof of Storage* [81] to ensures that the data is actually stored in the trustless environment.
3. Third, the application binary could be obfuscated to prevent potential reverse engineering by analyzing the binary.

The poor accuracy of fingerprint recognizer in the *unspecified hardware* case is not a surprise. Indeed, for example, when an application is executed on a physical machine with a large volume of memory buffer, the operating system may delay disk write operations and thus improve the performance [104]. This could prevent the ML algorithm from identifying the relationship between the application and the selected feature metrics (*e.g.*, utilization of the memory, write back). A consequence, it may fail in identifying the application.

There are several parameters that could affect the accuracy of the model. In the performed experiments, we did not test the sensitivity of the model to changes in kernel parameters. It may be relevant to evaluate the kernel configuration parameters that have a significant impact on model accuracy. Container resource allocation can be configured at runtime using the CGroup configuration interface <sup>2</sup>. As with the static configuration parameter of the kernel, it may be relevant to assess the impact of such a dynamic evolution on the model accuracy. In addition, the virtualization technique used such as Docker, Lxd, Qemu could also have an impact on the model accuracy.

---

2. <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>

Moreover, the use of a different version of the same application or the modification of the application parameters may also induce a change in the estimated model due to a change in its behavior. We did not consider how co-located application workloads may interfere on applications. We have considered only a fixed capacity per host, while the capacity may depend on all running applications [49]

Finally, monitoring may also affect the estimated model. Indeed, a modification of the implementation of the monitoring component could also affect the model accuracy (e.g., the CPU sampling differs between the training and the testing phase).

## 8.5 Summary

Tracking the correctness of the application execution over time is necessary to prevent malicious infrastructure owners from sabotaging the computation. Machine learning combined with a fingerprint technique seems to be a relevant approach for homogeneous and heterogeneous hardware. It also shows that the approach is not viable for unspecified hardware.

This contribution shows that it is not necessary to take into account application characteristics when trying to track the execution of applications when using our fingerprinting approach that combines both a genetic and a machine learning algorithm.

We evaluated our approach with RF. Our results show that we were able to detect the correct execution with an accuracy of 99.88% with homogeneous hardware, 98% with heterogeneous and 40% with unspecified hardware on the five selected applications.

# AN ARCHITECTURE IMPLEMENTATION TO LEVERAGE CLOUD UNUSED RESOURCES

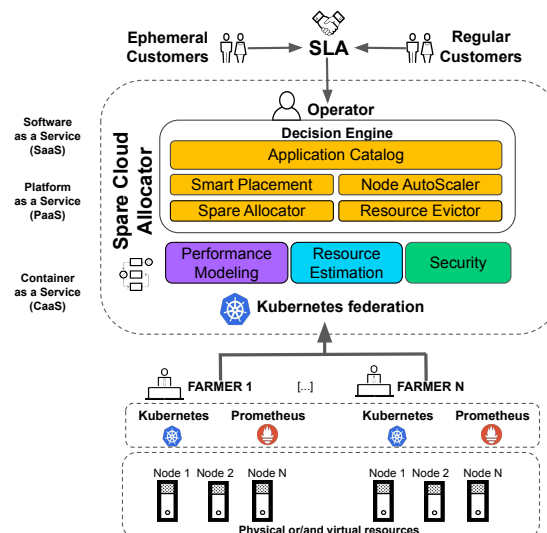


Figure 9.1: Architecture overview

In this chapter, we give more details on the proposed architecture. Our architecture is based on the shelves solutions (e.g., Kubernetes, Prometheus, Hadoop, Apache Spark) to leverage Cloud unused resources. An overview of the architecture is depicted in Figure 9.1. The first level (i.e., at the bottom of figure) is responsible of managing the underlying physical infrastructure made up of N nodes that are supervised by kubertenes or any given **laaS** such as OpenStack, VMware vSphere. Then, these physical or/and virtual resources are aggregated with a Kubernetes federation. A Kubernetes federation allows us to coordinate the configurations of multiple Kubernetes clusters. Each Kubernetes cluster of a Farmer

is monitored using the Prometheus solution. Prometheus is an open-source monitoring and alerting solution <sup>1</sup>. For deploying this solution in our context, a proxy has to be developed to avoid buying unnecessary hardware. Indeed, most organizations are using capacity planner such as DC Scope <sup>2</sup> to determine if they need to purchase new and more efficient hardware. The second level is composed of eight modules (*i.e.*, Application Catalog, Smart Placement, Spare Allocator, Resource Evictor, Performance Modeling, Resource Estimation, Security, and Node AutoScaler). There are two ways of deploying these modules. The core modules are all the services that are essential for operating the solution. These modules have to be deployed onto dedicated resources. In contrast, ephemeral modules can be interrupted. Most part of the ephemeral applications are deployed on ephemeral resources. However, most of the architecture components are deployed as core modules except the resource estimation and performance modeling modules. We give below additional details on the implementation:

The **Spare allocator** is a module that assigns and shares Cloud unused resources between applications of *ephemeral customers*. To achieve that, we

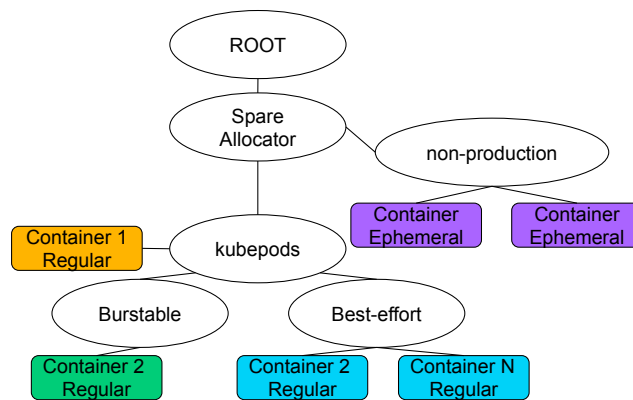


Figure 9.2: cgroups CPU hierarchy

introduced a new class of QoS in Kubernetes called *non-production* (*i.e.*, about 1000 lines of code added in the kubertenes source code). This class aims to avoid any interference on regular workloads and even *best-effort* ones. This QoS class is automatically reclaiming unused resources for allocating them to applications of ephemeral customers. The non-production

1. <https://prometheus.io>  
2. <https://www.easyvirt.com>

QoS class relies on a specific configuration of the Linux kernel module called *cgroup*. The *cgroup* functionality offers the possibility to limit and prioritize resource usage (e.g., CPU, block I/O, network, etc.) for each container (see Background Chapter). In the case of CPU, *cgroups* allows controlling the amount of time a group spends in CPU (i.e., quota) on a specific period. The amount of CPU time provided to the group (i.e., non-production) depends on the state of neighboring groups. CPU time within the same group is shared towards the container(s).

Figure 9.2 shows our configuration of *cgroups* for the CPU. The same configuration is applied for I/O and network resources. The *root* and *spare allocator* groups receive all the amounts of available processor time. Then, *non-production* group is configured for receiving only the amount of available processor time that is not consumed by the *kubepods* group. Then, the configuration is the same as the default Kubernetes (i.e., the child container(s) of *kubepods* group in yellow belongs to the guaranteed class).

The **Resource Evictor** implements a mechanism that reacts to underestimation of used resources from the *Resource Estimation module*. This is critical in a context of incompressible compute resources, such as memory or disk space (see Background chapter). Note that the implementation of this mechanism relies on a feature provided by Kubernetes *kubelet* module (i.e., kubelet supports eviction decisions based on signals). This module relies on the contribution described in chapter 7.

**Resource Estimation** aims to predict resource volatility. In particular, it can estimate available resources for running applications of *ephemeral customers*. Estimation of resources is achieved for each node. To achieve that, this component is deployed as a Kubernetes daemonset (i.e., a daemonSet ensures that all eligible nodes run a copy of a Pod), this means that each node is in charge of training its own model). A clear distinction is done between what is consumed by the farmer (i.e., regular consumers workloads) for its own needs and what is consumed by the ephemeral customers workloads. To achieve that, we used dedicated Kubernetes *namespaces* (see Chapter Background) that allows us to filter resources consumption between the Farmer and the ephemeral customer. In chapter 6, we have shown that

quantile regression may increase the amount of savings by up to 20% compared to traditional approaches. In this architecture, we applied the same configuration.

In this chapter, we presented an overview of an architecture based on the shelves components (.e., kubernetes) to leverage cloud unused resources. We observe that this architecture provides a first step to reclaim unused resources with limited changes. A demonstration version was developed and details provided in French in the 'Meetup Machine Learning Rennes' (see video: <https://www.youtube.com/watch?v=UnHIUvNz27Y>).

PART III

# **Conclusion & Perspectives**





---

---

## Conclusion

Managing efficiently resources and reducing costs are major concerns for Cloud providers. Although the use of virtualization has improved the use of computing resources in data centers [130], several studies have demonstrated that the average usage of resources remains low. To address such an issue, making profit of those unused resources appears to be a very interesting solution to optimize the total cost of ownership.

