CONCEPTION D'OUTILS DE GÉNÉRATION DE CHARGE DE TRAFIC DANS DES ENVIRONNEMENTS DE COMMUNICATION VIRTUALISÉS

par

Cédric ST-ONGE

MÉMOIRE PAR ARTICLES PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE COMME EXIGENCE PARTIELLE À L'OBTENTION DE LA MAÎTRISE EN SCIENCES APPLIQUÉES
CONCENTRATION GÉNIE DES TECHNOLOGIES DE L'INFORMATION
M. Sc. A.

MONTRÉAL, LE 8 JUIN 2018

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Mme Nadjia Kara, directrice de mémoire
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Ségla Kpodjedo, président du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

M. Abdelouahed Gherbi, membre du jury
Département de génie logiciel et des TI à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 14 MAI 2018

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Mes collègues du laboratoire LASI, pour leur camaraderie, leur esprit d'équipe et leurs nombreux encouragements. Ces 20 mois passés à vos côtés auront passé trop vite.

Le personnel, les professeurs et chargés de cours du département de génie logiciel et des TI de l'ÉTS, qui forment une équipe formidable et qui m'ont transmis leur fierté de faire partie de cette grande institution.

Messieurs Samuel Docquier et Patrice Dion, qui ont ôté toutes les embûches logistiques et technologiques durant mon cheminement académique, me permettant de me concentrer pleinement sur mon projet.

Mes parents, mon frère et ma sœur, pour m'avoir transmis les valeurs me guidant quotidiennement et qui ont fait de moi ce que je suis aujourd'hui.

Ma nièce Olivia, pour qui la communauté scientifique et moi, à ma façon, contribuons activement pour qu'elle vive dans un monde meilleur.

Mes beaux-parents, pour m'avoir convaincu qu'il n'est jamais trop tard.

Ma conjointe Émilie, qui a vu en moi l'étincelle et qui m'a permis d'aller jusqu'au bout de mes rêves.

Mme Nadjia Kara, pour son soutien indéfectible, sa confiance et sa rigueur, qui m'aura permis d'exploiter mon plein potentiel.

CONCEPTION D'OUTILS DE GÉNÉRATION DE CHARGE DE TRAFIC ET DE GESTION DE RESSOURCES DANS DES ENVIRONNEMENTS DE COMMUNICATION VIRTUALISÉS

Cédric ST-ONGE

RÉSUMÉ

Les systèmes infonuagiques sont connus pour leur extensibilité, permettant l'ajout ou le retrait de ressources basé sur un modèle de service dit « sur-demande ». Cependant, la nature instable de la charge de trafic (« *workload* ») et la variété d'applications s'exécutant dans le Nuage entraîne certains problèmes de sur-approvisionnement et de sous-approvisionnement des ressources, causant un gaspillage de ressources et ayant souvent des répercussions négatives au niveau de l'expérience utilisateur. Une méthode efficace permettant de pallier ces problèmes réside en la modélisation de charge de trafic, qui est utilisée pour la conception d'approches proactives de prise de décision, permettant des stratégies d'approvisionnement de ressources ainsi que l'anticipation de potentiels problèmes de performance.

Les modèles de charges de trafic sont habituellement conçus à partir du comportement d'usagers et d'applications d'un système, limitant ainsi leur utilisation à des domaines bien spécifiques. Indubitablement, cette pratique crée un dilemme dans le domaine des systèmes infonuagiques, dû au fait qu'une panoplie d'applications hétérogènes y sont exécutées et que plusieurs utilisateurs ont accès à leurs ressources. Les modèles de charges de trafic pour de telles infrastructures requièrent une adaptation à l'évolution des paramètres de configuration du système, tels que la fluctuation d'exécution de tâches, le dimensionnement horizontal et vertical, ainsi que la migration de ressources virtualisées. De plus, les jeux de données de charges de trafic collectées recèlent souvent de précieuses informations relatives à des comportements (« *patterns* ») récurrents dans le système évalué. La classification de ces comportements périodiques par leur amplitude, leur forme et leur longueur se veut un atout majeur pour tout chercheur visant à améliorer ses modèles de charge de trafic. L'objectif de notre projet est de générer des modèles de charges de trafic (1) génériques, indépendants du comportement des usagers et des applications opérant dans un système infonuagique, (2) pouvant convenir à n'importe quel type ou domaine de charge de trafic, (3) permettant de modéliser des variations brusques qui sont susceptibles de se produire dans des environnements infonuagiques, et (4) avec une grande fidélité relative aux données observées, dans un temps d'exécution relativement bas. Un objectif sous-jacent est également visé sous la forme d'un mécanisme de détection de périodicité de charge de trafic, permettant la détection de comportements périodiques des charges de trafic.

Mots-clés: Infonuagique, Nuage, Modélisation de charge de trafic, Estimation de charge de trafic, Modèle Hull-White, Algorithmes génétiques, Séries temporelles, Transréversion par préfixe.

DESIGN OF WORKLOAD GENERATION AND RESOURCE MANAGEMENT TOOLS FOR VIRTUALIZED COMMUNICATION ENVIRONMENTS

Cédric ST-ONGE

ABSTRACT

Cloud computing systems are known for their elastic property which allows the dynamic addition and removal of resources based on the on-demand service model. However, the unsteady workloads and the variety of the applications running in the cloud entail the problems of resources over-provisioning and under-provisioning, which cause resources wastage and user experience degradation. In order to address this problem, workload modeling can be used to design proactive decision-making approaches to optimize the resource provisioning strategies and anticipate any performance problem.

Workload models are typically built based on user and application behavior in a system, limiting them to specific domains. Undoubtedly, such practice forms a dilemma in a Cloud environment where a wide range of heterogeneous applications are running, and many users have access to these resources. The workload model in such infrastructure must adapt to the evolution in the system configuration parameters like job load fluctuation, horizontal scaling, vertical scaling, or migration of virtualized resources. Moreover, the collected workload data often hold precious information about recurring patterns in the evaluated system's resource behavior. Classification of such periodic patterns by amplitude, length and shape can be an invaluable asset to researchers aiming to improve workload models.

The aim of this work is (1) to generate generic workload models which are independent of user behavior and the applications running in the system, (2) that are able to fit any workload domain and type, (3) that are able to model sharp workload variations that are most likely to appear in cloud environments, and (4) with high degree of fidelity with respect to the observed data in a short period of execution time. As a subset, a workload periodicity detection mechanism is also proposed, enabling detection of cyclic workload patterns and workload behavior anomalies. To achieve this aim, this work is therefore divided into two complementary working areas. The first working area centers around two approaches for workload estimation, while the second working area centers around workload periodicity detection.

Keywords: Cloud computing, Workload modeling, Workload estimation, Hull-White model, Genetics, Time series, Prefix transreversal.

TABLE DES MATIÈRES

Page

INTRODUCTION 1

CHAPITRE 1 REVUE DE LITTÉRATURE 7

CHAPITRE 2 ADVANCED WORKLOAD MODELING FOR CLOUD SYSTEMS 11
2.1 Abstract 11
2.2 Introduction 12
2.3 Background 14
2.4 Related Work 15
2.4.1 Workload Domains 15
2.4.2 Workload Types 16
2.4.3 Workload Attributes 17
2.4.4 Static and Dynamic Workloads 18
2.4.5 Workload Modeling 19
2.4.6 Proposed Approach Motivation 21
2.5 Problem Illustration 21
2.6 Workload Generator Architecture 22
2.7 Hull-White Workload Modeling 24
2.7.1 Stochastic Differential Equations (SDE) 24
2.7.2 Splines 24
2.7.3 Hull-White Process 25
2.7.4 Estimation of $\boldsymbol{\theta}$ 26
2.8 GA Workload Estimation 26
2.8.1 Individuals 26
2.8.2 Segments 27
2.8.3 Fitness Function 27
2.8.4 Selection 29
2.8.5 Crossover Operator 29
2.8.6 Mutation Operator 30
2.8.7 Algorithm 30
2.9 Kalman-SVR Workload Estimation 31
2.10 Experimental Results 34
2.10.1 Use Cases 34
2.10.2 Configuration 34
2.10.3 Results 36
2.11 Conclusion and Future Work 43
2.12 Acknowledgement 44

CHAPITRE 3 WORKLOAD PERIODICITY DETECTION FOR CLOUD SYSTEMS 45
3.1 Abstract 45
3.2 Introduction 46

3.2.1 Contributions ... 47
3.3 Background ... 48
3.4 Related Work ... 49
3.4.1 Periodicity Detection Algorithms ... 49
3.4.2 Workload Pattern Recognition based on Prefix Transreversal ... 52
3.4.3 Discussions and Comparison with the State-of-the-art ... 53
3.5 Problem Illustration ... 54
3.6 Architecture ... 54
3.6.1 Preparation Phase ... 55
3.6.2 Decision Phase ... 55
3.7 Workload Data Estimation ... 56
3.7.1 Splines ... 56
3.7.2 Prefix Transreversals ... 57
3.7.2.1 Prefix Transreversals on Binary Strings ... 57
3.7.2.2 Estimation of *x* and *y* ... 57
3.7.2.3 Digital Print ... 58
3.7.2.4 L-transreversal ... 59
3.7.2.5 Grouping Distance ... 59
3.7.3 Preparation Phase Algorithm ... 60
3.8 Workload Periodicity Detection ... 61
3.8.1 Computing Digital Print Deviations ... 62
3.8.2 Evaluation Patterns and Sliding Windows ... 63
3.8.3 Algorithm of the Evaluation Phase ... 64
3.9 Experimental Results ... 66
3.9.1 Use cases ... 66
3.9.2 Configuration ... 67
3.9.3 Results ... 69
3.9.3.1 Automated Parameter Selection Process ... 69
3.9.3.2 Workload Periodicity Detection vs. Autocorrelation ... 71
3.9.3.3 Length, Shape and Amplitude ... 73
3.10 Conclusion and Future Work ... 75
3.11 Acknowledgment ... 76

CHAPITRE 4 DISCUSSION DES RÉSULTATS ... 77

CONCLUSION ... 79

BIBLIOGRAPHIE ... 81

LISTE DES TABLEAUX

Page

Table 2.1 Average MAPE of solutions based on 10 simulations....43

Table 2.2 Average execution time (s) based on 10 minutes simulations, 60 generations for GA....43

Table 3.1 IMS1, Cycle 1 Load Profile....67

Table 3.2 IMS1, Load Profile Variations, Cycles 2-6....68

Table 3.3 IMS2, Cycle 1 Load Profile....68

Table 3.4 IMS2, Load Profile Variations, Cycles 2-6....69

Table 3.5 Automated Parameter Selection....70

Table 3.6 Periodicity Detection Comparison....73

Table 3.7 IMS1 - Cycle length, starting point and ending point....73

Table 3.8 IMS2 - Cycle length, starting point and ending point....74

Table 3.9 Web App - Cycle length, starting point and ending point....74

Table 3.10 Google - Cycle length, starting point and ending point....75

LISTE DES FIGURES

Page

Figure 2.1 Proposed Workload Generator Scheme 23

Figure 2.2 Individuals and Segments 27

Figure 2.3 Selection Process 28

Figure 2.4 Crossover Operation 29

Figure 2.5 Mutation Operator 30

Figure 2.6 IMS Hull-White Models: Observed Data, μ(t) and σ(t) 36

Figure 2.7 IMS: Observed CPU Workloads 37

Figure 2.8 IMS: Estimated CPU Workload-GA 37

Figure 2.9 Google CPU Workload Model: Hull-White-GA and Observed Data 38

Figure 2.10 Web Application CPU Workload Model: Hull-White-GA and Observed Data 38

Figure 2.11 IMS1: SVR and Kalman SVR 40

Figure 2.12 IMS2: SVR and Kalman SVR 41

Figure 2.13 IMS3: SVR and Kalman SVR 41

Figure 2.14 Google: SVR and Kalman SVR 42

Figure 2.15 Web Application: SVR and Kalman SVR 42

Figure 3.1 Digital Print Deviations from Web App Dataset 62

Figure 3.2 IMS1: Observed Data, Periodicity Detection and Autocorrelation Function 71

Figure 3.3 IMS2: Observed Data, Periodicity Detection and Autocorrelation Function 72

Figure 3.4 Web App: Observed Data, Periodicity Detection and Autocorrelation Function 72

Page

Figure 3.5 Google: Observed Data, Periodicity Detection and Autocorrelation Function 72

LISTE DES ALGORITHMES

Page

Algorithm 2.1 LoadModelingGA 31
Algorithm 2.2 EM-KalmanFilter-SVR 33
Algorithm 3.1 PreparationPhase 61
Algorithm 3.2 EvalPhase 64

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

AR	Autoregression
ARMA	Autoregression Moving Average
ATM	Asynchronous Transfer Mode
CAPEX	Capital Expenditures
CPS	Calls per Second
CPU	Central Processing Unit
CSCF	Call/Session Control Function
EA	Evolutionary Algorithm
EC2	Elastic Cloud Computing
EM	Expectation Maximisation
E/S	Entrées/sorties
ÉTS	École de technologie supérieure
GA	Genetic Algorithm
HW	Hull-White
IEEE	Institute of Electrical and Electronics Engineers
IMS	IP Multimedia Subsystem
I/O	Input/Output
IP	Internet Protocol
IT	Information Technology
MA	Moving Average
MAPE	Mean Absolute Percentage Error

MLE	Maximum Likelihood Estimation
MPLS	Multiprotocol Label Switching
NSERC	Natural Science and Engineering Research Council of Canada
Nuage	Nuage informatique
OPEX	Operational Expenditures
QoS	Quality of Service
RAM	Random Access Memory
RBF	Radial Basis Function
SDE	Stochastic Differential Equation
SLA	Service Level Agreement
STD	Standard Deviation
SVM	Support Vector Machine
SVR	Support Vector Regression
Telecom	Telecommunication
TI	Technologies de l'information
TIC	Technologies de l'information et des communications
USPTO	United States Patent and Trademark Office
VM	Virtual Machine
VMM	Virtual Machine Monitor
VoIP	Voice over Internet Protocol

LISTE DES SYMBOLES ET UNITÉS DE MESURE

CPS	Calls per Second
CPU %	CPU usage rate
GB	Gigabyte
GHz	Gigahertz
Lag	Lag in a cross-correlation
MAPE	Mean Absolute Percentage Error
s	Seconde

INTRODUCTION

0.1 Contexte

L'émergence, au milieu de la dernière décennie, des systèmes de virtualisation et, plus récemment, des environnements infonuagiques, a provoqué de profonds bouleversements dans l'ensemble du secteur des TIC (technologies de l'information et des télécommunications). Alors qu'auparavant, les entreprises déployaient localement une multitude de serveurs dédiés à une seule tâche homogène (ex : un serveur de base de données, un serveur d'impression, un serveur de courriels, un serveur web, CSCF (« *Call/Session Control Function* »), etc.), ces nouveaux paradigmes changèrent définitivement les méthodes de déploiement ainsi que l'architecture des services et des infrastructures informatiques; les applications pouvaient désormais être concentrées dans un environnement virtualisé, hétérogène et délocalisé, permettant de maximiser l'utilisation des ressources, de réduire les coûts de capitalisation (« *CAPEX* ») et les coûts d'opération (« *OPEX* ») des entreprises.