This thesis aims to make unused and heterogeneous private IT resources available through a secured distributed Cloud to deploy applications at a cheaper price. The first use case of the thesis is to provide a framework that leverages unused Cloud resources to deploy big data applications.

To achieve that, six challenges were identified: heterogeneity, connectivity and interoperability between the farms, volatility of resources, avoidance of interferences between ephemeral customers and regular customers' workloads, and security. This thesis aims at addressing four out of the six challenges (*i.e.*,  users SLA guarantee,  resources volatility,  Cloud heterogeneity, and  security). To address these challenges, this thesis stated four main problems:

### **Problem 1 (Real system capacity estimation): How to model performance variations?**

We designed SSD performance models that take into account interactions between executed processes/containers, the operating system and the SSD. This model is used to prevent bad I/O interferences scenarios (*i.e.*, SLA violations). Our machine learning-based framework succeeded in modeling I/O interference with a median NRMSE of 2.5%.

### **Problem 2 (Future use estimation): How can we estimate, in a flexible and accurate manner, future resources utilization?**

We proposed a predictive model that estimates the future use of resources. One of the key contributions is the use of quantile regression to make our predictive model flexible for the CP, rather than using the simple mean regression of resource usage. This enables a CP to make relevant and accurate trade-off between the volume of resources that can be leased and the



---

risk in SLA violations. Our approach can increase the amount of savings up to 20% compared to traditional approaches.

**Problem 3 (Ephemeral-aware applications adaptation): How big data applications can be adapted to run on ephemeral heterogeneous resources?**

We designed an approach that relies on three mechanisms: *i*) a Data placement planner to cope with Cloud heterogeneity, *ii*) a Forecasting builder to predict resource volatility, and *iii*) a QoS controller to ensure users SLA guarantee by avoiding interferences. The experimental results show that our approach divides Hadoop job execution time by up to 7 when compared to the standard Hadoop implementation.

**Problem 4 (Malicious farmers prevention): How can we prevent malicious infrastructure owners from compromising the computation?**

We proposed to analyze and characterize a set of metrics to create predictive fingerprint recognition models that make it possible to verify that the remote machines are effectively executing the requested applications. When running these applications on untrusted machines (with either homogeneous, heterogeneous or unspecified hardware from the one that was used to build the model), the fingerprint recognizer was able to ascertain whether the execution of the application is correct or not with a median accuracy of about 98% for heterogeneous hardware and about 40% for the unspecified one.

This thesis presented a framework to leverage Cloud unused resources while achieving SLA. We showed that unused resources can be used to deploy applications in particular big data jobs at a low cost without compromising on quality and security. Our thesis shows that the approach and associated tools can be currently deployed in an industrial context. A demonstrator was developed that integrates most of the problems described in the thesis. The approach presented in this thesis is being integrated into an industrial version (namely b<>com \*Spare Cloud Allocator\*) currently under development at IRT b<>com (see the post on orange blog: <https://oran.ge/32qfUK2>).

It has to be noted that while technological progress can improve resource utilization efficiency on the short term (*e.g.*, decreasing the amount of resources for

---

the same task), it is observed paradoxically that this may lead on the longer term to an increase of resources overall consumption [99]. This means that, besides our thesis provides a path to resource optimization in a near future, the answer for sustainability of data centers cannot just be technological.

## Perspectives

We now discuss various perspectives of this research. Firstly, we focus on a number of additional investigations that could be done to enhance the proposed solutions of this thesis (see Figure 9.3). Secondly, we propose new long term research directions/improvements beyond the proposed contributions.

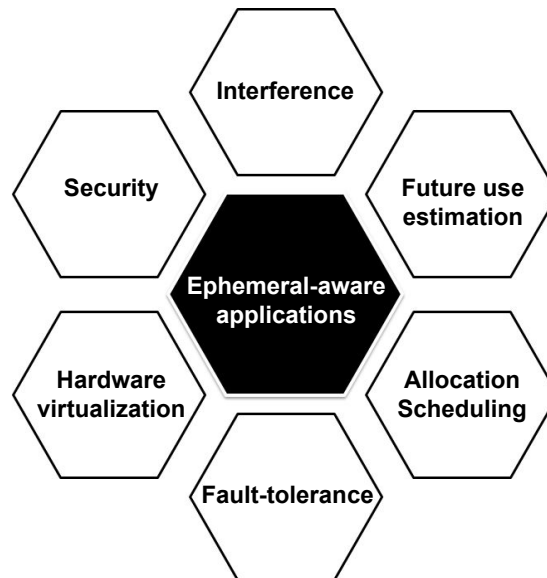


Figure 9.3: Perspectives

**Interference:** Our evaluation results have highlighted the importance of considering I/O interferences. However, it would be interesting to extend this work to other types of metrics (*e.g.*, CPU, memory, network) and units (*e.g.*, latency, energy) For example, interference in the shared last-level cache (LLC) seems to be essential for avoiding performance degradation on regular and ephemeral customers [166, 97]. It could also be interesting to consider new features to train our model such as the number of invalid blocks, file-system

---

aging, CPU, memory. While in our contribution we limited the experiments to 5 containers, a perspective could be to add more containers and to evaluate the potential limitations. It would also be relevant to be able to identify the container or virtual machine at the root of interference in order to isolate or stop those. Indeed, interference could be used as a vector of attack for malicious users. Many questions still remain open: can an attacker push/set/configure the system in a particular way that is generating a strong interference? Can we develop mechanisms to avoid such scenarios?

**Future use estimation:** Cloud time series are continuously updated. It would be interesting to use online learning (*i.e.*, refine the learning model step by step following the input time series data), also called incremental learning. Our question would be: Can we leverage online learning algorithms for improving Cloud time series forecast? Another improvement could be to consider Multi-Input Multi-Output (MIMO) strategies for discovering correlations among the metrics (*e.g.*, CPU and memory) in order to improve the accuracy. It would be promising to extend the model to consider more features, hardware such as GPU and metrics such as network latency / disk latency, and specific interference metrics. It would be also relevant to predict more metrics in our model such as GPU usage or network latency.

**Allocation/Scheduling:** it would be interesting to consider a wider set of criteria. For example, ephemeral customers could specify if they want to minimize the price even if the computation takes a longer time or if they want to improve performance at a higher cost. Indeed, the workload deployed for ephemeral customers who wish to reduce the price can be throttled or killed first for dealing with unpredictable workloads, and let workloads with higher prices run first. It would be also essential to consider that the capacity can be reduced due to interference between co-located workloads.

It would also be relevant to consider hardware wear out (*e.g.*, SSD) and energy costs due to reselling in the scheduling. It would also be interesting to consider how the ephemeral customers are sharing the Cloud unused resources: do we need to share proportionally Cloud unused resources among all ephemeral customers or to be priority or cost aware? Also, we envision to use reinforcement learning to decrease the impact of unused

---

resources volatility on QoS by including in our model reserved resources. We would expect by this approach to reach a compromise between costs management and impact on QoS without prior knowledge in a dynamic environment. Finally, we mainly used simulations and it would be essential to evaluate our strategies using a testbed for experimental research, such as Grid5000.

**Fault-tolerance:** Unpredictable workloads are inevitable. Available fault-tolerance mechanisms may have to be used or revisited: check-pointing, migration, erasure coding vs replication for our environment. For example, check-pointing fault-tolerance techniques could be used to avoid using reserved resources for hosting intermediate data due to evictions by finding a trade-off between the network overhead and the re-computation costs by choosing carefully the high volatile hosts.

**Hardware virtualization:** While the thesis focuses on containerized virtualization, the proposed methodology may be applied to applications running on baremetal, VMs or containers, katacontainers or edges devices. However, in both cases, the selection of the instance type remains a challenge. Indeed, in this thesis we decided to rely on a fixed instance type. However an efficient selection of the instance type should include an evaluation of application needs and of monthly/daily variations. Our question is : How to choose the right instance type for an application subject to resource volatility in order to find the best trade-off between cost/performance?

**Security:** Our approach based on resource consumption metrics faces some limitations regarding heterogeneous and unspecified hardware. Indeed, there are several other scenarios such as hardware tampering, software configuration tampering overcommitment of the resources, or noisy neighbors. It would be interesting to look at metrics that are not hardware sensitive such as system calls issued from applications. It would be also relevant to evaluate deep learning algorithms such as glslstm which is designed to capture dependencies within an input sequence and Generative Adversarial Network (GAN). Also, the prediction ability for *unspecified hardware* could be improved by normalizing the performance metrics using hardware information provided for example by sysconf. In addition, during the feature

---

selection step, we could also add in the testing set *unspecified hardware* to let the feature selection algorithm select features more robust to hardware variations. Finally, it would be interesting to apply one-class (*i.e.*, unary classification) to train a model per application.

**Ephemeral-aware applications adaptation:** We mainly worked on customizing Hadoop to be ephemeral aware. However, many other improvements can be achieved. For example, Hadoop has three important concepts: data locality, job and task scheduling. In this thesis, we have considered task scheduling based on through data locality with CPU forecast, a solution that can be improved. Indeed, this may lead to slow execution and poor resource utilization as we have shown in [83]. Our contribution provides a Holistic task and job scheduler with three different solving strategies that rely on future resources predictions (*e.g.*, CPU, RAM). In addition, a scheduler-based data placement strategy is used to improve the locality of data. Finally, a reactive QoS controller, considering compressible and incompressible resources, was proposed.

It could also be interesting to evaluate other types of applications or frameworks such as TensorFlow or Apache Flink. Moreover, most applications do not have any mechanism that would alert if a node is going down/killed, or that performance is decreasing that would for example trigger specific actions such as check-pointing or migrations.

## Focus on security aspects

Thanks to the flexibility offered by different Cloud approaches, these are now used to deploy a wide variety of applications such as video coding/decoding, virtualization of network functions but also training machine learning algorithms. One of the challenges for users is to guarantee code and data confidentiality, integrity and user privacy.

An approach to deal with these attacks is hardware enclaves such as Intel Software Guard Extensions (SGX) [17] or ARM's TrustZone [175]. These enclaves allow executing software on a remote computer owned and maintained by an untrusted party, with some integrity and confidentiality guarantee [46]. To achieve that, the enclaves allow to transfer encrypted data and code from a

---

trusted computer to a remote computer. Then, in a dedicated hardware part of the processor of the remote computer, the data and the code are decrypted. The owner of the remote computer or the administrator of the operating system, or the hypervisor, cannot access the data.

In this thesis, we have shown that it is possible to verify that the remote machines are effectively executing the requested applications based on a set of metrics. One question that could be asked is whether the same approach could be implemented on hardware enclaves in order to determine which application is running. Indeed, the enclaves give the attacker the possibility to collect any available metrics (*e.g.*, hardware counter, energy consumption) for building a fingerprint model.

The second question is then how to avoid such scenario: How to improve hardware enclaves to ensure user privacy, data confidentiality and integrity? One solution could be to inject random instructions to blur information on the running applications. In this case a trade-off between confidentiality and the cost of execution of fake instructions has to be found. One way to achieve that could be to use a Generative Adversarial Network algorithm. In this context, the generator would be in charge of generating fake instructions and the discriminator would discover what is the running application.

## **Long-term perspectives: How can applications be natively ephemeral aware?**

In this thesis, we defended that applications must be adapted to be ephemeral aware. These adaptations are complex and in most cases require a deep understanding of how applications are implemented. One solution to bridge the gap between design and deployment in volatile and heterogeneous environments would be to propose a programming language that provides support for making applications natively ephemeral-aware. For example, the programming language could include annotations that would notify developers that a resource will be temporarily unavailable. Indeed, most applications do not have any mechanism that would notify that a node is going down, in order to avoid interferences on regular customer applications. These notifications can be used to trigger actions such as check-pointing or memory states migrations. In addition, to deal with unpre-

---

dictable workloads, a solution could be to create an annotation that would enable the use of approximate computing at hardware and software levels to reduce computational costs.

PART IV

# Indexes

---



# LIST OF FIGURES

---

1	Projet global	15
2	Défis du projet IRT b<>com	16
3	Une carte des problèmes et des défis	17
1.1	Overall project	26
1.2	Box plots of (a) <i>CPU</i> and (b) <i>RAM</i> usage for each host with <i>Private Company 1</i>	28
1.3	IRT b<>com project challenges	30
1.4	The problems addressed	31
2.1	Cloud Computing service models	40
2.2	Cloud Unused Resources	42
2.3	Service Models	43
2.4	hardware-level virtualization (left) vs. operating system-level virtualization (right)	44
2.5	Hardware virtualization Memory overcommitment	47
2.6	Resource Management	48
2.7	MAPE-K Management	50
2.8	Architectural overview of OpenStack	52
2.9	Architectural overview of Kubernetes	53
2.10	The basis functions $\max(0, x - t)$ and $\max(0, t - x)$ used by MARS	59
2.11	Cloud Computing service models	62
3.1	A map of problems and associated approaches	68
3.2	Problem 1 (Real system capacity estimation)	69
3.3	Simplified architecture of a NAND flash memory chip from [27]	70
3.4	Problem 2 (Future use estimation)	72
3.5	Ephemeral-aware applications adaptation	76
3.6	Problem 4 (Malicious farmers prevention)	78
4.1	PhD Overview	85

---

5.1	I/O performance of random writes for 4 SSDs . . . . .	89
5.2	I/O Interference of mixed workloads . . . . .	90
5.3	MAPE-K . . . . .	92
5.4	Overall Approach . . . . .	93
5.5	Box-plot of NRMSE for each algorithm on all SSDs. . . . .	101
5.6	Learning curves on the testing set as a function of the number of training samples . . . . .	103
5.7	Feature importance . . . . .	104
5.8	Median computation time used for the training of different learning algorithms . . . . .	105
6.1	Forecasting of six hours of CPU with: (a) The conditional Mean curve in black, (b) Five different quantile regression curves. . . . .	111
6.2	Overall Approach . . . . .	112
6.3	Aggregated Potential Cost Savings . . . . .	118
6.4	Aggregated cost violations for <i>Private Company 1</i> when there is no exhaustive SLA metrics awareness ( <i>i.e.</i> , only CPU) . . . . .	121
7.1	Hadoop architecture . . . . .	128
7.2	Overview of the Cuckoo architecture . . . . .	131
7.3	Job execution time for standard Hadoop and Cuckoo . . . . .	137
7.4	Median percentage relaunched tasks comparison between Hadoop and Cuckoo . . . . .	138
7.5	Median percentage remote tasks comparison between Hadoop and Cuckoo . . . . .	139
8.1	Overall approach . . . . .	145
8.2	Training fingerprint models approach . . . . .	146
8.3	Sequence diagram of the interaction between the trusted environ- ment and the trustless environment . . . . .	149
8.4	Confusion matrix with homogeneous hardware . . . . .	152
8.5	Confusion matrix with heterogeneous hardware . . . . .	153
8.6	Confusion matrix on unspecified hardware . . . . .	154
8.7	Learning curves on the testing set as a function of the number of training samples . . . . .	154
8.8	Accuracy curves as function of the number of testing samples . . . .	155

---

9.1	Architecture overview	158
9.2	cgroups CPU hierarchy	159
9.3	Perspectives	165

# LIST OF TABLES

---

1.1	Hosts characteristics of private company 1 . . . . .	27
1.2	Available aggregated $Cap(t, mtr)$ of the data centers . . . . .	29
1.3	Average usage of resources calculated at the host-level . . . . .	29
2.1	Comparison between hardware virtualization and OS virtualization	48
2.2	Some characteristics of the learning methods used [85]. . . . .	57
3.1	Summary of economy class solutions . . . . .	67
3.2	Summary of performance modeling and I/O interference . . . . .	72
3.3	Summary of Cloud time series forecast strategies . . . . .	75
3.4	Summary of opportunistic mapreduce on ephemeral and hetero- geneous Cloud resources . . . . .	78
3.5	Summary of sabotage-tolerance mechanisms . . . . .	80
5.1	Applications and benchmarks used . . . . .	94
5.2	Sample of I/O requests stored in the time series database . . . . .	95
5.3	Pre-processed data, $X$ : Inputs (features) and $Y$ : Output . . . . .	98
5.4	Measured workload characteristics . . . . .	101
6.1	Discount applies in case violations for a 24-hour window . . . . .	117
6.2	Potential Cost Savings with regards to $\tau$ for all datasets . . . . .	120
6.3	Median ( $M$ ) and interquartile range ( $IQR$ ) of $NMQE$ for all fore- cast models and all hosts with 0.9 quantile level with <i>private com- pany 1</i> dataset. . . . .	122
6.4	Median computation time used for the training and forecast 24 hours for one host. . . . .	123
7.1	Total capacity of each data center . . . . .	136
8.1	Applications and Benchmarks used . . . . .	146
8.2	Farmer physical machines . . . . .	151

---

8.3	Selected features for homogeneous hardware . . . . .	152
8.4	Selected features for heterogeneous hardware and unspecified hardware . . . . .	152

# BIBLIOGRAPHY

---

- [1] *8 surprising facts about real docker adoption*, <https://www.datadoghq.com/docker-adoption>, 2018 (cit. on p. 94).
- [2] Giuseppe Aceto et al., “Cloud monitoring: A survey”, in: *Computer Networks* 57.9 (2013), pp. 2093–2115 (cit. on pp. 49, 143).
- [3] Anurag Acharya, Guy Edjlali, and Joel Saltz, “The utility of exploiting idle workstations for parallel computation”, in: *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, 1, ACM, 1997, pp. 225–234 (cit. on p. 65).
- [4] Orna Agmon Ben-Yehuda et al., “Deconstructing amazon ec2 spot instance pricing”, in: *ACM Transactions on Economics and Computation* 1.3 (2013), p. 16 (cit. on p. 67).
- [5] D. Agrawal et al., “Trojan Detection using IC Fingerprinting”, in: *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 296–310 (cit. on p. 79).
- [6] Nitin Agrawal, Andrea C Arpaci-Dusseau, and Remzi H Arpaci-Dusseau, “Towards realistic file-system benchmarks with CodeMRI”, in: *In SIGMETRICS*, ACM 36.2 (2008), pp. 52–57 (cit. on p. 71).
- [7] Sungyong Ahn, Kwanghyun La, and Jihong Kim, “Improving I/O Resource Sharing of Linux Cgroup for NVMe SSDs on Multi-core Systems”, in: *8th USENIX Workshop on Hot Topics in Storage and File Systems (HotStorage 16)*, Denver, CO: USENIX Association, 2016 (cit. on pp. 18, 31, 69, 71, 72, 87).
- [8] Katevenis et al., “Weighted round-robin cell multiplexing in a general-purpose ATM switch chip”, in: *IEEE Journal on Selected Areas in Communications* 9.8 (1991), pp. 1265–1279 (cit. on p. 132).
- [9] B ALLEN, *Smartmontools Project*, <https://www.smartmontools.org/>, 2018 (cit. on p. 106).