Il y eut d'abord de grandes avancées au niveau de la performance des processeurs et de la capacité de stockage, notamment, qui eurent pour effet de réduire considérablement la consommation globale des ressources d'un service par système (ex : CPU (« *Central Processing Unit* »), RAM (« *Random Access Memory* »), E/S (entrées/sorties) de disques, trafic réseau, etc.). Ces avancées offrirent donc la possibilité de concentrer plusieurs services sur un seul système à l'aide de la virtualisation, qui permit à son tour de pallier le problème de sous-utilisation des ressources. Par la suite, l'émergence du Nuage (« nuage informatique ») ajouta une flexibilité supplémentaire aux entreprises en leur offrant la possibilité de déployer une énorme quantité de services sur une grappe de systèmes délocalisés, offrant ainsi une extensibilité sur demande.

Cette évolution a permis de réduire considérablement les coûts des entreprises, certes, mais certains problèmes persistent et d'autres ont émergé dans son sillage. Parmi ceux-ci, les

enjeux de QoS (« *Quality of Service* ») et de SLA (« *Service Level Agreement* ») sont les plus critiques. Encore aujourd'hui, bien des entreprises ont recourt au sur-approvisionnement (« *over-provisionning* ») pour réduire tout risque de bris de SLA et pour conserver un niveau de QoS maximal. Cela est bien évidemment contraire à la philosophie qui a donné naissance au modèle infonuagique, puisque nous retournons à un état de sous-utilisation des ressources, qui a une incidence sur les coûts d'exploitation (CAPEX, OPEX). De plus, la sous-utilisation des ressources virtuelles limite le potentiel et l'efficacité de fonctions telles que le dimensionnement horizontal et vertical ainsi que la migration de machines virtuelles, puisque ces dernières sont échelonnées dans des situations peu optimales en termes de coûts et de risques sur la disponibilité d'un service.

Pour s'attaquer à cette problématique, un domaine de recherche se consacre à l'observation, l'estimation, la prédiction et la modélisation des charges de trafic dans des environnements virtualisés (Feitelson, 2014). Ce domaine vise à étudier le comportement des charges de trafic en réaction à l'utilisation d'un système virtualisé afin d'y déceler des tendances susceptibles d'en améliorer les politiques et configurations. Ceci permet à des administrateurs système, par exemple, de connaître le moment idéal pour déployer une nouvelle machine virtuelle pour réduire la charge de travail sur un CPU, et même, d'en automatiser le déploiement.

C'est dans cette optique que nous nous consacrons, dans ce projet, à la modélisation de charge de trafic dans des environnements de communication virtualisés.

0.2 Problématique

Bien que de nombreuses équipes de recherche se soient penchées sur la question de la modélisation de charge de trafic dans des environnements virtualisés au cours des dernières années (Bahga & Madisetti, 2011), (Yang, Luan, Li & Qian, 2012), (Moreno, Garraghan, Townend & Xu, 2013), nos observations sur l'état de l'art nous ont porté à croire que les solutions proposées ne correspondaient pas à nos besoins. Les outils de modélisation

disponibles sont limités à une sphère bien spécifique (ex : utilisation de ressources CPU, E/S de disques, trafic réseau, etc.) et se concentrent essentiellement sur des modèles de distribution et de probabilités relatives à la charge de ressources dans un système donné (ex : distribution d'utilisateurs suivant la loi normale dans un système avec, pour chacun, une probabilité d'utilisation d'une tâche spécifique avec un profil de charge fixe). Cela s'avère particulièrement problématique pour un environnement infonuagique, où la répartition des tâches est distribuée entre plusieurs machines virtuelles, et donc entre plusieurs ressources CPU, réseau, serveurs, etc. De plus, aucune des solutions proposées n'offre de mécanisme de détection de cycles dans les jeux de données de ressources systèmes. Ce mécanisme peut offrir la possibilité de classifier les comportements des ressources sous certaines charges de trafic, essentielles pour bâtir des profils de charges et, ainsi, de permettre la détection d'anomalies dans le comportement des ressources dans des environnements opérant en temps réel.

0.3 Objectif

Pour donner suite aux problématiques énoncées dans la section précédente, l'objectif visé et la portée de notre projet seront donc de concevoir un outil innovant de génération de charges de trafic pour des environnements de communication virtualisés. Cet outil doit impérativement répondre aux exigences suivantes :

1. L'outil doit être générique de nature; il doit offrir la possibilité de modéliser n'importe quel type de ressource (CPU, RAM, trafic réseau, E/S disques);
2. L'outil doit se concentrer sur la modélisation des ressources système, sans considération pour le nombre d'utilisateurs ou de tâches dans le système;
3. L'outil doit comporter un mécanisme de détection de cycles, permettant la classification de comportements de charges de trafic en profils de charge;
4. L'outil doit être entièrement automatisé; un utilisateur devrait simplement sélectionner un jeu de données en entrée;

5. L'outil doit générer des charges de trafic synthétiques les plus fidèles aux charges de trafic réelles.

0.4 Méthodologie

Dans le but d'atteindre les objectifs visés, ce projet sera divisé en deux axes de travail. Le premier axe se consacrera à la conception d'un outil de modélisation et de génération de charges de trafic, tandis que le deuxième se consacrera à la conception d'un mécanisme de détection de cycles dans des charges de trafic. Chacun de ces aspects étant complémentaires, ils seront intégrés dans un outil unique bénéficiant des points énoncés dans la section précédente. Les expérimentations seront réalisées à l'aide de jeux de données CPU provenant d'environnements virtualisés industriels et de cas d'utilisations réels, nous permettant ainsi de tester les mécanismes que nous aurons conçus. Cela nous permettra d'offrir, ultimement, une solution potentiellement commercialisable et applicable dans l'industrie. Au cours de ce projet, nous aurons bénéficié de jeux de données provenant de domaines différents : certains proviennent d'environnements infonuagiques du secteur des technologies de l'information (Google, application web virtualisée provenant de l'ÉTS (École de technologie supérieure)) et d'autres du secteur des télécommunications (système IMS (« *IP Multimedia Subsystem* ») virtualisé). Nos algorithmes sont testés et conçus à l'aide de Matlab R_2017b et les performances ont été évaluées sur un serveur ayant un processeur Intel Core i7 cadencé à 2.3 GHz et 16 Go de RAM.

0.5 Contributions techniques

Vues l'étendue et la complexité de la tâche reliée à la conception d'outils de génération de charges de trafic et de gestion de ressources dans des environnements virtualisés, il fût nécessaire de diviser ce projet en deux étapes distinctes. Nous avons donc planifié la réalisation de notre outil autour de deux axes principaux, qui furent réalisés séquentiellement et résultèrent en l'élaboration de deux approches innovantes. Ces dernières furent sujettes à la

rédaction de deux articles de journaux, ainsi qu'à l'application provisoire de deux brevets. Ainsi, les principales contributions de ce mémoire sont :

1. Un algorithme de modélisation de charges de trafic pour une estimation efficace de la consommation de ressources dans des environnements virtualisés.
2. Un algorithme de détection de périodicité dans des charges de trafic permettant la détection de profils de consommation de ressources de différentes formes, différentes amplitudes et à des intervalles de temps variables.

Tout d'abord, notre première contribution se concentre sur l'aspect de la modélisation de charge de trafic dans des environnements virtualisés. Ce fût le sujet de notre premier article, intitulé « Advanced Workload Modeling for Cloud Systems », soumis au journal "*IEEE Transactions on Services Computing*" en mars 2018. De plus, l'approche qui y est proposée, nommée « Hull-White-GA workload modeling », fût sujette à une application provisoire de brevet, dont nous sommes propriétaire, au USPTO (« United States Patent and Trademark Office ») par l'unité de brevets de Ericsson Canada en février 2018. Vous retrouverez l'article documentant cette approche au Chapitre 2.

Notre seconde contribution s'articule autour de l'aspect de la détection de périodicité de charges de trafic dans des environnements virtualisés. Cet aspect fût le sujet de notre second article, intitulé « Workload Periodicity Detection for Cloud Systems », soumis au journal "*IEEE Transactions on Cloud Computing*" en avril 2018. L'approche qui y est proposée, nommée « Advanced Workload Periodicity Detection Algorithm », fût également sujette à une application provisoire de brevet, dont nous sommes propriétaire, au USPTO par l'unité de brevets de Ericsson Canada en avril 2018. Vous retrouverez l'article documentant cette approche au Chapitre 3.

a) Cédric St-Onge and Nadjia Kara. GA-Based Hull-White Workload modeling for cloud systems. Application Patent reference number P74171, March 2018.
b) Cédric St-Onge and Nadjia Kara. Generic workload periodicity detection algorithm for virtualized systems. Application Patent reference number P74351, April 2018.

c) Cédric St-Onge, Souhila Benmakrelouf, Nadjia Kara, Hanine Tout, Claes Edstrom and Rafi Rabipour. Advanced workload modeling for cloud systems. IEEE Trans. On Services Computing Journal (Submitted), April 2018.
d) Cédric St-Onge, Nadjia Kara, Omar Abdel Wahab, Claes Edstrom and Yves Lemieux. Workload periodicity detection for cloud systems. IEEE Trans. On Cloud Computing Journal (Submitted), April 2018.

0.6 Organisation du mémoire

Sachant que ce document est structuré par articles, nous présentons une revue de la littérature générale suivie par deux chapitres dédiés à chacune des contributions. Dans chaque chapitre, vous retrouverez une discussion des résultats, une conclusion générale dans laquelle nous faisons un bilan des thèmes et des propositions abordés au cours de notre projet. Le chapitre 4 présente une discussion au sujet des principaux résultats obtenus. La conclusion et les travaux futurs sont présentés à la fin de ce mémoire.

CHAPITRE 1

REVUE DE LITTÉRATURE

La modélisation de charges de trafic et la détection de cycles dans des séries chronologiques sont deux sujets qui ont soulevé l'attention de nombreuses équipes de recherche au cours des dernières décennies. Vous retrouverez entre ces lignes une revue de littérature concernant ces deux techniques ainsi qu'un mécanisme de transréversion de chaînes binaires (« *prefix transreversal* »), jusqu'alors inutilisé dans le domaine de la détection de patrons, mais qui constitue un élément important de notre approche.

En ce qui a trait à la modélisation de charges de trafic, ce domaine en est un d'intérêt majeur dans le secteur des télécommunications depuis de très nombreuses années, favorisant l'optimisation des politiques de gestion d'infrastructures voix et IP dans le but d'en accroître la rentabilité. Parmi des exemples d'infrastructures de télécommunications ayant bénéficié de la modélisation de charge de trafic pour en optimiser les politiques de gestion, nous pouvons notamment citer des réseaux communément utilisés depuis l'avènement de l'ère Internet, tels que les réseaux ATM (« *Asynchronous Transfer Mode* ») (Dziong, 1997), MPLS (« *Multiprotocol Label Switching* ») (Minei, 2000) et VoIP (« *Voice over Internet Protocol* ») (Brunner & Ali, 2004). Ces derniers se basent bien souvent sur des modèles de trafic de charges de voix et données générés à l'aide de la théorie des files d'attente, puis de processus stochastiques comme l'AR (auto-régression), MA (« *moving average* ») ou une combinaison des deux (« ARMA ») pour créer des modèles de prévision (Dziong, 2010). La théorie des chaînes de Markov (Magalhaes, Calheiros, Buyya & Gomes, 2015), (Yang et al., 2012), (Moreno et al., 2013) y est également privilégiée pour faire l'analyse de coûts d'états dans le système et pour définir des politiques optimales. D'autres approches sont également préconisées dans le secteur des technologies de l'information, plus particulièrement dans un ouvrage de référence très exhaustif de Feitelson (Feitelson, 2014), où il est question de modélisation de charges de trafic se référant à n'importe quel domaine, type et attribut de

charge de trafic en lien avec des ressources informatiques. Dans son livre, l'auteur met particulièrement l'emphase sur des techniques de modélisation basées sur des distributions statistiques et de probabilités, qui sont encore couramment utilisées de nos jours sous la forme de la méthode de modélisation « *bag-of-tasks* » (Minh, Nam & Epema, 2014).

En ce qui concerne la détection de périodicité dans les charges de trafic, une méthode traditionnelle de détection de cycles, l'autocorrélation (Dziong, 2010), laisse graduellement sa place à de nouvelles techniques moins encombrantes et automatisées. Parmi celles-ci, nous retrouvons la méthode de *clustering* (Feitelson, 2010). Nous en retrouvons d'ailleurs de très bons exemples dans l'état de l'art, telle que l'approche préconisée par Prats, Berral & Carrera (Prats, Berral & Carrera, 2018), où l'équipe présente une méthode de détection de charges de trafic non-supervisée basée sur des modèles de Markov caché et à k-moyennes.

Finalement, le mécanisme de transréversion de chaînes de données par préfixe *(« prefix transreversal* ») est un algorithme de tri utilisé principalement pour obtenir des distances de transposition entre deux permutations (Dutta, Hasan & Rahman, 2013). Cette technique est utilisée, entre autres, pour estimer le nombre global de mutations entre des génomes et peut être utilisée par des biologistes moléculaires pour faire la déduction entre des relations évolutionnistes et fonctionnelles (Chen, Zheng, Fu, Nan, Zhong, Lonardi & Jiang, 2005). Plus récemment, certaines équipes de recherche se sont penchées sur le « problème de retournement de crêpes » (« *pancake flipping problem* »), où il est démontré avec des résultats quantitatifs comment les ratios d'approximation pour ce mécanisme s'améliorent avec l'augmentation du nombre de « *prefix transreversals* » appliqués par des algorithmes optimaux (Sharmin, Yeasmin, Hasan, Rahman & Rahman, 2010), (Hasan, Rahman, Rahman, Rahman, Sharmin & Yeasmin, 2015).

Dans ce document, vous retrouverez une revue de littérature plus détaillée dans chacun des chapitres se consacrant aux solutions que nous proposons. Vous retrouverez ainsi une revue de littérature dédiée à la modélisation de charges de trafic dans la section « *Related Work* »

du Chapitre 3, puis une revue de littérature se consacrant à la détection de périodicité dans des charges de trafic dans la section « *Related Work* » du Chapitre 4.