- 
- [10] Ethem Alpaydin, *Introduction to machine learning*, In MIT press, 2014 (cit. on pp. 56, 102, 104, 154).
  - [11] Maryam Amiri and Leyli Mohammad-Khanli, “Survey on prediction models of applications for resources provisioning in cloud”, in: *Journal of Network and Computer Applications* 82 (2017), pp. 93–113 (cit. on pp. 12, 24, 72, 74, 108, 112, 113).
  - [12] Nadav Amit, Dan Tsafir, and Assaf Schuster, “VSwapper: A Memory Swapper for Virtualized Environments”, in: *SIGARCH Comput. Archit. News* 42.1 (Feb. 2014), pp. 349–366 (cit. on p. 47).
  - [13] David P Anderson and Gilles Fedak, “The computational and storage potential of volunteer computing”, in: *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, vol. 1, IEEE, 2006, pp. 73–80 (cit. on p. 65).
  - [14] David P Anderson et al., “SETI@ home: an experiment in public-resource computing”, in: *Communications of the ACM* 45.11 (2002), pp. 56–61 (cit. on pp. 66, 142).
  - [15] Julio CS Anjos et al., “MRA++: Scheduling and data placement on MapReduce for heterogeneous environments”, in: *Future Generation Computer Systems* 42 (2015), pp. 22–35 (cit. on pp. 78, 136).
  - [16] Julio Anjos et al., “Enabling strategies for big data analytics in hybrid infrastructures”, in: *Proceedings of the 16th International Conference on High Performance Computing & Simulation*, 2018, pp. 869–876 (cit. on p. 76).
  - [17] Sergei Arnautov et al., “SCONE: Secure Linux Containers with Intel SGX”, in: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, Nov. 2016, pp. 689–703 (cit. on pp. 80, 143, 168).
  - [18] *Amazon EC2 Spot Instances*, <https://goo.gl/nEN8Bu>, 2018 (cit. on pp. 43, 66).
  - [19] Jens Axboe, *Fio-flexible io tester*, <http://freecode.com/projects/fio>, 2014 (cit. on p. 89).

- 
- [20] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, "Learning long-term dependencies with gradient descent is difficult", in: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166 (cit. on pp. 61, 62).
- [21] Farid Benhammedi, Zahia Gessoum, Aicha Mokhtari, et al., "CPU load prediction using neuro-fuzzy and Bayesian inferences", in: *Neurocomputing* 74.10 (2011), pp. 1606–1616 (cit. on pp. 73, 75).
- [22] F Benson, "A note on the estimation of mean and standard deviation from quantiles", in: *Journal of the Royal Statistical Society. Series B (Methodological)* 11.1 (1949), pp. 91–100 (cit. on p. 110).
- [23] Christian Bienia et al., "The PARSEC Benchmark Suite: Characterization and Architectural Implications", in: *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, PACT '08, Toronto, Ontario, Canada: ACM, 2008, pp. 72–81 (cit. on p. 94).
- [24] Simona Boboila, "Analysis, Modeling and Design of Flash-based Solid-state Drives", PhD thesis, Boston, MA, USA: In Northeastern University, Dept. College of Computer and Information Science, 2012, ISBN: 978-1-267-83989-3 (cit. on p. 71).
- [25] Norman Bobroff, Andrzej Kochut, and Kirk Beaty, "Dynamic placement of virtual machines for managing sla violations", in: *Integrated Network Management, 2007. IM'07. 10th IFIP/IEEE International Symposium on*, IEEE, 2007, pp. 119–128 (cit. on p. 49).
- [26] Gianluca Bontempi, "Long term time series prediction with multi-input multi-output local learning", in: *Proc. 2nd ESTSP* (2008), pp. 145–154 (cit. on p. 112).
- [27] Jalil Boukhobza and Pierre Olivier, *Flash Memory Integration: Performance and Energy Issues*, Elsevier, 2017 (cit. on pp. 68, 69, 70).
- [28] Aaron Brazell, *WordPress Bible*, vol. 726, John Wiley and Sons, 2011 (cit. on pp. 94, 146).
- [29] Leo Breiman, "Random forests", in: *Machine learning* 45.1 (2001), pp. 5–32 (cit. on p. 60).



- 
- [30] Leo Breiman and Philip Spector, “Submodel Selection and Evaluation in Regression. The X-Random Case”, in: *International Statistical Review* 60.3 (1992), pp. 291–319 (cit. on pp. 58, 99).
- [31] Leo Breiman et al., *Classification and Regression Trees*, New York: Chapman & Hall, 1984, p. 358, ISBN: 0-412-04841-8 (cit. on pp. 57, 58, 104).
- [32] Jon Brodtkin, *Case Study: Parallel Internet: Inside the Worldwide LHC computing Grid, 2008* (cit. on pp. 11, 23).
- [33] John S Bucy et al., “The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)”, in: *In PDL (2008)*, Greg Ganger (2008), p. 26 (cit. on p. 71).
- [34] Brendan Burns et al., “Borg, omega, and kubernetes”, in: (2016) (cit. on p. 41).
- [35] *cAdvisor Online documentation*, Website, Accessed May, 27st, 2019, 2019, URL: <https://github.com/google/cadvisor> (cit. on p. 143).
- [36] Calif and Armonk, *Google and IBM Announce University Initiative to Address Internet-Scale Computing Challenges*, Website, Accessed sept, 16st, 2019, 2007, URL: <https://www-03.ibm.com/press/us/en/pressrelease/22414.wss> (cit. on pp. 11, 23, 38).
- [37] Maria Carla Calzarossa et al., “Workloads in the Clouds”, in: *Principles of Performance and Reliability Modeling and Evaluation*, Springer, 2016, pp. 525–550 (cit. on pp. 16, 30).
- [38] Rich Caruana and Alexandru Niculescu-Mizil, “An empirical comparison of supervised learning algorithms”, in: *Proceedings of the 23rd international conference on Machine learning*, ACM, 2006, pp. 161–168 (cit. on p. 113).
- [39] Marcus Carvalho et al., “Long-term SLOs for reclaimed cloud computing resources”, in: *Proceedings of the 5th ACM Symposium on Cloud Computing*, ACM, 2014, pp. 1–13 (cit. on pp. 13, 25, 66, 126, 127).
- [40] Marcus Carvalho et al., “Long-term SLOs for reclaimed cloud computing resources”, in: *ACM Symposium on Cloud Computing (SoCC)*, Seattle, WA, USA, 2014, 20:1–20:13 (cit. on p. 29).

- 
- [41] Marcus Carvalho et al., “Long-term slobs for reclaimed cloud computing resources”, in: *Proceedings of the ACM Symposium on Cloud Computing*, ACM, 2014, pp. 1–13 (cit. on pp. [12](#), [24](#), [66](#), [74](#), [75](#), [108](#)).
- [42] Davide Castelvechi, “Artificial intelligence called in to tackle LHC data deluge”, in: *Nature News* 528.7580 (2015), p. 18 (cit. on pp. [11](#), [23](#)).
- [43] Girish Chandrashekar and Ferat Sahin, “A survey on feature selection methods”, in: *Computers & Electrical Engineering* 40.1 (2014), pp. 16–28 (cit. on p. [148](#)).
- [44] Tianqi Chen and Carlos Guestrin, “XGBoost: A Scalable Tree Boosting System”, in: *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD ’16, San Francisco, California, USA: ACM, 2016, pp. 785–794 (cit. on p. [100](#)).
- [45] Navraj Chohan et al., “See Spot Run: using spot instances for mapreduce workflows”, in: *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, USENIX, 2010, pp. 7–14 (cit. on p. [76](#)).
- [46] Victor Costan and Srinivas Devadas, “Intel SGX Explained”, in: *IACR Cryptology ePrint Archive* 2016 (2016), p. 86 (cit. on p. [168](#)).
- [47] Jean-Emile DARTOIS et al., “Tracking Application Fingerprint in a Trustless Cloud Environment for Sabotage Detection”, in: *MASCOTS 2019 - 27th IEEE International Symposium on the Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Rennes, France: IEEE, Oct. 2019, pp. 74–82 (cit. on pp. [22](#), [35](#)).
- [48] Jean-Emile Dartois et al., “Cuckoo: a Mechanism for Exploiting Ephemeral and Heterogeneous Cloud Resource”, in: *IEEE International Conference on Cloud Computing*, IEEE, 2019 (cit. on pp. [21](#), [34](#), [144](#)).
- [49] Jean-Emile Dartois et al., “Investigating machine learning algorithms for modeling SSD I/O performance for container-based virtualization”, in: *IEEE Transactions on Cloud Computing* 14 (2019), pp. 1–14 (cit. on pp. [20](#), [33](#), [46](#), [141](#), [157](#)).

- 
- [50] Jean-Emile Dartois et al., “Using Quantile Regression for Reclaiming Unused Cloud Resources while achieving SLA”, in: *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, IEEE, 2018, pp. 89–98 (cit. on pp. 20, 24, 34, 142).
- [51] Jeffrey Dean and Sanjay Ghemawat, “MapReduce: simplified data processing on large clusters”, in: *Communications of the ACM* 51.1 (2008), pp. 107–113 (cit. on pp. 126, 128).
- [52] Christina Delimitrou and Christos Kozyrakis, “Quasar: resource-efficient and QoS-aware cluster management”, in: *ACM SIGPLAN Notices* 49.4 (2014), pp. 127–144 (cit. on p. 24).
- [53] Sheng Di, Derrick Kondo, and Walfredo Cirne, “Host load prediction in a Google compute cloud with a Bayesian model”, in: *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*, IEEE, 2012, pp. 1–11 (cit. on pp. 73, 75).
- [54] Peter A Dinda and David R O’Hallaron, “An evaluation of linear models for host load prediction”, in: *High Performance Distributed Computing, 1999*. IEEE, 1999, pp. 87–96 (cit. on pp. 73, 75).
- [55] Matthieu Dorier et al., “CALCioM: Mitigating I/O Interference in HPC Systems through Cross-Application Coordination”, in: *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, May 2014, pp. 155–164 (cit. on pp. 71, 72).
- [56] Harris Drucker, “Improving regressors using boosting techniques”, in: *ICML*, vol. 97, 1997, pp. 107–115 (cit. on p. 61).
- [57] Wenliang Du, Mummoorthy Murugesan, and Jing Jia, “Uncheatable grid computing”, in: *Algorithms and theory of computation handbook*, Chapman & Hall/CRC, 2010, pp. 30–30 (cit. on pp. 79, 80).
- [58] Truong Vinh Truong Duy, Yukinori Sato, and Yasushi Inoguchi, “Improving accuracy of host load predictions on computational grids by artificial neural networks”, in: *International Journal of Parallel, Emergent and Distributed Systems* 26.4 (2011), pp. 275–290 (cit. on p. 73).
- [59] Easen Ho Esther Spanjer, *Survey Update: Users Share Their 2017 Storage Performance Needs*, <https://goo.gl/y3XVDv>, 2017 (cit. on p. 100).

- 
- [60] Richard Evans and Jim Gao, “Deepmind AI reduces Google data centre cooling bill by 40%”, in: *DeepMind blog* 20 (2016) (cit. on pp. 11, 23).
- [61] Christoph Fehling et al., *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*, Springer, 2014 (cit. on p. 39).
- [62] Wes Felter et al., “An updated performance comparison of virtual machines and Linux containers”, in: *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Mar. 2015, pp. 171–172 (cit. on pp. 45, 100).
- [63] Matthias Feurer et al., “Efficient and robust automated machine learning”, in: *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970 (cit. on p. 57).
- [64] “FFmpeg”, in: *Available from: <http://ffmpeg.org>* (2012) (cit. on pp. 94, 146).
- [65] A Fielding and CA O’Muircheartaigh, “Binary segmentation in survey analysis with particular reference to AID”, in: *The Statistician* (1977), pp. 17–28 (cit. on p. 58).
- [66] Yoav Freund and Robert E Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, in: *J. Comput. Syst. Sci.* 55.1 (Aug. 1997) (cit. on p. 61).
- [67] Jerome H Friedman, “Greedy function approximation: a gradient boosting machine”, in: *Annals of statistics* (2001), pp. 1189–1232 (cit. on p. 61).
- [68] Jerome H Friedman et al., “Multivariate adaptive regression splines”, in: *The annals of statistics* 19.1 (1991), pp. 1–67 (cit. on p. 59).
- [69] Jerome Friedman, Trevor Hastie, and Robert Tibshirani, *The elements of statistical learning*, vol. 1, Springer series in statistics New York, 2001 (cit. on pp. 59, 113, 148).
- [70] Eran Gal and Sivan Toledo, “Algorithms and data structures for flash memories”, in: *IN CSUR, ACM* 37.2 (2005), pp. 138–163 (cit. on p. 71).
- [71] Simson Garfinkel, *Architects of the information society: 35 years of the Laboratory for Computer Science at MIT*, MIT press, 1999 (cit. on p. 38).

- 
- [72] Rahul Garg et al., “A SLA framework for QoS provisioning and dynamic capacity allocation”, in: *Quality of Service, 2002. Tenth IEEE International Workshop on*, IEEE, 2002, pp. 129–137 (cit. on pp. 14, 26, 117).
- [73] Daniel Gmach et al., “Workload analysis and demand prediction of enterprise data center applications”, in: *Workload Characterization, 2007. IISWC 2007. IEEE 10th International Symposium on*, IEEE, 2007, pp. 171–180 (cit. on pp. 73, 75).
- [74] SC Goh, “Design-adaptive nonparametric estimation of conditional quantile derivatives”, in: *Journal of Nonparametric Statistics* (2012) (cit. on p. 113).
- [75] Philippe Golle and Ilya Mironov, “Uncheatable distributed computations”, in: *Topics in Cryptology—CT-RSA 2001* (2001), pp. 425–440 (cit. on pp. 79, 80).
- [76] Gerrit De Vynck, *Google to Spend \$13 Billion on Data Centers, Offices Across U.S.* <https://www.bloomberg.com/news/articles/2019-02-13/google-to-spend-13-billion-on-data-centers-offices-across-u-s>, Accessed Sept, 20st, 2019, 2018 (cit. on pp. 12, 24).
- [77] *Preemptible Virtual Machines*, <https://goo.gl/zoqP1x>, 2018 (cit. on pp. 43, 116).
- [78] Albert Greenberg et al., “The cost of a cloud: research problems in data center networks”, in: *ACM SIGCOMM computer communication review* 39.1 (2008), pp. 68–73 (cit. on pp. 12, 24).
- [79] Laura M. Grupp, John D. Davis, and Steven Swanson, “The Harey Tortoise: Managing Heterogeneous Write Performance in SSDs”, in: *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, San Jose, CA: USENIX, 2013, pp. 79–90 (cit. on p. 87).
- [80] Berk Gulmezoglu, Thomas Eisenbarth, and Berk Sunar, “Cache-Based Application Detection in the Cloud Using Machine Learning”, in: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS ’17, Abu Dhabi, United Arab Emirates: ACM, 2017, pp. 288–300 (cit. on pp. 79, 80).

- 
- [81] Shai Halevi et al., “Proofs of ownership in remote storage systems”, in: *Proceedings of the 18th ACM conference on Computer and communications security*, Acm, 2011, pp. 491–500 (cit. on p. 156).
- [82] Pengfei Han et al., “Large-scale prediction of long disordered regions in proteins using random forests”, in: *BMC bioinformatics* 10.1 (2009), p. 1 (cit. on p. 60).
- [83] Mohamed Handaoui et al., “Salamander: a Holistic Scheduling of MapReduce Jobs on Ephemeral Cloud Resources”, in: *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, IEEE, May 2020 (cit. on pp. 140, 168).
- [84] Hongwei Hao, Cheng-Lin Liu, and Hiroshi Sako, “Comparison of genetic algorithm and sequential search methods for classifier subset selection”, in: *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.* Citeseer, 2003, pp. 765–769 (cit. on p. 148).
- [85] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics, May 2013 (cit. on pp. 57, 96, 102, 104, 154).
- [86] Trevor Hastie et al., “The entire regularization path for the support vector machine”, in: *Journal of Machine Learning Research* 5.Oct (2004), pp. 1391–1415 (cit. on p. 57).
- [87] Kelsey Hightower, Brendan Burns, and Joe Beda, *Kubernetes: Up and Running Dive into the Future of Infrastructure*, 1st, O’Reilly Media, Inc., 2017, ISBN: 978-1491935675 (cit. on pp. 51, 53, 92).
- [88] Sepp Hochreiter and Jürgen Schmidhuber, “Long short-term memory”, in: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 62).
- [89] Paul Horn, “Autonomic computing: IBM’s perspective on the state of information technology”, in: (2001) (cit. on p. 50).
- [90] <http://man7.org/linux/man-pages/man7/namespaces.7.html> (cit. on p. 45).
- [91] H. Howie Huang et al., “Performance modeling and analysis of flash-based storage devices”, in: *In MSST, IEEE*, 2011, pp. 1–11 (cit. on p. 71).