CHAPITRE 2

ADVANCED WORKLOAD MODELING FOR CLOUD SYSTEMS

Cédric St-Onge[a], Souhila Benmakrelouf[b], Nadjia Kara[c], Hanine Tout[d], Claes Edstrom[e] and Rafi Rabipour[f]

[a, b, c, d]Department of Software and IT Engineering, École de technologie supérieure,
1100 rue Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[e, f]Ericsson Canada,
8275 Route Transcanadienne, Ville Saint-Laurent, Québec, Canada H4S 0B6

Paper submitted to the journal "IEEE Transactions on Services Computing" in March 2018.
Provisional patent application P74171 filed by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USPTO) in February 2018

2.1 Abstract

Workload models are typically built based on user and application behavior in a system, limiting them to specific domains. Undoubtedly, such practice forms a dilemma in a cloud environment where a wide range of heterogeneous applications are running, and many users have access to these resources. The workload model in such infrastructure must adapt to the evolution in the system configuration parameters like job load fluctuation. The aim of this work is to generate generic workload models which are independent of user behavior and the applications running in the system, and that are able to fit any workload domain and type, able to model sharp workload variations that are most likely to appear in cloud environments, and with high degree of fidelity with respect to the observed data in a short period of execution time. In this work we propose two approaches for workload estimation. The first approach is a combination of Hull-White and Genetic Algorithm (GA), while the second one is a combination of Support Vector Regression (SVR) and Kalman-filter. Thorough experiments are conducted on real datasets to study the efficiency of both propositions. The

results show higher accuracy for the Hull-White-GA approach with marginal overhead over SVR-Kalman-Filter combination.

Index Terms: Cloud computing, Workload modeling, Workload estimation, Hull-White model, Genetics, Support vector regression, Kalman filter.

2.2 Introduction

With the ubiquity of Cloud Computing technologies over the last decade, Cloud providers and researchers have strived to design tools for evaluating and enhancing different Quality of Service (QoS) aspects of their systems, mainly performance, availability and reliability. Failing to comply to such aspects can compromise service availability and incur Service Level Agreement (SLA) violations, and hence impose penalties on cloud providers. The development of system management policies that support QoS is therefore crucial. However, the latter is a challenging task, as it must rely on evaluation tools which are able to accurately represent the behavior of multiple attributes (e.g., CPU, RAM, network traffic) of Cloud systems. It is also complicated by the essence of Cloud systems, built on heterogeneous physical infrastructure and experiencing varying demand; these systems have different physical resources and network configurations and different software stacks. Further, the reproduction of conditions under which the system management policies are evaluated, and the control of evaluation conditions are challenging tasks.

In this context, workload modeling facilitates performance evaluation and simulation as a "black box" system. Workload models allow Cloud providers to evaluate and simulate resource management policies aiming to enhance their system QoS before their deployment in full-scale production environments. As for researchers, it provides a controlled input, allowing workload adjustments to fit particular situations, repetition of evaluation conditions and inclusion of additional features. Furthermore, workload simulation based on realistic scenarios enables the generation of tracelogs, scarce in Cloud environments because of business and confidentiality concerns. Workload modeling and workload generation are

challenging, especially in Cloud systems, due to multiple factors: (i) workloads are composed of various tasks and events submitted at any time, (ii) heterogeneous hardware in a Cloud infrastructure impacts task execution time and arrival time and (iii) the virtualization layer of the Cloud infrastructure incurs overhead due to additional I/O processing and communications with the Virtual Machine Monitor (VMM). These factors make it difficult to design workload models and workload generators fitting different workload types and attributes. In the current state of the art, effort is instead deployed to design specialized workload modeling techniques focusing mainly on specific user profiles, application types or workload attributes.

To tackle the above issues, we propose in this article a hybrid workload modeling and optimization approach to accurately estimate any attribute of workload, from any domain. The objective is to develop realistic CPU workload profiles for different virtualized telecom and IT systems, based on data obtained from real systems. Our proposition consists first of modeling workload data sets by using different Hull-White modeling processes, and then determining an optimal estimated workload solution based on a custom Genetic Algorithm (GA). We also propose a combination of Kalman filter (Hu, Jiang & Wang,2014) and support vector regression (SVR) (Cortes & Vapnik, 1995) to estimate workloads. Our IMS workload data sets include three variations of the same load profile, where CPU workload is generated by stressing a virtualized IMS environment with varying amounts of calls per second, thus producing sharp increases and decreases in CPU workload over long periods of time. Our IT workload types take two different approaches. The Google CPU workload under evaluation, for example, is composed of sharp spikes of CPU workload variations over short periods of time. By contrast, another CPU workload, namely Web App, is characterized by periodicity trends. These datasets therefore display unique trends and behavior that provide interesting scenarios to evaluate the efficiency of the workload modeling and workload generation techniques that we are proposing in this paper. The evaluation of the mean absolute percentage error (MAPE) of the best estimated data provided by our Hull-White based approach against the observed data, shows significant improvement in the

accuracy level compared to other workload modeling approaches such as standard SVR and the SVR with a Kalman filter.

The main contributions of this work consist of:

- proposing a generic workload modeling approach fitting any workload domain (e.g., Telecom, IT) and any workload attribute (e.g., CPU, RAM, I/O);
- providing an automated workload generating tool capable of generating estimated workload with minimal user input;
- generating workload models without requiring knowledge of the inner behaviors of the modeled systems (i.e., "black box" approach);
- generating workload profile data while limiting dependence on external organizations for providing such data.

2.3 Background

With the rise of Cloud computing in the last decade, there has been an increasing amount of research conducted to help Cloud providers improve their systems performance, e.g. energy efficiency and Quality of Service (QoS). To achieve this goal, a common practice is to evaluate a system's workload. The notion of predictable workload denotes a "normal" and continuous usage of system resources. A non-predictable workload, on the other hand, is one which is triggered by a short, unexpected and extreme event. It can be, for example, a surge in calls being processed by an IMS system during an earthquake. In the context of Cloud computing, this is an important factor to consider since it has impact on how to handle resource allocation. In Shaw & Singh (Shaw & Singh, 2015), the authors propose different methodologies and allocation decisions based on both workload types. In the presence of predictable workload, for example, their algorithm is designed to invoke a virtual machine (VM) migration only if a fixed threshold is violated for a sustained time. In the presence of non-predictable workload, on the other hand, VM migration takes place when utilization is above a fixed threshold value. The use of predictable workload data is therefore preferred for

building workload models where a user wishes to forecast future load, while non-predictable workload data is preferred for building workload models where there are variations such as sharp, critical spikes in the load that might impact the system.

Workload modeling appears in many contexts and therefore has many different types (Feitelson, 2014). From the outset, it is therefore essential for someone planning to evaluate specific workloads to understand what domains, types and attributes of workloads is necessary to investigate. It is also important to study if the workload is taking the shape of a static or dynamic rate of events, and whether the workload is predictable or non-predictable. The following will explore all these concepts in the context of our research.

2.4 Related Work

In this section, we review existing works relevant to workload modeling to set the perspective for our contributions. The present review emphasizes on, but is not limited to, virtualized Cloud environments such as Google real traces and Open IMS datasets. This overview aims to give a broader picture of general types of workloads and workload modeling techniques that have been previously addressed by the research community

2.4.1 Workload Domains

Workload domain characterization is the first step and the uppermost level that should be considered before planning performance evaluation. It has a major impact upon what type of workload is to be considered. Domains vary in shape and scope, depending on the size and purpose of the environment; one can go from the performance evaluation of a single application on a workstation, to a full-scale performance evaluation of a multi-tenant, heterogeneous cloud environment (e.g., Amazon EC2). Google clusters and virtualized IP Multimedia Subsystem (IMS) cloud environments, both subjects of this work, are considered as workload domains belonging to the field of Cloud computing. Different relevant works have been cited in this context.

Moreno et al. (Moreno, Garraghan, Townend & Xu, 2013) provide a reusable approach for characterizing Cloud workloads through large-scale analysis of real- world Cloud data and trace logs from Google clusters. In this work, we address the same workload domain. There are, however, differences in their approach, starting with the dynamic behavior of their workload, in contrast with the static behavior of ours. The difference between a static and dynamic workload behavior is discussed next. MapReduce and Hadoop performance optimization (Moreno et al., 2013), (Yang, Luan & Li, 2012), (An, Zhou, Liu & Geihs, 2016) also bring interesting avenues because of the nature of the data intensive computing taking place in such environments, and because this framework is at the core of most of the leading tech company datacenters in the world, like Google, Yahoo and Facebook. This type of optimization, on the other hand, focuses on long-term analysis of predictable workloads and scheduled tasks, rather than quick bursts in demand for a specific application in a non-predictable manner. Performance evaluation of web and cloud applications (An et al., 2016), (Bahga & Madisetti, 2011), (Magalhaes, Calheiros, Buyya & Gomes, 2015) is another popular domain worthy of consideration. This domain usually involves the evaluation of user behavior, among other things, which is less prevalent in other domains. However, this domain is typically application-centric and rarely considers the workload characteristics of the whole cloud environment.

2.4.2 Workload Types

The next step when planning performance evaluation is to define what workload types to investigate. Feitelson (Feitelson, 2014) gives a good description of the constitution of a workload type. For instance, the basic level of detail of a workload is considered an instruction, many of which compose a process or a task. A set of processes or tasks can in turn be initiated by a job sent by a user, an application, the operating system, or a mix thereof.

In the field of cloud computing, the purpose of performance evaluation is mainly to optimize hardware resource utilization. There is therefore little to no variation in workload types, aside

from minor differences in philosophy or context from the researchers. Workloads typically originate from a mix of user, application and task workload types. For instance, the workloads studied by Magalhaes et al. (Magalhaes et al., 2015) and Bahga et al. (Bahga et al., 2011) are spawned from the behavioral patterns of specific user profiles using select web applications. A similar work by Moreno et al. (Moreno et al., 2013) uses a similar approach but focuses on the workload processed within a cloud datacenter driven by users and tasks. Studying these workload types can be useful in evaluating our own datasets, since the latter also depend on user behavior. In fact, user behavior is the prime factor in workload generation for our performance evaluations. Another approach proposed by An et al. (An et al., 2016) consists of viewing the user, the application and the service workload types as three layers of granularity. The users launch a varying amount of application-layer jobs, which in turn execute a varying number of service-layer tasks. This method is effective in the context of predictable, dynamic workloads. In some cases, only one workload type will be investigated, such as parallel processes and jobs under a specific cloud environment (Yang et al., 2012). Such work is intended to acquire very specific benchmarks from selected applications like MapReduce, and to evaluate optimal hardware configuration parameters in a cluster.

2.4.3 Workload Attributes

Workload attributes are the last step to consider when planning the performance evaluation of a system. Attributes are what characterize workload types, and directly influence hardware resources of the system. It is therefore essential to have a clear idea of how each instruction interacts with one or more resources from the system if you want to correctly interpret the workload data. For instance, I/O (disk and memory usage, network communications, etc.) attributes include the distribution of I/O sizes, patterns of file access and the use of read and/or write operations (Feitelson, 2014).

In this work, only the scheduling of the CPU is of interest. Hence, the relevant attributes are each job's arrival and running times (Feitelson, 2014). CPU scheduling is a common interest

in performance evaluation, and most studies analyze many workload attributes related to this resource. In An et al. work (An et al., 2016), the system CPU rate, threads, Java Virtual Machine (JVM) memory usage and system memory usage are analyzed. Similar attributes are considered in Magalhaes et al. work (Magalhaes et al., 2015), where the system CPU rate, memory rate and the users' transactions per second are evaluated. Other works are much more thorough in their analysis (Moreno et al., 2013), where attributes are subdivided in user patterns (submission rate, CPU estimation, memory estimation) and task patterns (length, CPU utilization, Memory utilization). Putting aside the differences in workload domains such as IMS and Google performance evaluation, we can safely assume that the system CPU behavior remains the same in both cloud environments.

2.4.4 Static and Dynamic Workloads

As Feitelson describes it: "An important difference between workload types is their rate of events. A static workload is one in which a certain amount of work is given, and when it is done that is it. A dynamic workload, in contrast, is one in which work continues to arrive all the time; it is never done." (Feitelson, 2014). Evaluation of static versus dynamic workloads will depend on the objective to be achieved. For instance, since jobs from a static workload are processed by a "clean" system, it usually implies the need to evaluate a system's behavior over a short time span. Jobs processed in a dynamic manner, on the other hand, require the evaluation of a system's behavior over a longer time span (several hours, days), since the same set of jobs are being processed by a system that is continually processing other jobs.

Static workload evaluation usually involves the analysis of a smaller number of workload types and attributes, increasing the complexity of the process. Since job execution times are not considered, it incurs a "drift" in variation in the rate of usage resources, since the system keeps processing jobs stacked in its queue while new jobs arrive. Dynamic workload evaluation overcomes this issue by adding a probabilistic and/or distributed (e.g., normal distribution, exponential distribution) approach to job arrivals and execution times that give a more accurate representation of the impact of workload types and attributes on the system

resources (Yang et al., 2012), (An et al., 2016), (Bahga & Madisetti, 2011), (Magalhaes et al., 2015).

2.4.5 Workload Modeling

The goal of workload modeling is to be able to create workloads that can be used in performance evaluations. Of course, the objective is to get a workload model as close as possible to the real workload: “Workload modeling is the attempt to create a simple and general model, which can then be used to generate synthetic workloads at will, possibly with slight (but well-controlled!) modifications” (Feitelson, 2014). It always starts with measured workload data and is a common alternative to using the traced workload directly to drive simulations.

To achieve the main objective of this work, which is to design realistic workload profiles for different virtualized telecom and IT systems, and also to provision cloud environment resources so as to fulfill service level agreements (SLA) as a use case of this work, we use workload modeling for a special case of performance evaluation; capacity planning. In a sense, capacity planning is performance evaluation in reverse. In other words, instead of deriving the performance of a given system configuration, we seek the configuration that will provide the desired performance (Menasce, Almeida, Dowdy & Dowdy, 2004). The most common approach found in the literature to create a workload model fitting our needs is to create a statistical summary of an observed workload. We apply this summarization to all the workload attributes (e.g., CPU, memory usage, I/O, etc.), then we fit distributions to the observed values of the different parameters (Moreno et al., 2013), (Bahga & Madisetti, 2011), (Magalhaes et al., 2015). In one case, for example, a statistical analysis of the user requests is performed to identify the right distributions that can be used to model the workload model attributes such as inter-session interval, think time and session length (Bahga & Madisetti, 2011). Four candidate distributions are then considered for each workload attribute based on efficiency: exponential distribution for inter-session arrival,

hyper-exponential distribution for modeling think times and inter-session intervals, Weibull distribution and Pareto distribution to model session lengths.

Other approaches applied to entirely different fields can also be of great interest. For instance, Tahmasbi and Hashemi (Tahmasbi & Hashemi, 2014) propose a model for forecasting urban traffic volume by using, among other things, the Hull-White model. This model is almost exclusively used in the finance sector to predict fluctuations in stock prices over a short period of time. In the case of short-term urban volume, the Hull-White model provided interesting results and we expect it does the same for our needs. Also noteworthy is the use of quadratic splines with irregular intervals as statistical summaries, and the use of the Wiener process as a distribution fit.