- 
- [92] IDC/Seagate, *Data Age 2025; The Evolution of Data to Life-Critical*, Website, Accessed May, 1st, 2019, 2017, URL: <https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf> (cit. on pp. 11, 23, 126).
- [93] Sadeka Islam et al., “Empirical prediction models for adaptive resource provisioning in the cloud”, in: *Future Generation Computer Systems* 28.1 (2012), pp. 155–162 (cit. on pp. 73, 75).
- [94] Kevin Jackson, *OpenStack cloud computing cookbook*, Packt Publishing Ltd, 2012 (cit. on pp. 13, 25, 41, 51).
- [95] Bart Jacob et al., “A practical guide to the IBM autonomic computing toolkit”, in: *IBM Redbooks* 4 (2004), p. 10 (cit. on p. 91).
- [96] Anil Jain and Douglas Zongker, “Feature selection: Evaluation, application, and small sample performance”, in: *IEEE transactions on pattern analysis and machine intelligence* 19.2 (1997), pp. 153–158 (cit. on p. 148).
- [97] Seyyed Ahmad Javadi et al., “Scavenger: A Black-Box Batch Workload Resource Manager for Improving Utilization in Cloud Environments”, in: *Proceedings of the ACM Symposium on Cloud Computing*, SoCC ’19, Santa Cruz, CA, USA: ACM, 2019, pp. 272–285 (cit. on p. 165).
- [98] Brendan Jennings and Rolf Stadler, “Resource Management in Clouds: Survey and Research Challenges”, in: *J. Netw. Syst. Manage.* 23.3 (July 2015), pp. 567–619, ISSN: 1064-7570 (cit. on pp. 48, 49).
- [99] William Stanley Jevons, “The coal question: Can Britain survive”, in: *First published in* (1865) (cit. on p. 165).
- [100] Hui Jin et al., “Adapt: Availability-aware mapreduce data placement for non-dedicated distributed computing”, in: *Proceedings of the 32nd IEEE International Conference on Distributed Computing Systems*, IEEE, 2012, pp. 516–525 (cit. on p. 75).
- [101] Myoungsoo Jung et al., “Hios: A host interface i/o scheduler for solid state disks”, in: *In SIGARCH*, ACM 42.3 (2014), pp. 289–300 (cit. on pp. 71, 72).



- 
- [102] Ritu Jyoti, *TCO Analysis Comparing Private and Public Cloud Solutions for Running Enterprise Workloads Using the 5Cs Framework*, 2017, URL: <https://www.nutanix.com/go/idc-tco-analysis-comparing-private-and-public-cloud-solutions-for-running-enterprise-workloads> (cit. on p. 39).
- [103] Murat Karakus et al., “OMTiR: Open Market for Trading Idle Cloud Resources”, in: *2014 IEEE 6th International Conference on Cloud Computing Technology and Science*, IEEE, 2014, pp. 719–722 (cit. on p. 42).
- [104] Ramakrishna Karedla, J Spencer Love, and Bradley G Wherry, “Caching strategies to improve disk system performance”, in: *Computer* 27.3 (1994), pp. 38–46 (cit. on p. 156).
- [105] Katarzyna Keahey et al., “Sky computing”, in: *IEEE Internet Computing* 13.5 (2009), pp. 43–51 (cit. on p. 40).
- [106] Jeffrey O Kephart and David M Chess, “The vision of autonomic computing”, in: *Computer* 1 (2003), pp. 41–50 (cit. on p. 50).
- [107] Mukhtaj Khan et al., “Hadoop performance modeling for job estimation and resource provisioning”, in: *IEEE Transactions on Parallel and Distributed Systems* 27 (2016), pp. 441–454 (cit. on p. 133).
- [108] Jaeho Kim, Donghee Lee, and Sam H. Noh, “Towards SLO Complying SSDs Through OPS Isolation”, in: *13th USENIX Conference on File and Storage Technologies (FAST 15)*, Santa Clara, CA: USENIX Association, 2015, pp. 183–189 (cit. on pp. 71, 72).
- [109] Youngjae Kim et al., “Flashsim: A simulator for nand flash-based solid-state drives”, in: *In SIMUL (2009)*, IEEE, IEEE, 2009, pp. 125–131 (cit. on p. 71).
- [110] Ana Klimovic, Heiner Litz, and Christos Kozyrakis, “ReFlex: Remote Flash & Local Flash”, in: *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS ’17*, Xi’an, China: ACM, 2017, pp. 345–359 (cit. on p. 106).



- 
- [111] Muhammad Anas Knefati, Abderrahim Oulidi, and Belkacem Abdous, "Local linear double and asymmetric kernel estimation of conditional quantiles", in: *Communications in Statistics-Theory and Methods* 45.12 (2016), pp. 3473–3488 (cit. on p. 113).
- [112] Roger Koenker and Gilbert Bassett Jr, "Regression quantiles", in: *Econometrica: journal of the Econometric Society* (1978), pp. 33–50 (cit. on pp. 110, 113).
- [113] Ron Kohavi, "A Study of Cross-validation and Bootstrap for Accuracy Estimation and Model Selection", in: *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'95*, Montreal, Quebec, Canada: Morgan Kaufmann Publishers Inc., 1995, pp. 1137–1143 (cit. on p. 58).
- [114] A. Kougkas et al., "Leveraging burst buffer coordination to prevent I/O interference", in: *2016 IEEE 12th International Conference on e-Science (e-Science)*, Oct. 2016, pp. 371–380 (cit. on p. 71).
- [115] Jitendra Kumar, Rimsha Goomer, and Ashutosh Kumar Singh, "Long Short Term Memory Recurrent Neural Network (LSTM-RNN) Based Workload Forecasting Model For Cloud Datacenters", in: *Procedia Computer Science* 125 (2018), pp. 676–682 (cit. on pp. 73, 75, 114).
- [116] Dan Kusnetzky, *Virtualization: A Manager's Guide*, "O'Reilly Media, Inc.", 2011 (cit. on p. 44).
- [117] *KVM Kernel Virtual Machine*, Website, Accessed May, 1st, 2019, 2019, URL: <https://www.linux-kvm.org> (cit. on p. 45).
- [118] Nikolay Laptev et al., "Time-series extreme event forecasting with neural networks at uber", in: *International Conference on Machine Learning*, vol. 34, 2017, pp. 1–5 (cit. on p. 114).
- [119] A. Lenk et al., "What's inside the Cloud? An architectural map of the Cloud landscape", in: *2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing*, May 2009, pp. 23–31 (cit. on p. 40).

- 
- [120] Yale Li, Yushi Shen, and Yudong Liu, “Cloud Computing Networks: Utilizing the Content Delivery Network”, in: *Enabling the New Era of Cloud Computing: Data Security, Transfer, and Management*, IGI Global, 2014, pp. 214–225 (cit. on p. 73).
- [121] Heshan Lin et al., “Moon: Mapreduce on opportunistic environments”, in: *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ACM, 2010, pp. 95–106 (cit. on pp. 76, 78).
- [122] Ying-Dar Lin et al., “Application classification using packet size distribution and port association”, in: *Journal of Network and Computer Applications* 32.5 (2009), Next Generation Content Networks, pp. 1023–1030 (cit. on p. 79).
- [123] Michel J Litzkow, Miron Livny, and Matt W Mutka, *Condor-a hunter of idle workstations*, tech. rep., University of Wisconsin-Madison Department of Computer Sciences, 1987 (cit. on p. 142).
- [124] Bill Loudon, “Increase Your 100’s Storage with 128K from Compuserve”, in: *Portable 100.1* (1983), p. 1 (cit. on pp. 11, 23).
- [125] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos, “Statistical and Machine Learning forecasting methods: Concerns and ways forward”, in: *PloS one* 13.3 (2018), e0194889 (cit. on p. 73).
- [126] Paul Marshall, Kate Keahey, and Tim Freeman, “Improving utilization of infrastructure clouds”, in: *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, 2011, pp. 205–214 (cit. on pp. 12, 65, 108, 127).
- [127] Michael Maurer, Ivona Brandic, and Rizos Sakellariou, “Self-adaptive and resource-efficient sla enactment for cloud computing infrastructures”, in: *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, IEEE, 2012, pp. 368–375 (cit. on p. 91).
- [128] Peter Mell, Tim Grance, et al., “The NIST definition of cloud computing”, in: (2011) (cit. on p. 38).
- [129] Paul Menage, *CGroup online documentation*, <https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt>, 2016 (cit. on p. 87).