Other existing works have proposed workload prediction using linear regression (Lloyd, Pallickara, Lyon, Arabi & Rojas, 2013), Neural Networks (Islam, Keung, Lee & Liu, 2012), Kalman filter (Zhang-Jian, Lee & Hwang, 2014), (Wang, Huang, Qin, Zhang, Wei & Zhong, 2012) and support vector regression (SVR) (Wei, Tao, ZhuoShu & Zio, 2013). An optimal strategy for resource management should allow dynamic on-demand adjustment and provisioning of resources. This can be greatly facilitated if the workload can be predicted accurately. In this context, Hu et al. (Hu, Jiang, Liu & Wang, 2013), have proposed to address CPU usage estimation based on time series. Their approach is based on support vector regression and Kalman filter, particularly smooth Kalman filter, in order to remove the noise in the data, and reduce the prediction error rate. Higher weights are assigned to the latest data in the training set as such data provide more recent information on the behavior of the system. This idea was developed previously by Cao et al. (Cao & Tay, 2003), in their study of SVR with adaptive parameters in the prediction of time series in the financial domain. These approaches have outperformed prediction algorithms which are based on linear regression and neural networks, and even standard SVR.

2.4.6 Proposed Approach Motivation

Authentic industrial-grade workload data is sparse and often incomplete. Also, actual workload models are limited to very specific domains and are not flexible enough, as they cannot to be applied to a combination of domains such as IT and Telecom Virtualized Cloud environments. Further, the workload model must adapt to evolution in the system configuration parameters (e.g., Load, switched off CPU core). In this context, our proposed approach differs from other existing solutions in different aspects:

- combining Hull-White processes with GA automates the selection of segmented potential solutions through different models;
- GA reinforces our Hull-White process and optimizes our workload models by selecting the fittest solutions through many generations;
- significant improvement in execution time compared to other types of workload estimation models (e.g. auto-regression, moving average);
- accommodation of on-demand workload profile changes in a simulation thanks to adaptable and continuous spline functions.

2.5 Problem Illustration

One of the many challenges that arise when aiming to evaluate Cloud system policies, is the ability to get reliable data and tracelogs from organizations hosting Cloud environments. To circumvent this issue, workload modeling is introduced as a good alternative, enabling research teams to generate large amounts of workload profile data for use in the course of their work, while limiting dependence on external organizations for providing such data.

Many workload modeling solutions exist in the research community, but none of them answers the needs for a general purpose, generic modeling algorithm that can fit different types of telecom and IT workloads. Current workload models are limited as they are built on top of user and application behaviors of specific domains. While this might sound adequate

for a single web server where the number and the type of applications are limited, this is not the case in a cloud environment where a wider range of heterogeneous applications are running, and many users have access to these resources. The workload model must adapt to the evolution in the system configuration parameters (e.g., booting up a new VM or enabling new CPU cores to handle exceeding levels of resource loads).

Such issues motivate the need to generate generic workload models:

1. without having to know the specifics of user behavior and the deployed applications;
2. with high degree of fidelity with respect to the observed data;
3. able to efficiently estimate rapid workload variations and sudden/unpredictable changes.

2.6 Workload Generator Architecture

In this section, we give an overview of the proposed scheme for our workload generator. The novelty of our approach lies on the relationship between (1) the Hull-White workload modeling process and (2) a custom Genetic Algorithm (GA) involved in the workload generation and optimization process, as outlined in Figure 2.1. For Hull-White modeling, we estimate first the mean and the standard deviation of an observed workload data set in a chronological order. To get smooth and continuous functions of mean and standard deviation, our workload generator employs uniform and non-uniform quadratic spline curves. Next, fixed and variable theta values (θ, a Hull-White parameter) will be estimated. The full process of Hull-White modeling is explained in Section 2.7. The breakdown of segments for non-uniform splines and also for variable thetas are evaluated in an entirely automated process of the workload generator. The latter sets boundaries where there are large variations in the values of the mean and/or standard deviation over a short period of time. We thus obtain four different Hull-White models from which to generate our workload data, namely; (1) uniform splines and fixed theta, (2) uniform splines and variable theta, (3) non-uniform splines and fixed theta and (4) non-uniform splines with variable theta.

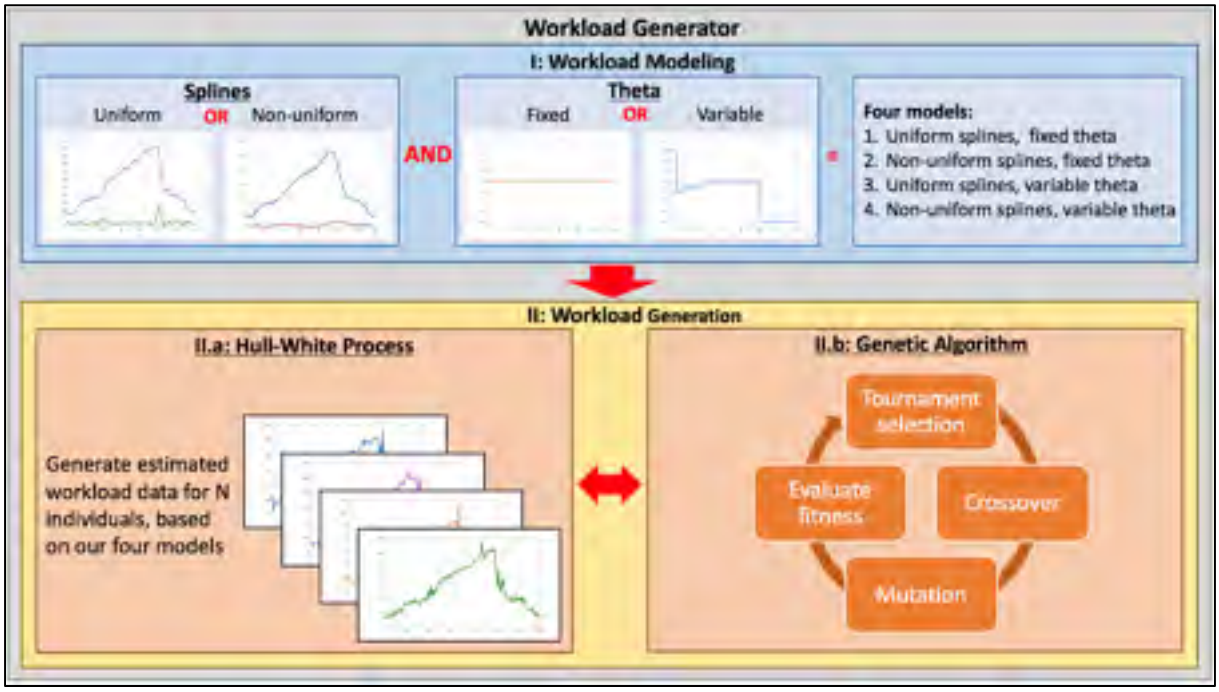

Figure 2.1 Proposed Workload Generator Scheme

On its own, the Hull-White modeling process proves to be an efficient algorithm to estimate workload data. However, by enhancing the Hull-White modeling with a custom GA optimization, our experiments demonstrate that the Mean Absolute Percentage Error (MAPE) of the estimated data generated by our approach is significantly improved, with minimal impact on the execution time. That proves to be especially true when pairing the benefits of the GA with the four afore-mentioned Hull-White models, each one providing different levels of fidelity, and excelling in different areas.

To briefly summarize the necessary steps to obtain estimated workload data through this workload modeling approach: (1) we provide discrete observed workload data as an input to the workload generator, (2) that workload data is processed and four Hull-White workload models are automatically generated and saved under a workload profile, (3) a workload profile is used in full or in part of a workload generation process and (4) the custom GA

generates estimated workload data from many instances of the corresponding workload profile(s) and proposes the fittest solutions.

2.7 Hull-White Workload Modeling

In this section, we present the formulation of the workload modeling problem and its underlying processes. The main objective of this process is to develop realistic workload profiles for different virtualized telecom and IT systems, based on data obtained from real systems, without requiring knowledge of their inner working processes (Black-box approach).

2.7.1 Stochastic Differential Equations (SDE)

From each consecutive samples of the real data, we generate mean and standard deviation values. SDEs are an excellent choice to model the time evolution of dynamic systems subject to random changes.

$$dX_t = \mu(t)dt + \sigma(t)dW_t, \qquad t \geq 0. \tag{2.1}$$

Where,

X_t : Observed workload

$\mu(t)$: Mean value of observed workload at time t

$\sigma(t)$: Variance value of observed workload at time t

W_t : Weiner process

2.7.2 Splines

Next, we generate splines for curve-fitting continuous mean $\mu(t)$ and continuous standard deviation $\sigma(t)$ values. Splines are used to estimate the mean $\mu(t)$ and standard deviation $\sigma(t)$ of a set of workload data in order to achieve smooth and continuous functions. We use both uniform and non-uniform splines. The first one uses knots (aka “anchor points”) set at

regular time intervals (ex: one knot every 20 seconds), while non-uniform splines use knots set at irregular time intervals.

$$\hat{f}(X_t(t)) = \begin{cases} a_i t_{i-1}^2 + b_i t_{i-1} + c_{i-1} = \hat{f}(X_{i-1}) \\ a_i t_i^2 + b_i t_i + c_i = \hat{f}(X_i) \\ a_i t_{i+1}^2 + b_i t_{i+1} + c_{i+1} = 0 \end{cases} \quad i = 1, \dots, n \tag{2.2}$$

Where,

$a_1 = 0$: First linear spline

$\hat{f}\big(X_t(t)\big) = \mu(t)$: Continuous mean

or

$\hat{f}\big(X_t(t)\big) = \sigma(t)$: Continuous standard deviation

2.7.3 Hull-White Process

The following describes the main properties of the Hull-White model, which is a popular choice of SDE in the finance sector for modeling future interest rates.

$$dX_t = (\mu(t) - \theta X_t)dt + \sigma(t)dW_t, \qquad t \geq 0. \tag{2.3}$$

Where,

X_t : Observed workload

$\mu(t)$: Mean spline value of observed workload at time t

$\sigma(t)$: Standard deviation spline value of observed workload at time t

θ : Estimated parameter

W_t : Weiner process

2.7.4 Estimation of θ

The parameter theta θ is a key feature of the Hull-White process. It is an estimated value that provides a drift (an upwards/downwards motion) to the estimated workload. In this work, we generate models with both, fixed and variable theta values. For fixed theta, we take the whole data from the data set and we calculate a single theta value. As for variable theta, we use time windows whose lengths depend on the variations in the data.

$$L_t(\theta) = \exp(-\int_0^T \hat{u}(t, X_T) dX_T - \frac{1}{2}\int_0^T \hat{u}^2(t, X_T) dt) \tag{2.4}$$

Where,

$$\hat{u}(t, X_T) = \frac{\mu(t) - \theta X_T}{\sigma(t)} \tag{2.5}$$

Therefore, the Maximum Likelihood Estimation (MLE) is defined as

$$\widehat{\theta_T} = \arg max\ L_T(\theta) \tag{2.6}$$

2.8 GA Workload Estimation

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection. It belongs to the larger class of evolutionary algorithms (EAs) and is commonly used in computer science to generate optimized solutions to complex search problems. To achieve this goal, a GA relies on bio-inspired operators such as mutation, crossover and selection to simulate the propagation of fittest individuals over consecutive generations.

2.8.1 Individuals

In GA, a population is a set of *n* individuals that form potential solutions. For workload estimation we define each individual/chromosome as a set of decimal values. Each

chromosome is an estimated workload value for the current workload attribute of the workload profile under evaluation. For instance, if we evaluate the CPU attributes of an IMS workload profile, each individual would be composed of chronologically and randomly estimated CPU workload values provided by any of the defined Hull-White models (Figure 2.2).

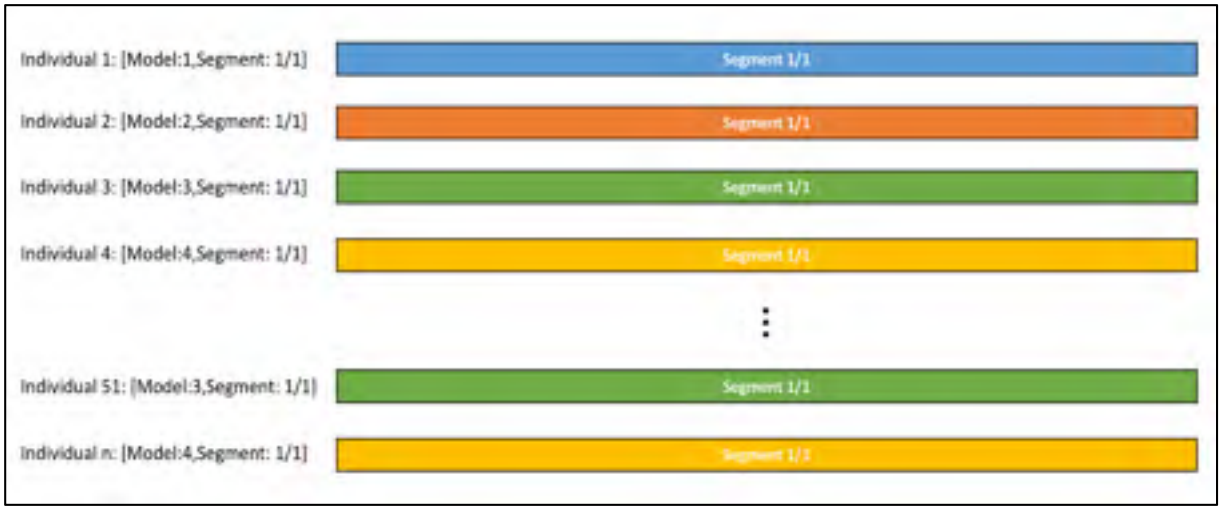

Figure 2.2 Individuals and Segments

2.8.2 Segments

An individual is divided into smaller segments, or genes, to go further in an evolution process of crossover and mutation operations. Each segment is of varying size and represents a portion of the individual (Figure 2.2). For instance, an individual of length L=100 can be divided into 4 even parts, hence creating 4 segments: segment 1 contains genes 1 through 25, segment 2 contains genes 26 through 50, etc.