- 
- [130] Xiaoqiao Meng et al., “Efficient resource provisioning in compute clouds via vm multiplexing”, in: *Proceedings of the 7th international conference on Autonomic computing*, ACM, 2010, pp. 11–20 (cit. on pp. 12, 24, 163).
- [131] Mohamed Merabet et al., “A Predictive Map Task Scheduler for Optimizing Data Locality in MapReduce Clusters”, in: *International Journal of Grid and High Performance Computing* 10.4 (2018), pp. 1–14 (cit. on p. 77).
- [132] Dirk Merkel, “Docker: lightweight linux containers for consistent development and deployment”, in: *Linux Journal* 2014.239 (2014), p. 2 (cit. on pp. 45, 55).
- [133] James N Morgan, Robert C Messenger, and A THAID, “A sequential analysis program for the analysis of nominal scale dependent variables”, in: *Institute for Social Research, University of Michigan* (1973) (cit. on p. 58).
- [134] AB MySQL, *MySQL*, <https://www.mysql.com>, 2001 (cit. on pp. 94, 146).
- [135] Vlad Nitu et al., “StopGap: Elastic VMs to enhance server consolidation”, in: *Software: Practice and Experience* 47.11 (2017), pp. 1501–1519 (cit. on p. 65).
- [136] Qais Noorshams et al., “Automated Modeling of I/O Performance and Interference Effects in Virtualized Storage Systems”, in: *2014 IEEE 34th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, June 2014, pp. 88–93 (cit. on pp. 71, 72, 97).
- [137] Hamza Ouarnoughi, “Placement autonome de machines virtuelles sur un système de stockage hybride dans un cloud IaaS”, PhD thesis, Université de Bretagne occidentale-Brest, Dept. Informatique, 2017 (cit. on pp. 50, 91).
- [138] Edouard Outin et al., “Enhancing cloud energy models for optimizing datacenters efficiency”, in: *Cloud and Autonomic Computing (ICCAC), 2015 International Conference on*, IEEE, 2015, pp. 93–100 (cit. on p. 91).
- [139] Fabian Pedregosa et al., “Scikit-learn: Machine learning in Python”, in: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830 (cit. on pp. 63, 100, 151).

- 
- [140] Xing Pu et al., “Understanding Performance Interference of I/O Workload in Virtualized Cloud Environments”, in: *2010 IEEE 3rd International Conference on Cloud Computing*, July 2010, pp. 51–58 (cit. on p. 87).
- [141] Will Reese, “Nginx: the high-performance web server and reverse proxy”, in: *Linux Journal* 2008.173 (2008), p. 2 (cit. on pp. 94, 146).
- [142] Kai Ren and Garth Gibson, “TABLEFS: Enhancing Metadata Efficiency in the Local File System”, in: *Presented as part of the 2013 USENIX Annual Technical Conference (USENIX ATC 13)*, San Jose, CA: USENIX, 2013, pp. 145–156 (cit. on p. 106).
- [143] *rkt: A security-minded, standards-based container engine*, <https://coreos.com/rkt/>, 2019 (cit. on p. 55).
- [144] Drew Robb, *Data Center Strategy: Tips for Better Capacity Planning*, Website, Accessed May, 20st, 2019, 2017, URL: <https://www.datacenterknowledge.com/archives/2017/05/24/data-center-strategy-tips-for-better-capacity-planning> (cit. on pp. 13, 25).
- [145] Chris Ruemmler and John Wilkes, “An Introduction to Disk Drive Modeling”, in: *Computer* 27.3 (Mar. 1994), pp. 17–28 (cit. on p. 71).
- [146] Eric Sammer, *Hadoop Operations: A Guide for Developers and Administrators*, "O'Reilly Media, Inc.", 2012 (cit. on p. 135).
- [147] Ignacio Sañudo et al., “A Survey on Shared Disk I/O Management in Virtualized Environments Under Real Time Constraints”, in: *SIGBED Rev.* 15.1 (Mar. 2018), pp. 57–63 (cit. on p. 71).
- [148] Luis FG Sarmenta, “Sabotage-tolerance mechanisms for volunteer computing systems”, in: *Future Generation Computer Systems* 18.4 (2002), pp. 561–572 (cit. on pp. 79, 80, 143, 150).
- [149] Roei Schuster, Vitaly Shmatikov, and Eran Tromer, “Beauty and the Burst: Remote Identification of Encrypted Video Streams”, in: *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC: USENIX Association, 2017, pp. 1357–1374, ISBN: 978-1-931971-40-9 (cit. on p. 80).
- [150] Malte Schwarzkopf et al., “Omega: flexible, scalable schedulers for large compute clusters”, in: *SIGOPS European Conference on Computer Systems (EuroSys)*, Prague, Czech Republic, 2013, pp. 351–364 (cit. on p. 53).

- 
- [151] Muhamad Shaari et al., “DYNAMIC PRICING SCHEME FOR RESOURCE ALLOCATION IN MULTI-CLOUD ENVIRONMENT.”, in: *Malaysian Journal of Computer Science* 30.1 (2017) (cit. on pp. 74, 112, 116).
- [152] Prateek Sharma et al., “Containers and Virtual Machines at Scale: A Comparative Study”, in: *Proceedings of the 17th International Middleware Conference*, Middleware ’16, Trento, Italy: ACM, 2016, 1:1–1:13, ISBN: 978-1-4503-4300-8 (cit. on pp. 46, 55).
- [153] Elizabeth Shriver, Arif Merchant, and John Wilkes, “An Analytic Behavior Model for Disk Drives with Readahead Caches and Request Reordering”, in: *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS ’98/PERFORMANCE ’98, Madison, Wisconsin, USA: In SIGMETRICS, ACM, 1998, pp. 182–191 (cit. on p. 71).
- [154] Konstantin Shvachko et al., “The hadoop distributed file system”, in: *Proceedings of the 26th IEEE Symposium on Mass Storage Systems and Technologies*, IEEE, 2010, pp. 1–10 (cit. on pp. 11, 24, 126).
- [155] Stelios Sidiroglou-Douskos et al., “Managing performance vs. accuracy trade-offs with loop perforation”, in: *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ACM, 2011, pp. 124–134 (cit. on pp. 19, 32, 142).
- [156] R. Singhal and A. Verma, “Predicting Job Completion Time in Heterogeneous MapReduce Environments”, in: *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, May 2016, pp. 17–27 (cit. on p. 77).
- [157] Stephen Soltesz et al., “Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors”, in: *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys ’07, Lisbon, Portugal: ACM, 2007, pp. 275–287 (cit. on p. 45).
- [158] Binbin Song et al., “Host load prediction with long short-term memory in cloud computing”, in: *The Journal of Supercomputing* (2017), pp. 1–15 (cit. on pp. 73, 75, 114).

- 
- [159] Gokul Soundararajan and Cristiana Amza, “Towards End-to-End Quality of Service: Controlling I/O Interference in Shared Storage Servers”, in: *Middleware 2008: ACM/IFIP/USENIX 9th International Middleware Conference Leuven, Belgium, December 1-5, 2008 Proceedings*, ed. by Valérie Issarny and Richard Schantz, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 287–305 (cit. on pp. 71, 88).
- [160] Apache Spark, *Apache Spark™-Lightning-Fast Cluster Computing*, 2014 (cit. on p. 118).
- [161] Richard M Stallman et al., “Using the GNU compiler collection”, in: *Free Software Foundation 4.02* (2003) (cit. on pp. 94, 146).
- [162] Luiz Angelo Steffene et al., “Mapreduce challenges on pervasive grids”, in: *Journal of Computer Science* 10.11 (2014), pp. 2194–2210 (cit. on p. 76).
- [163] Souhaib Ben Taieb and Amir F Atiya, “A bias and variance analysis for multistep-ahead time series forecasting”, in: *IEEE transactions on neural networks and learning systems* 27.1 (2016), pp. 62–76 (cit. on p. 112).
- [164] Bing Tang et al., “Availability/network-aware mapreduce over the internet”, in: *Information Sciences* 379 (2017), pp. 94–111 (cit. on p. 77).
- [165] Vasily Tarasov, Erez Zadok, and Spencer Shepler, “Filebench: A flexible framework for file system benchmarking”, in: *The USENIX Magazine* 41.1 (2016) (cit. on pp. 94, 146).
- [166] Alain Tchana et al., “Mitigating Performance Unpredictability in the IaaS Using the Kyoto Principle”, in: *Proceedings of the 17th International Middleware Conference, Middleware ’16, Trento, Italy: Association for Computing Machinery*, 2016, ISBN: 9781450343008 (cit. on pp. 106, 165).
- [167] Aline Tenu, “Les débuts de la comptabilité en Mésopotamie. Archéologie de la comptabilité. Culture matérielle des pratiques comptables au Proche-Orient ancien”, in: *Comptabilités. Revue d’histoire des comptabilités* 8 (2016) (cit. on pp. 11, 23).