2.8.3 Fitness Function

In GA, a fitness function is required to rank individuals in the current population. The score of an individual depends on how close its estimated values are to the observed values of a

workload profile. In the problem at hand, we evaluate the fitness of an individual by its mean absolute percentage error (MAPE) score:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{X_t - \hat{X}_t}{X_t}\right| \tag{2.7}$$

Where,

X_t : Observed value

$\hat{X}_t$: Estimated value

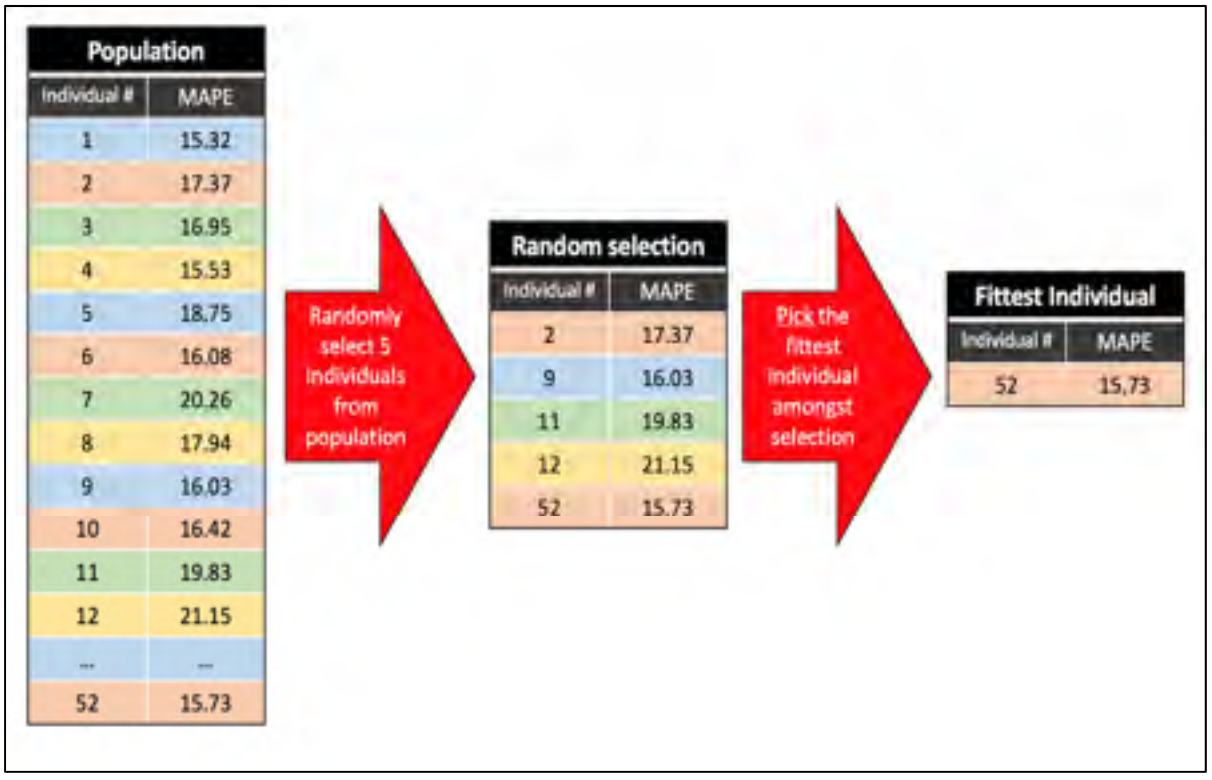

Figure 2.3 Selection Process

2.8.4 Selection

During each successive generation, individuals of the current population are selected to breed a new population (hence the term "generation"). For the purpose of our workload modeling problem, we proceed by tournament selection (Figure 2.3). To do so, we pick one candidate with the fittest solution amongst x randomly selected individuals. The process is then repeated to select a second candidate. Both candidates become parents to generate a new offspring in the crossover operator.

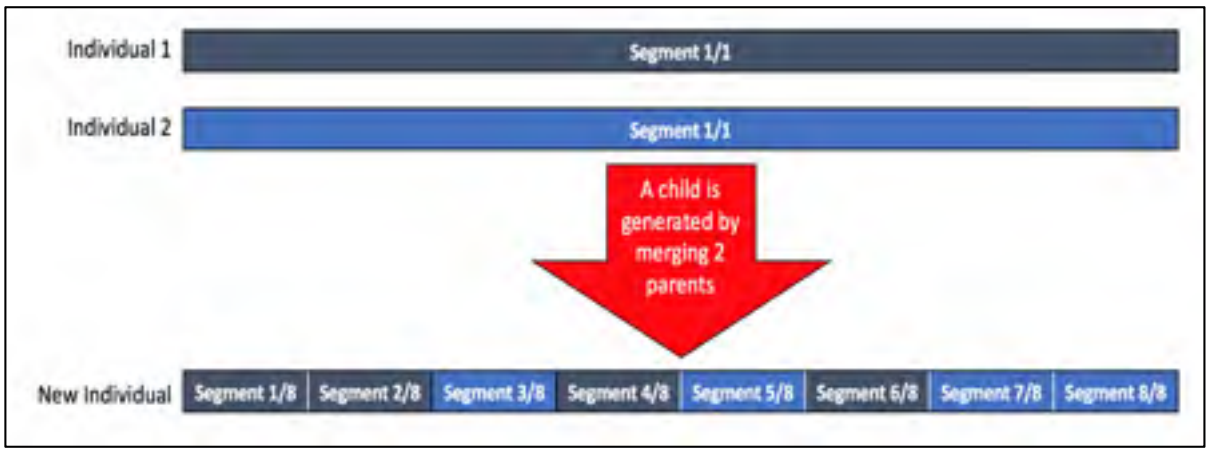

Figure 2.4 Crossover Operation

2.8.5 Crossover Operator

The crossover operator generates a next generation population of solutions from those selected through tournament selection. As shown in Figure 2.4, segments are chosen randomly, based on a uniform rate of ρ_c (probability of selecting segments from one parent over the other), from both parents I_1 and I_2 to generate a new offspring.

2.8.6 Mutation Operator

The mutation operator generates new estimated values from a randomly selected Hull-White model for randomly selected segments of an individual. It is based on a mutation rate of ρ_m (probability of mutating a segment). Figure 2.5 depicts how new workload estimation values of segments 4/8 and 6/8 are modified during the mutation process.

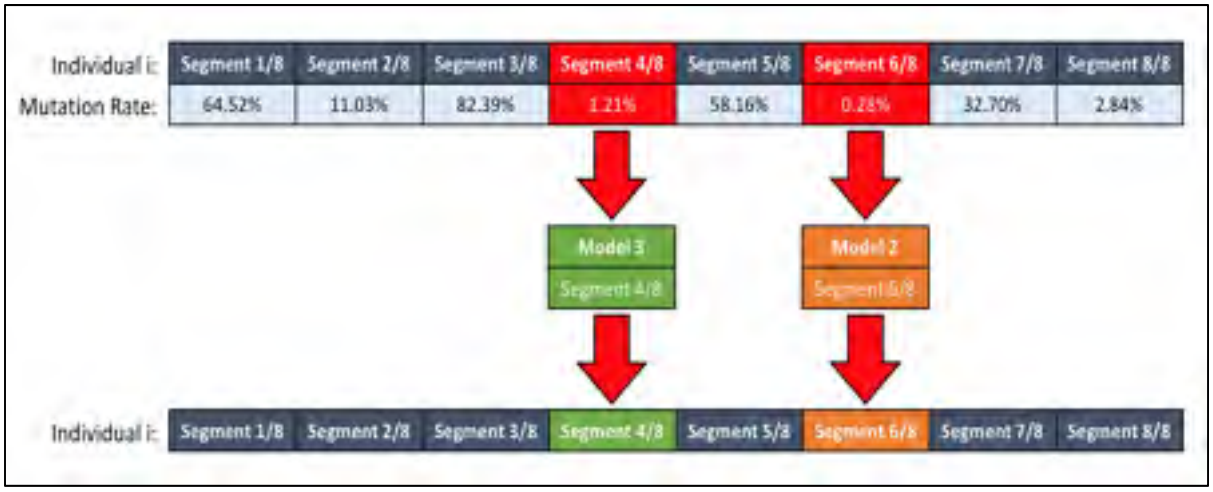

Figure 2.5 Mutation Operator

2.8.7 Algorithm

Algorithm 2.1 describes the proposed process for workload modeling with GA. It starts by generating a random population of candidate workload values (Line 4) then evaluates the fitness function, which is the MAPE for each individual (Line 5 to 11). Afterwards candidates are selected through a tournament process (Lines 15 and 16) to go into the evolution process of crossover (Line 17) and mutation (Line 20). The fitness of these individuals forming the new population is then calculated (Line 22 to 28). The process is repeated until the maximum number of generations (G) is achieved. Finally, the algorithm returns the fittest individual having the least MAPE found throughout the generated populations (Line 31).

Algorithm 2.1: *LoadModelingGA(L,M,N,D,W,G,T, ρc, ρm)*

```
1:  Input: L:= Individual length, M := Set of Hull-White models, N := population size,
    D:= Set of observed data, W := Set of segments, G:= Max generations, T := Tournament
    size, ρc:= crossover rate, ρm:= mutation rate
2:  Output: S:= fittest individual.
3:  Initialize population index k := 0
4:  Generate population Pk := GenerateRandomPop(N, L, M)
5:  for each individual ∈ Pk do
6:    e := Evaluate fitness of individual
7:    if e < fitness of current fittest individual then
8:      fitness of current fittest individual := e
9:      S := i
10:   end if
11: end for
12: while k < G do
13:   Insert S into Pk+1
14:   for i = 1 : i ≤ N do
15:     Pick candidate1 := tournamentSelection(Pk , T)
16:     Pick candidate2 := tournamentSelection(Pk , T)
17:     crossover candidate1 and candidate2 to produce new offspring and
        insert offspring into P_{k+1}
18:   end for
19:   for each individual ∈ Pk+1 do
20:     Mutate segments at random rate ≤ ρm
21:   end for
22:   for each individual ∈ Pk+1 do
23:     e := Evaluate fitness of individual
24:     if e < fitness of current fittest individual then
25:       fitness of current fittest individual := e
26:       S := i
27:     end if
28:   end for
29:   Increment k := k + 1
30: end while
31: return S the fittest individual
```

2.9 Kalman-SVR Workload Estimation

Beside Hull-White-GA, we also propose and study a combination of Kalman filter (Hu et al., 2014) and support vector regression (SVR) (Cortes & Vapnik, 1995) for workload estimation. Kalman Filter is a well-known model to estimate a hidden state x of system

indirectly from measured data and it can integrate data from as many measurements as available (Hu et al., 2014). The Kalman filter model is defined as follows:

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{2.8}$$

With a measurement z,

$$z_k = Hx_k + v_k \tag{2.9}$$

Where,

A : Transition matrix from time $k-1$ to k

B : Control matrix

u_{k-1} : Known vector

H : Matrix showing the relationship between z_k and x_k

w_{k-1} and v_k : Process and measurement noise respectively

After filtering observed workloads with the Kalman filter to remove the noise and hence minimize the prediction error, we use SVR to estimate workloads. In SVR, input data are separated into training data classes using linear hyperplanes. If they cannot be linearly separated, then input vectors (observed data) are mapped into high-dimensional feature space using non-linear mapping function (kernel function). SVR identifies the optimal hyperplane that maximizes the margin between the vectors of the considered classes. This optimal hyperplane is defined as linear decision function (find optimal parameters w and b):

$$f(x) = wK(x) + b \tag{2.10}$$

Where,

x : Input data, w is the weigth vector and b is the bias parameter

$K(x)$: Kernel function (e.g., linear, Radial Basis Function (RBF))

Algorithm 2.2: EM-KalmanFilter-SVR*(yi, m, n)*

```
1:  Initialize SVR parameters: Kernel function, C, Gamma
2:  Initialize data samples for Kalman Filter training and test: TrainData, TestData
3:  Create Kalman filter Object:
    kf = KalmanFilter(initial_state_mean=0, n_dim_obs=1)
4:  Estimate state with filtering: estimated_state = kf.EM(TrainData).KF(TestData)
5:  Estimate state with filtering and smoothing: smooth_state =
    kf.EM(TrainData).SmoothKF(TestData)
6:  for each m filtered data
7:     Filter-Prediction= SVR( estimated_state , m, n , KernelFunction, C, Gamma )
8:     SmoothFilter-Prediction= SVR(smoothed_state, m, n , KernelFunction, C, Gamma )
9:  end for
10: return filtered estimated data and smooth filtered estimated data
```

Algorithm 2.2 shows the proposed Expectation Maximization Algorithm (EM) EM-Kalman Filter-SVR. The algorithm starts by initializing SVR parameters (Line 1). The Kernel function model can be Radial Basis Function (RBF), linear, or polynomial. The parameter C trades off misclassification of training examples against simplicity of the decision surface, while Gamma can be defined as the inverse of the radius of influence of samples selected by the model as support vectors (RBF SVM Parameters, 2017). Then another initialization phase is conducted for the data samples of the Kalman filter. Next, the initial state mean is set to 0 while ndimobs, which is the size of observation space, is set to 1 (Line 3). The estimated state is calculated using EM (Lines 4 and 5). EM, is a meta-algorithm for learning parameters in probabilistic models. It aims to find parameters that maximize the expected likelihood of the observed variables. The EM algorithm is used in this work to estimate model parameters with the Kalman Filter. The algorithm estimates the state with the Kalman filter (Line 4), then estimate it with filtering and smoothing (Line 5). Afterwards, both the estimated state and the smoothed state are used in SVR for filter prediction, and smooth filter prediction, respectively (Lines 7 and 8). Finally, the algorithm returns the filtered estimated data and the smooth filtered estimated data (Line 9).

2.10 Experimental Results

In this section, we present the results of our experiments.

2.10.1 Use Cases

In our experiments we use CPU workload datasets from two different domains: IT and telecom. We selected these workloads because they differ significantly from each other and we want to evaluate the performance of our approach under different load behaviors, and to assess its general efficiency under various situations. For instance, the IT CPU workloads show sharper variations in short bursts while telecom workloads, on the other hand, show flatter, continuous loads under normal customer demand. This is due to the way that the CPUs handle the different jobs and tasks from each workload domain, as IT CPU loads involve distributed computing while telecom CPU loads generally comprise individual call setup activity. Under critical, unexpected customer demand, however, telecom CPU loads will have dramatic variations and can impact the whole system as customer calls may be dropped. To cover as many scenarios as possible, we use 5 different datasets; 3 from the telecom domain (IMS1, IMS2, IMS3) which have a similar configuration, but with slight variations in the amount of customer calls per second (CPS) generated, and 2 from the IT domain (Google85, ETS Web). The Google85 dataset comes from a single server from a cluster in a Google cloud environment and shows a standard behavior of a CPU workload under normal utilization in a cloud environment. The Web App dataset comes from a virtual machine hosting a web server at Ecole de technologie superieure (ETS) and shows periodicity (cycles) over one week of normal utilization.

2.10.2 Configuration

To demonstrate the proposed hybrid workload modeling approach, we created workload models based on observed CPU workload of virtualized IP Multimedia Subsystem (IMS) and Google clusters. The experiments were performed on a server with a 2.3 GHz Intel Core i7

quad-core processor, 16 GB of RAM, using Matlab R2017a. The following describe the configurations and scenarios used in this paper.

a. Configuration IMS1:
    1. Starts at 150 CPS (calls per second), increment: 50 CPS every 10 sec until 400 CPS
    2. 400 CPS constant during 100 seconds
    3. 600 CPS constant during 300 seconds
    4. 200 CPS decrement: -50 CPS every 50 sec until 50 CPS

b. Configuration IMS2: IMS1+50 CPS.

c. Configuration IMS3: IMS1-50 CPS

d. Configuration Google: CPU load of a single machine in a cluster captured and made available by Google.

e. Configuration Web Application: CPU load collected every 30 minutes from a virtual machine hosting a web server at Ecole de technologie superieure (ETS).

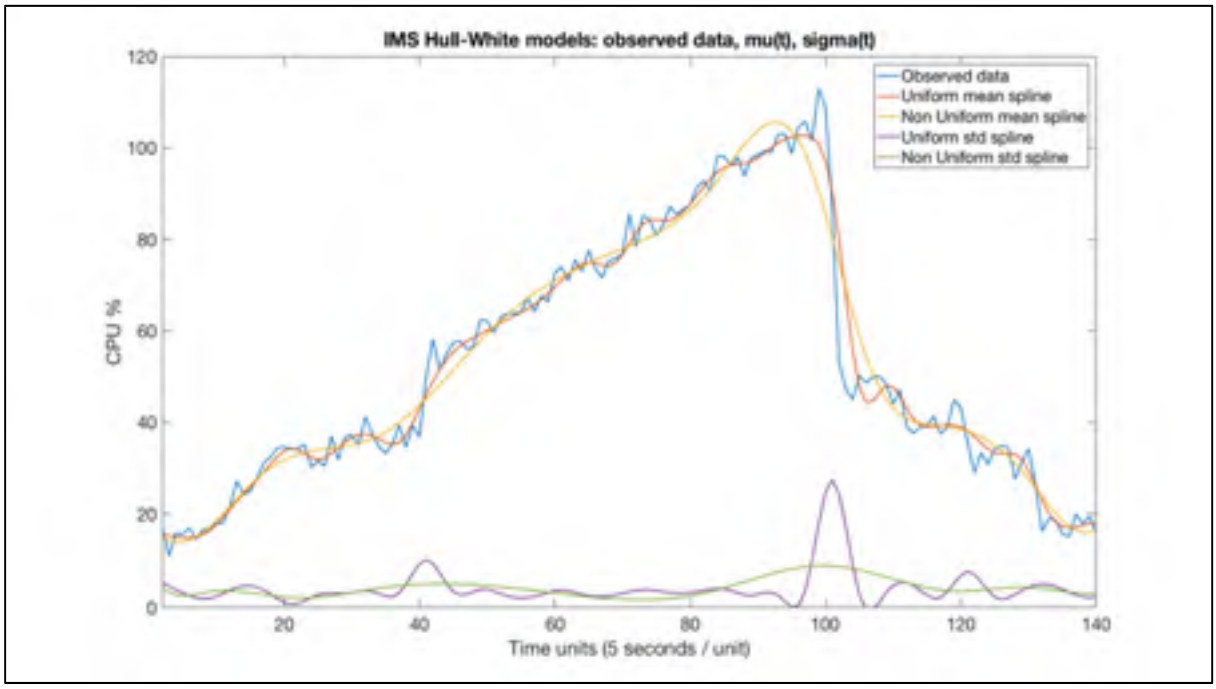

Figure 2.6 IMS Hull-White Models: Observed Data, μ(t) and σ(t)

2.10.3 Results

In the first set of experiments, we use four Hull-White models for workload modeling and then generate estimated workloads accordingly. In the second set of experiments, we combine Hull-White with GA, where the former is used to model workloads and the latter is used to generate optimized workload estimates. Figure 2.6, Figure 2.7, Figure 2.8, Figure 2.9 and Figure 2.10 depict the results for each dataset, while Table 2.1 illustrates the MAPE of the considered models. The results prove that the Hull-White combined with GA is able to generate workloads with the highest fidelity for all tested datasets.

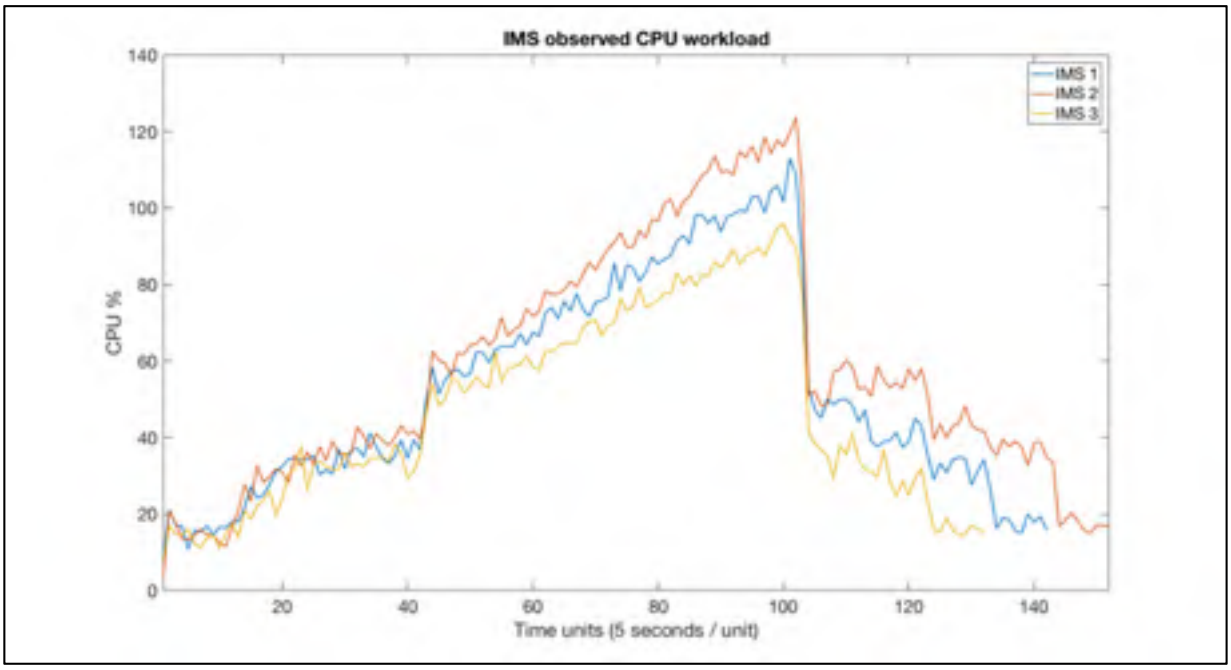

Figure 2.7 IMS: Observed CPU Workloads

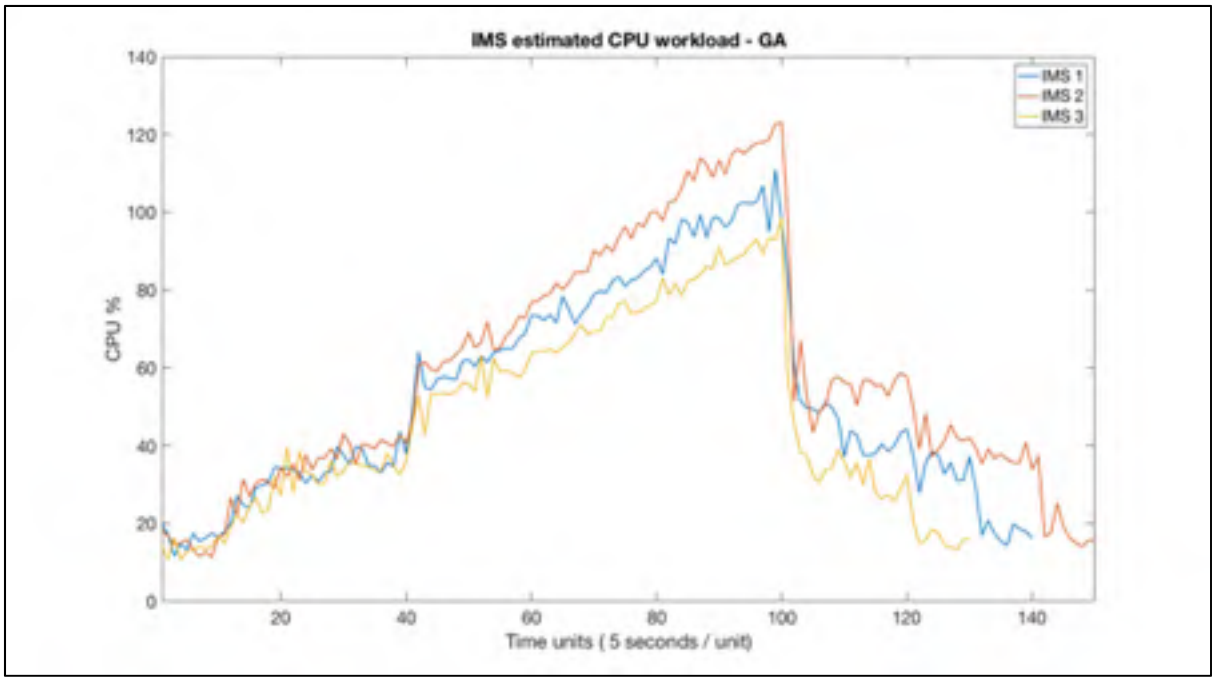

Figure 2.8 IMS: Estimated CPU Workload-GA

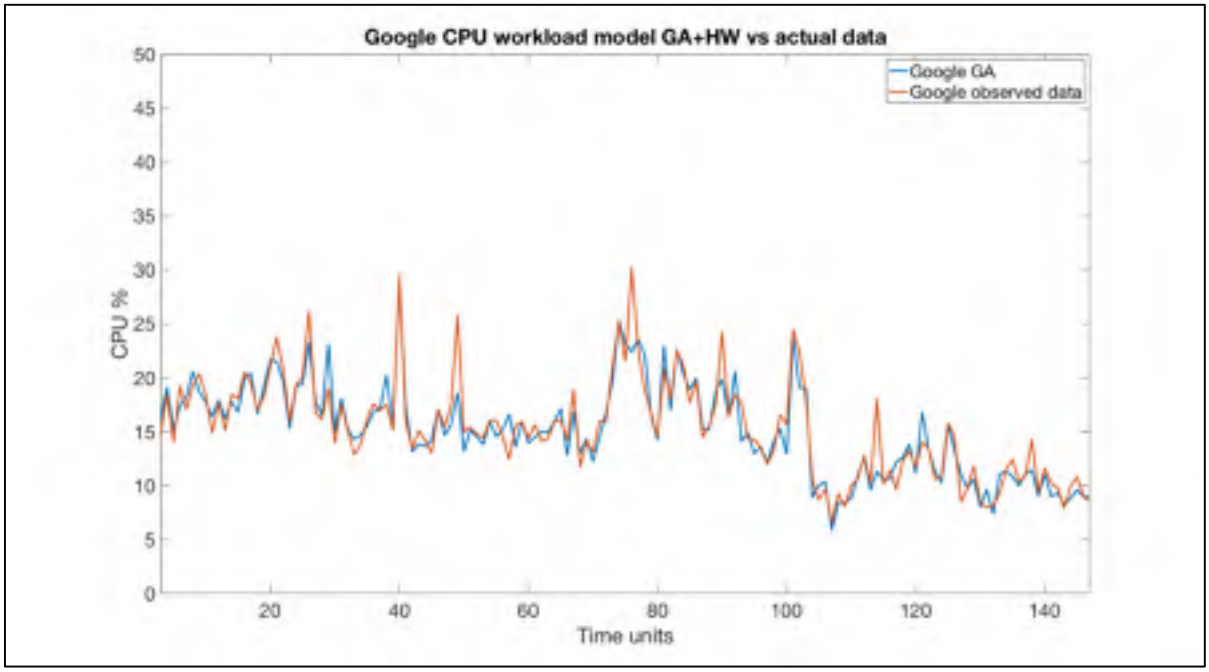

Figure 2.9 Google CPU Workload Model: Hull-White-GA and Observed Data

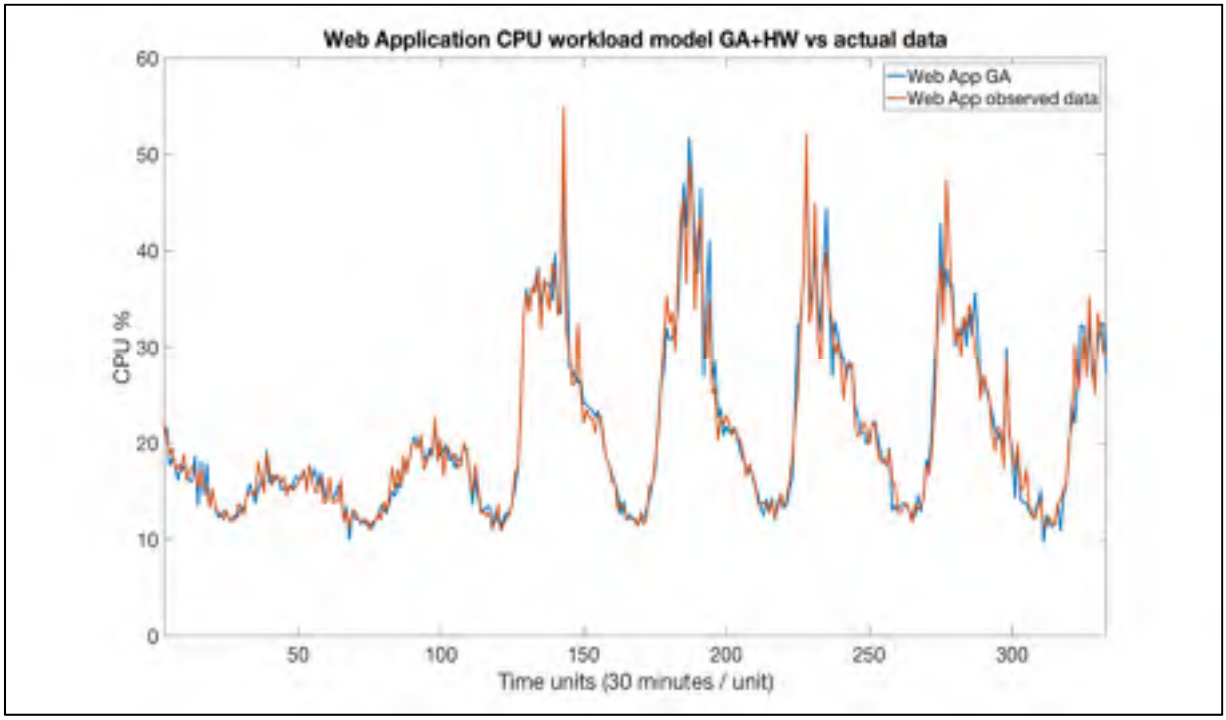

Figure 2.10 Web Application CPU Workload Model: Hull-White-GA and Observed Data

Figure 2.7 and Figure 2.8, show that the estimated workload fits the observed workload, with the exception of some areas where there are sharp CPU increases/decreases. Figure 2.9 shows that the estimated workload follows the same behavior as the observed workload. We notice, however, that the estimated workload is not an exact representation of the observed workload pattern. As from Figure 2.10, in the case of periodic workloads we observe a similar pattern. The estimated workload behavior, however, needs improvement. This is especially noticeable in areas where there are sharp spikes.