- 
- [168] Jonathan Thatcher et al., *Solid State Storage (SSS) Performance Test Specification (PTS) Enterprise Version 1.1*, [http://www.snia.org/sites/default/files/SSS\\\_PTS\\\_Enterprise\\\_v1.1.pdf](http://www.snia.org/sites/default/files/SSS\_PTS\_Enterprise\_v1.1.pdf), 2013 (cit. on pp. 89, 97).
- [169] Avishay Traeger et al., “A Nine Year Study of File System and Storage Benchmarking”, in: *Trans. Storage* 4.2 (May 2008), 5:1–5:56 (cit. on pp. 71, 93).
- [170] Jack V Tu, “Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes”, in: *Journal of clinical epidemiology* 49.11 (1996), pp. 1225–1231 (cit. on p. 57).
- [171] *VMware ESX*, <https://www.vmware.com/fr/products/vsphere-hypervisor.html>, Accessed Oct, 14st, 2019, 2019 (cit. on pp. 13, 25, 45).
- [172] *VMware vCenter*, <https://www.vmware.com/products/vcenter-server>, Accessed Sept, 20st, 2019, 2019 (cit. on p. 41).
- [173] Abhishek Verma et al., “Large-scale cluster management at Google with Borg”, in: *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015 (cit. on p. 53).
- [174] Mengzhi Wang et al., “Storage device performance prediction with CART models”, in: *Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004. (MASCOTS 2004). Proceedings. The IEEE Computer Society’s 12th Annual International Symposium on*, Oct. 2004, pp. 588–595 (cit. on p. 71).
- [175] Johannes Winter, “Trusted computing building blocks for embedded linux-based ARM trustzone platforms”, in: *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, 2008, pp. 21–30 (cit. on p. 168).
- [176] Yongwei Wu et al., “Load prediction using hybrid model for computational grid”, in: *Grid Computing, 2007 8th IEEE/ACM International Conference on*, IEEE, 2007, pp. 235–242 (cit. on pp. 73, 75).
- [177] Miguel G. Xavier et al., “A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds”, in: *Proceedings of the 2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-*



- 
- Based Processing*, PDP '15, Washington, DC, USA: IEEE Computer Society, 2015, pp. 253–260 (cit. on pp. [87](#), [88](#), [97](#)).
- [178] Miguel G Xavier et al., “Performance evaluation of container-based virtualization for high performance computing environments”, in: *2013 21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Feb. 2013, pp. 233–240 (cit. on p. [45](#)).
- [179] Ying Yan et al., “TR-Spark: Transient Computing for Big Data Analytics”, in: *Proceedings of the Seventh ACM Symposium on Cloud Computing*, SoCC '16, Santa Clara, CA, USA: ACM, 2016, pp. 484–496 (cit. on pp. [65](#), [78](#)).
- [180] Jihoon Yang and Vasant Honavar, “Feature subset selection using a genetic algorithm”, in: *Feature extraction, construction and selection*, Springer, 1998, pp. 117–136 (cit. on p. [148](#)).
- [181] Lingyun Yang, Ian Foster, and Jennifer M Schopf, “Homeostatic and tendency-based CPU load predictions”, in: *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, IEEE, 2003, 9–pp (cit. on p. [73](#)).
- [182] Qiangpeng Yang et al., “A new method based on PSR and EA-GMDH for host load prediction in cloud computing system”, in: *The Journal of Supercomputing* 68.3 (2014), pp. 1402–1417 (cit. on p. [73](#)).
- [183] Qiangpeng Yang et al., “Multi-step-ahead host load prediction using autoencoder and echo state networks in cloud computing”, in: *The Journal of Supercomputing* 71.8 (2015), pp. 3037–3053 (cit. on p. [73](#)).
- [184] Youngseok Yang et al., “Pado: A Data Processing Engine for Harnessing Transient Resources in Datacenters”, in: *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, Belgrade, Serbia: ACM, 2017, pp. 575–588, ISBN: 978-1-4503-4938-3 (cit. on pp. [65](#), [76](#), [78](#)).
- [185] Ziye Yang et al., “Understanding the effects of hypervisor I/O scheduling for virtual machine performance interference”, in: *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, Dec. 2012, pp. 34–41 (cit. on p. [87](#)).



- 
- [186] Yier Jin and Y. Makris, “Hardware Trojan detection using path delay fingerprint”, in: *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, June 2008, pp. 51–57 (cit. on p. 79).
- [187] Li Yin, Sandeep Uttamchandani, and Randy Katz, “An empirical exploration of black-box performance models for storage systems”, in: *14th IEEE International Symposium on Modeling, Analysis, and Simulation*, Sept. 2006, pp. 433–440 (cit. on p. 71).
- [188] Matei Zaharia et al., “Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling”, in: *Proceedings of the 5th European Conference on Computer Systems*, ACM, 2010, pp. 265–278 (cit. on p. 75).
- [189] Matei Zaharia et al., “Improving MapReduce performance in heterogeneous environments.”, in: *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, USENIX, 2008, pp. 29–42 (cit. on pp. 75, 78).
- [190] S. Zander, T. Nguyen, and G. Armitage, “Automated traffic classification and application identification using machine learning”, in: *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN’05)*, Nov. 2005, pp. 250–257 (cit. on pp. 79, 80).
- [191] Bo Zhang et al., “CloudGC: Recycling Idle Virtual Machines in the Cloud”, in: *5th IEEE International Conference on Cloud Engineering (IC2E)*, ed. by Indranil Gupta and Jiangchuan Liu, Proceedings of the 5th IEEE International Conference on Cloud Engineering (IC2E), Vancouver, Canada: IEEE, Apr. 2017, p. 10 (cit. on p. 65).
- [192] Yunqi Zhang et al., “History-based harvesting of spare cycles and storage in large-scale datacenters”, in: *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, EPFL-CONF-224446, 2016, pp. 755–770 (cit. on p. 78).
- [193] Shanyu Zhao, Virginia Lo, and C Gauthier Dickey, “Result verification and trust-based scheduling in peer-to-peer grids”, in: *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, IEEE, 2005, pp. 31–38 (cit. on pp. 79, 80).

- 
- [194] Ji Zhu et al., “Multi-class adaboost”, in: *Statistics and its Interface* 2.3 (2009), pp. 349–360 (cit. on p. [61](#)).



---

**Titre :** Exploitation des ressources hétérogènes inutilisées du Cloud pour des applications avec garanties de qualité de service

**Mot clés :** Cloud, ressources inutilisées, placement intelligent, données volumineuses, apprentissage machine, interférences.

**Résumé :** La gestion efficace des ressources est une dimension importante dans le domaine du Cloud computing tant pour des raisons économiques qu'écologiques. Il a été observé que les ressources des infrastructures Cloud ne sont utilisées qu'à hauteur de 20% en moyenne. Pour améliorer son modèle économique, un fournisseur de Cloud doit chercher à optimiser l'utilisation de l'ensemble de ses ressources matérielles sans jamais enfreindre la qualité de service minimale qu'il a contractualisé avec ses clients.

L'objectif de cette thèse est d'exploiter les ressources hétérogènes inutilisées du Cloud pour des applications avec garanties de qualité de service. Pour cela, la thèse présente quatre contributions. La première se concentre

sur l'estimation de la capacité réelle d'une machine virtualisée utilisant des périphériques de stockage SSD en tenant compte des performances variables causées par des interférences. La deuxième vise à estimer les ressources futures inutilisées d'une infrastructure Cloud et anticiper les risques d'impact sur la qualité de service. Puis, une troisième contribution démontre la possibilité d'exploiter efficacement les ressources inutilisées du Cloud pour des applications de données volumineuses sans perturber les applications des fournisseurs de ressources. Enfin, une dernière contribution propose de vérifier la bonne exécution d'une application dans un environnement sans confiance.

---

**Title:** Leveraging Cloud unused heterogeneous resources for applications with SLA guarantees

**Keywords:** Cloud, unused resources, smart placement, big data, machine learning, interference.

**Abstract:** Managing efficiently Cloud resources and reducing costs are major concerns for Cloud providers both economic and ecological reasons. However, It has been observed that the average usage of resources remains low, between 25-35% for the CPU. One way to improve Cloud data center resource utilization and thus reduce the total cost of ownership is to reclaim Cloud unused resources. However, reselling resources needs to meet the expectations of its customers in terms of quality of service.

In this thesis the goal is to leverage Cloud

unused resources for applications with SLA guarantees. To achieve that, this thesis proposes four contributions. The first one focuses on estimating real system capacity by considering SSD interferences. The second aims at estimating future use to provide availability guarantees. Then, a third contribution demonstrates the possibility of leveraging Cloud unused resources for big data without interfering with the co-located workloads. Finally, the last contribution aims at preventing malicious infrastructure owners from sabotaging the computation.