Further, we compare all these models with Support Vector Regression (SVR) and Kalman-SVR models. SVR allows data approximation based on statistical learning theory. In this SVR model, the prediction of future resource usage is based on observed data, which we divided into training and prediction sets to generate the estimated workloads. As for Kalman-SVR model, we filter observed data through the Kalman filter, then we predict using SVR. For SVR, we use four observations to train the model and two observations to estimate the next two values. The kernel function is set to RBF since it is more appropriate to nonlinear dataset, while C and Gamma values are fixed to 0.1. These parameters are set through extensive tests performed in order to find the configuration that minimizes the estimation error. As for the Kalman filter, we consider the transition as an identity matrix, we assume a vector of zero control input, and the noise measurement is of the state directly. Therefore, we set $A = 1$, $u = 0$, and $H = 1$ in Equations 2.11 and 2.12, which we define as follows:

$$x_k = x_{k-1} + w_{k-1} \tag{2.11}$$

$$z_k = x_k + v_k \tag{2.12}$$

Figure 2.11, Figure 2.12, Figure 2.13, Figure 2.14 and Figure 2.15 depict the results. Comparing SVR with Kalman-SVR, Table 2.1 proves the efficiency of the latter to generate more accurate results in all the datasets. Yet, the Hull-White-GA proves its efficiency over SVR and Kalman-SVM, showing higher accuracy for IMS and ETS Web datasets. As for the Google CPU workload, Kalman-SVR shows better results in terms of the least mean absolute

percentage error. On the other hand, we compare in Table 2.2 the execution time of all the studied models. The results show that Kalman-SVR outperforms the other models in all the datasets.

To summarize, these experiments show that Hull-White combined with GA is able to provide the highest accuracy for Workload modeling and estimation, but with higher overhead in terms of execution time compared to Kalman-SVR. Yet the latter loses efficiency proportionally to the window size and is more sensitive to large variations and peaks in the observed workload data.

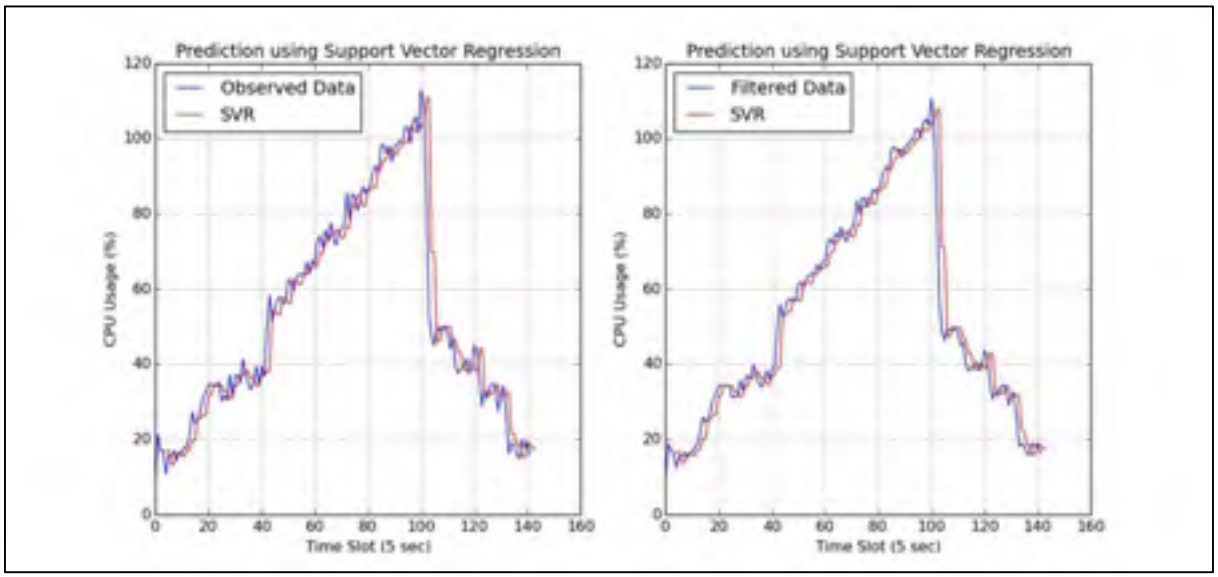

Figure 2.11 IMS1: SVR and Kalman SVR

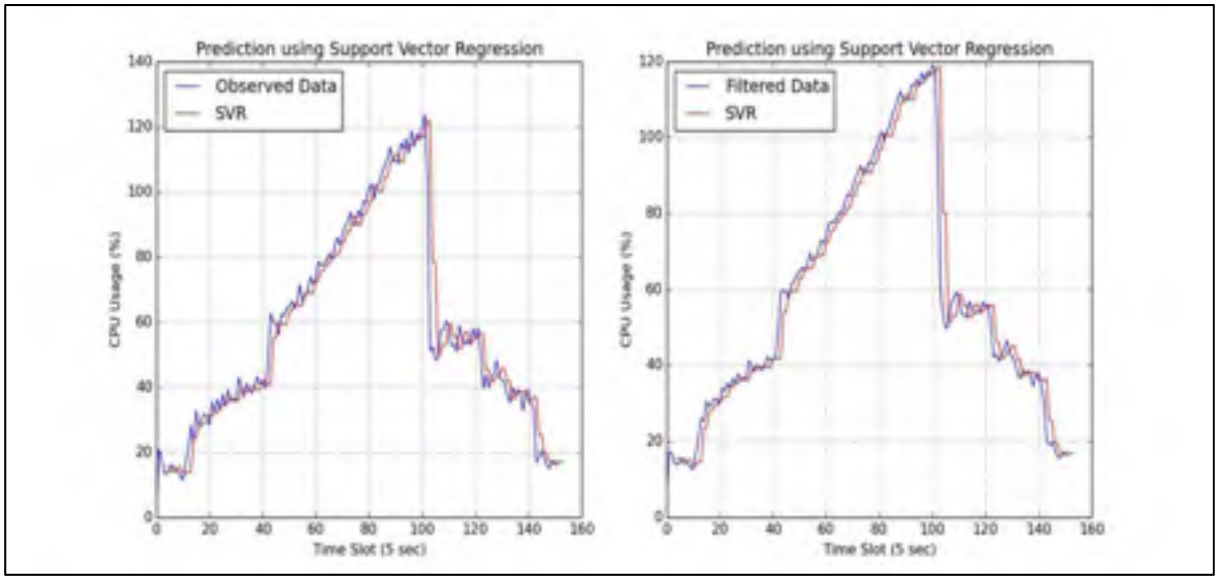

Figure 2.12 IMS2: SVR and Kalman SVR

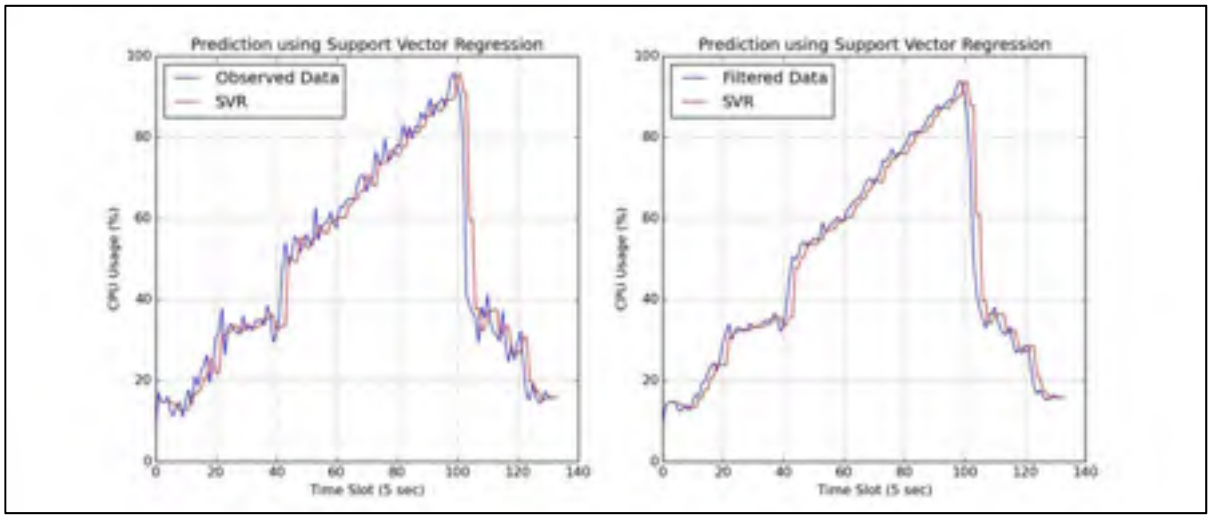

Figure 2.13 IMS3: SVR and Kalman SVR

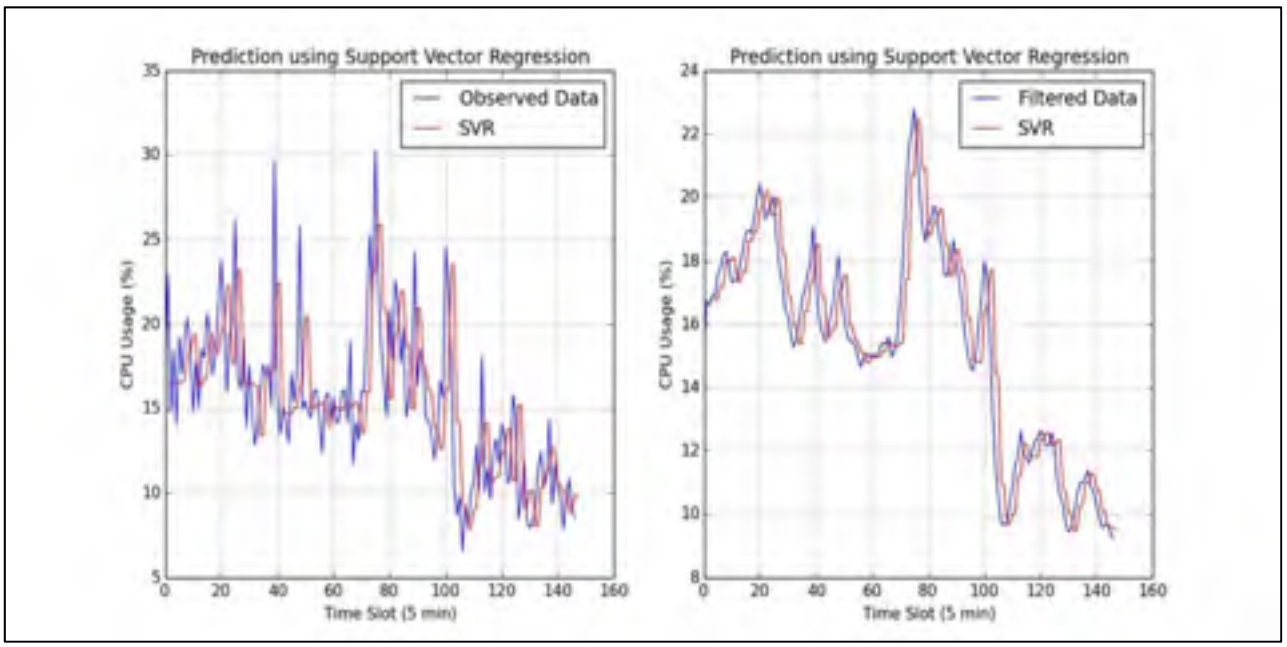

Figure 2.14 Google: SVR and Kalman SVR

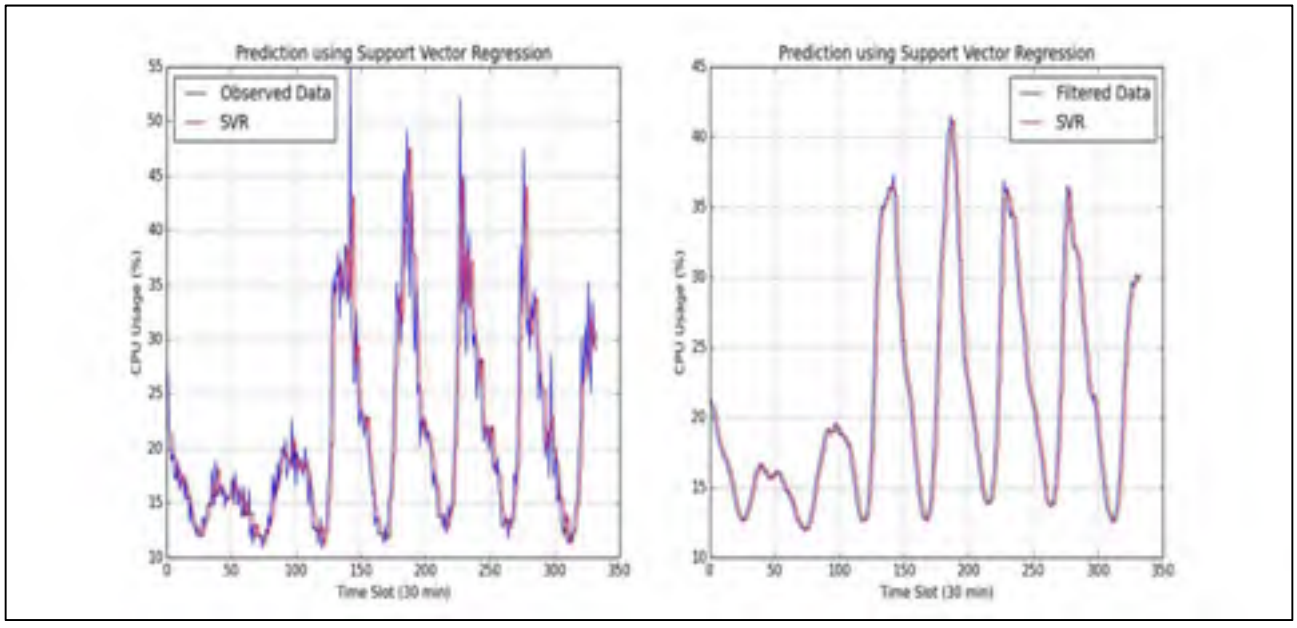

Figure 2.15 Web Application: SVR and Kalman SVR

Table 2.1 Average MAPE of solutions based on 10 simulations

Dataset	GA-Hull White	Hull-White Model 1	Hull-White Model 2	Hull-White Model 3	Hull-White Model 4	SVR	Kalman-SVR
IMS 1	**3,7625**	9,1578	10,8235	9,6495	10,8148	10.27	8.44
IMS 2	**3,931**	10,1615	14,0743	9,4488	14,3537	9.56	7.81
IMS 3	**4,3163**	10,6697	13,6215	10,7017	13,2126	11.15	7.48
Google	7,3226	18,5092	20,2361	17,7301	20,4648	19.19	**5.82**
Web App	**5,3969**	7,6307	9,0903	7,5744	9,1559	11.83	6.70

Table 2.2 Average execution time (s) based on 10 minutes simulations, 60 generations for GA

Dataset	GA-Hull White	SVR	Kalman-SVR
IMS 1	**2,77**	$9{,}60*10^{-3}$	$9{,}16*10^{-3}$
IMS 2	**2,80**	$10{,}11*10^{-3}$	$9{,}46*10^{-3}$
IMS 3	**2,63**	$9{,}19*10^{-3}$	$8{,}41*10^{-3}$
Google	**2,88**	$9{,}81*10^{-3}$	$8{,}99*10^{-3}$
Web App	**4,16**	$22{,}04*10^{-3}$	$21{,}46*10^{-3}$

2.11 Conclusion and Future Work

For dynamic on-demand adjustment and provisioning of resource needs in the Cloud environment, an accurate prediction of the system behavior is needed. The assessment of system behavior requires large amounts of workload data. To address the need for real workload data, something that is hard to obtain, we proposed in this article two novel paradigms for workload emulation, namely a Hull-White model combined with custom

genetic algorithm and support vector regression model optimized with Kalman filter. We evaluated both techniques on different datasets of IMS, Google and Web Application CPU workloads. The results have proved the advantage of Hull-White GA model over SVR and Kalman-SVM, showing higher accuracy for IMS and Web App datasets. As for the Google CPU workload data, Kalman-SVR showed better results in terms of the least mean absolute percentage error. However, for all datasets, Hull-White GA has outperformed both standard SVR and Kalman-SVR with negligible execution time of 0.00181 seconds. Such promising results open the door for a valuable track to examine the proposed hybrid workload modeling approaches on other workload attributes such as RAM and network traffic.

2.12 Acknowledgement

This work has been supported in part by Natural Science and Engineering Research Council of Canada (NSERC), in part by Ericsson Canada and in part by Rogers Communication Canada.

CHAPITRE 3

WORKLOAD PERIODICITY DETECTION FOR CLOUD SYSTEMS

Cédric St-Onge[a], Nadjia Kara[b], Omar Abdel Wahab[c], Claes Edström[d] and Yves Lemieux[e]

[a, b, c]Department of Software and IT Engineering, École de technologie supérieure,
1100 rue Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[d,e]Ericsson Canada,
8275 Route Transcanadienne, Ville Saint-Laurent, Québec, Canada H4S 0B6

Paper submitted to the journal "IEEE Transactions on Cloud Computing" in April 2018.
Provisional patent application P74351 filed by Ericsson Patent Unit Canada to the United States Patent and Trademark Office (USPTO) in April 2018

3.1 Abstract

Cloud computing systems are known for their elastic property which allows the dynamic addition and removal of resources based on the on-demand service model. However, the unsteady workloads and the variety of the applications running in the cloud entail the problems of resources over-provisioning and under-provisioning, which cause resources wastage and user experience degradation. In order to address this problem, workload periodicity detection can be used as a proactive decision-making approach to optimize the resource provisioning strategies and anticipate any performance problem. The existing workload periodicity detection approaches suffer from two essential limitations which restrict their effectiveness in cloud environments. Specifically, they require massive human intervention and are specific to particular types of workload data. To overcome these limitations, we propose in this paper a generic workload periodicity detection technique that employs the prefix transreversal approach from the molecular biology domain to detect cycles periodicity in cloud environments. The strengths of the proposed technique compared to the state-of-the-art lie in its generic nature in the sense that it can be applied to any type of workload and in its ability to detect patterns of varying lengths, amplitude, and shapes.

Experiments conducted on real datasets collected from Information Technology (IT) and Telecom domains reveal that our solution can improve the accuracy of the detections, especially in harsh environments wherein the lengths, shapes, and amplitude of patterns vary, compared to the auto-correlation technique.

Index Terms: Workload periodic patterns, periodicity detection, time series, prefix transreversal, workload modeling, cloud computing, time series analysis.

3.2 Introduction

Virtualization is the building block concept of cloud computing systems whereby different users running various applications share the same physical infrastructure for better resource utilization and reduced service cost. In order to guarantee the smooth functioning of virtualized systems and to optimize users experience, resources should be continuously and dynamically scaled to cope with the variations in the users demands. This necessitates developing a mechanism to monitor and analyze the behavior of the system in terms of resource utilization patterns, which would help cloud administrators proactively predict the future demands on the system and adjust the resource scaling strategies accordingly.

Workload periodicity detection has always been an active research area which is used hand in hand with data mining approaches to uncover hidden patterns in time-series data. Consequently, plenty of workload periodicity detection approaches which provide some interesting solutions have been proposed. These approaches, however, suffer from several limitations which restrict their performance, especially when it comes to cloud computing environments. First, these approaches require massive human intervention to specify the parameters and inputs of the detection approach (e.g., rate at which the time series is periodic). These demands require elaborating an automated detection approach which minimizes the human intervention and maintains at the same time optimal performance. Second, the existing approaches are effective only in situations wherein periodicity is of fixed length and shape, but become ineffective when the length, amplitude, and shape of the

patterns vary. This is a serious problem since, in real-life, the length and shape of the patterns vary depending on the underlying situation or period of time (e.g., weekdays vs. weekends). Third, the current approaches are designed to deal with datasets of specific types only (e.g., CPU utilization data), which limits their performance on datasets of other types. This requires thinking of a workload periodicity detection mechanism that is generic enough to be applied to different datasets of different types and hence avoid redesigning the solution whenever a new type of workload data is encountered.

3.2.1 Contributions

To address these limitations, we propose in this work a workload cycle detection approach for cloud computing systems that is generic enough to be applied to any type of discrete time-series data. Specifically, we capitalize on the prefix transreversal technique from molecular biology, which has shown its effectiveness in deducing evolutionary and functional connections in genomes. The proposed solution is composed of two essential steps, namely those of digital prints extraction and pattern detection. In the first step, workload data are considered as digital print and are converted into binary strings and stored in a prefix transreversal operation. In the second step, those digital prints are evaluated so that the digital print deviation for each data point is computed and compared to some predefined standard pattern (positive and negative) deviation shapes over a sliding window to detect periodic patterns. The outputs of this step are the number of cycles that have occurred, their length, shape, and amplitude. In summary, the main contributions of this paper can be summarized by the following points:

1. We propose a workload periodicity detection approach that takes advantage of the prefix transreversal technique from molecular biology to detect and identify periodic patterns in time-series data of any type. To the best of our knowledge, this work is the first that provides such a generic workload periodicity detection algorithm that can be applied to any type of discrete time series-based data.

2. The proposed workload periodicity detection technique is effective in detecting cycles without prior knowledge or assumption on the system thanks the mean spline generation process that we integrate into our solution.
3. The workload periodicity detection algorithm can detect cyclic patterns of any length, amplitude and shape. This overcomes the shortcomings of the existing approaches and provides a holistic solution that can be easily integrated into today's cloud computing systems.
4. The workload periodicity detection algorithm is entirely automated, where the only inputs are the dataset itself, evaluation pattern(s) to evaluate, and a proximity parameter. The evaluation pattern(s) and proximity parameters can be adjusted on-the-fly.
5. The workload periodicity detection algorithm can be used in offline as well as online simulations. This aspect is important to support the resource provisioning decisions in cloud systems in an efficient manner.

3.3 Background

Periodicity detection in time series has been the subject of active research work in the past decade, as research teams aim to discover different types of patterns. There are many real-life scenarios, like weather data, stock price movement, average precipitations, Cloud computing workload utilization, etc., where we can find such events that change with respect to time. A list of transactional data is known as time series database if it is stored with respect to an equal distant interval (Chanda, Ahmed, Samiullah & Leung, 2017). Finding patterns that periodically reappear in time series is therefore a topic garnering large interest in many fields of application, such as genetic biology (Ahdesmaki, Lahdesmaki, Pearson, Huttunen & Yli-Harja, 2005), workload anomaly detection (Wang, Wei, Zhang, Zhong & Huang, 2014) and transactional event analysis (Rasheed & Alhajj, 2010), (Rasheed & Alhajj, 2014), (Elfeky, Aref & Elmagarmid, 2005), (Chanda, Saha, Nishi, Samiullah & Ahmed, 2015), (Nishi, Ahmed, Samiullah & Jeong, 2013). Moreover, the process of predicting future events in time series databases helps in effective decision-making (Rasheed & Alhajj, 2010). Hence, many

algorithms have been developed by researchers to detect periodic patterns. The following section focuses on existing periodic pattern algorithms for time series.

3.4 Related Work

In this section, we discuss relevant related work in the areas of periodicity detection and workload pattern recognition using prefix transreversal operations.

3.4.1 Periodicity Detection Algorithms

In the literature, we consider that there are three types of periodic patterns that can be detected in a time series database: symbol periodicity, partial periodicity and segment or full-cycle periodicity. A time series is said to have symbol periodicity if at least one symbol is repeated periodically. Similarly, we have partial periodicity where a pattern (of length ≥ 1) in the time series is getting repeated periodically. Finally, if the time series is a result of approximate repetition of a segment of the series, it is said to be segment or full-cycle periodicity. While some algorithms focus exclusively on one type of periodic pattern detection, like symbol periodicity (Elfeky et al., 2005), partial periodicity (Wang et al., 2014) or full-cycle periodicity (Ahdesmaki et al., 2005), examples also abound where the three types are detected (Rasheed & Alhajj, 2010), (Rasheed & Alhajj, 2014), (Chanda et al., 2015), (Chanda et al., 2017) and (Nishi et al, 2013). Our approach described in this work also covers all three types of periodicity pattern detection.

Periodicity detection in time series data is an active research area and different approaches can be found in the literature on this topic (Rasheed & Alhajj, 2010), (Rasheed & Alhajj, 2014), (Ahdesmaki et al., 2005), (Elfeky et al., 2005), (Chanda et al., 2015), (Chanda et al., 2017) and (Nishi et al., 2013). Some approaches (Ahdesmaki et al., 2005) stem from the genetic biology field, where gene expression analysis aims to find periodicity in biological time series. In this work, the task of finding periodicity can be viewed as a decision problem based on spectral analysis together with multiple hypothesis testing. Even though the context of frequency spectrum analysis cannot be applied to workload data and, incidentally, differs

from our approach where we want to detect periodicity based on discrete data from observed workload, their method for estimating the cell cycle length/frequency by computing the average robust spectral estimate is particularly interesting and is a motivation to convert discrete data into continuous splines in our work, in order to get smooth, continuous re-discretized values.

Other approaches (Rasheed & Alhajj, 2010), (Rasheed & Alhajj, 2014), (Elfeky et al., 2005), (Chanda et al., 2015), (Chanda et al., 2017) and (Nishi et al., 2013) can be applied to detect periodicity in a wide range of time-series data, like weather data, stock price movement, computer network traffic density, gene expression, etc. It is interesting to note that all of these works follow similar rules, where the observed time-series data are discretized into symbols (e.g., values are rounded and converted into symbols of the alphabet) and hence symbol periodicity detection at the early stage of the pattern detections phase is applied. Some proposed algorithms such as the ones proposed in by Rasheed & Alhajj (Rasheed & Alhajj, 2010), (Rasheed & Alhajj, 2014) and Nishi et al. (Nishi et al., 2013) use suffix tree-based algorithms to classify periodic patterns in symbols and are effective in detecting the partial periodicity in time series. These algorithms start by building the suffix tree of a given time-series data, which helps in capturing the collection of occurrences of all the repeated patterns in the data. The algorithms then check if the repetitions are periodic and calculate the periodic strength of these patterns.

For instance, approaches proposed by Rasheed & Alhajj (Rasheed & Alhajj, 2010), (Rasheed & Alhajj, 2014), are efficient algorithms in periodic pattern detection where patterns are generated without user specifying period value. Their underlying structure based on STNR (Rasheed & Alhajj, 2010) consists of two phases. The first phase is a suffix tree construction that generates all suffixes of a sequence. In the second phase, periodicity detection is performed using the suffix tree structure. An important aspect of STNR is redundant period pruning. It searches events linearly and considers only consecutive events to calculate period value. Our approach, however, evaluates patterns based on deviation values compared with evaluation patterns, meaning that it has the ability to mine flexible patterns (the STNR

algorithm only detects fixed-length periodic patterns) of any amplitude and also categorize them, independent of their length.

The algorithm proposed by Nishi et al. (Nishi et al., 2013) is another example of approach leveraging suffix tree-based periodicity detection. The authors propose a sequential pattern mining approach that generates periodic pattern by allowing event skipping among intermediate events. Its main contribution is to aid the user to generate different kinds of patterns by skipping intermediate events in a time-series dataset and finding out the periodicity of the patterns within the database. At the initial step, single length patterns are mined. Later, larger length patterns of size K are gradually generated by joining interesting smaller length patterns of size K-1 at Kth step. The algorithm then generates all possible exclusive interesting patterns by skipping unimportant intermediate events by using occurrence vectors of already mined solid patterns. Finally, the generated patterns are tested for periodicity using a periodicity detection algorithm. However, due to its inability, like all other suffix tree-based algorithms, to skip any intermediate event in a generated pattern, it is impossible to generate the pattern which is a combination of the important and unimportant events from a user's point of view. To solve this issue, our approach leverages a proximity parameter which provides flexibility in pattern detection and eases the process of identifying and categorizing periods of similar shape in a time series.

Other algorithms proposed by Chanda et al. (Chanda et al., 2015), (Chanda et al., 2017) also apply symbolic pattern detection, but use a suffix trie (different from suffix tree) as a data structure. Both approaches require minimal user intervention to define periodic patterns and share features enabling the generation of periodic patterns with or without skipping unimportant intermediate events in time series databases with varying starting positions. In Chanda et al.'s work (Chanda et al., 2015), the algorithm simultaneously handles various starting positions throughout the sequences, thus fostering the flexibility among events in the mined patterns and enabling interactive tuning of period values on the go. The approach proposed in Chanda et al. in 2017 (Chanda et al., 2017) can mine three types of weighted periodic patterns (single, partial, and full) in a single run, making this proposition to generate

faster solutions and produce the most important patterns. While these approaches offer a good way to mine patterns, human interaction is necessary to manually set period length in order to generate patterns. Our approach, on the other hand, offers to possibility to detect patterns and evaluate the length of each period.

To summarize, all these proposals offer interesting solutions by discretizing data from a dataset into symbols. This process streamlines the pattern detection process by removing irrelevant values from the evaluation phase. However, the major shortcoming of these approaches lies in their inability to detect and classify cycles by shape and amplitude (i.e., a cycle of a certain shape may re-occur later in the dataset, but with higher or lower values, or even cycles of similar amplitude but with different shapes).

3.4.2 Workload Pattern Recognition based on Prefix Transreversal

Workload pattern recognition is another interesting research area which enables the detection of workload anomalies such as CPU intensive loops, memory leaks, disk I/O errors, and network anomalies. The approach proposed in Wang et al. (Wang et al., 2014) revolves around detecting faulty Web applications and locating fishy activities suspected to perform anomalous actions. Although the goal of this approach is not to propose a periodicity detection approach, their pattern detection algorithm (based on workload vectors to characterize dynamic workload) can be useful in defining combinations of amplitude, shape and length toward identifying anomalies in a dataset.

Other researches of interest center around prefix transreversal algorithms (Dutta, Hasan & Rahman, 2013), (Rahman & Rahman, 2015), (Dias & Dias, 2015), used by molecular biologists and bioinformaticians as a means to trace evolutionary distances between pairs of species. A basic prefix transversal problem can be stated as follows: Given two permutations, find a shortest sequence of rearrangement operations that transforms one given permutation into the other one. We have found in our work that by combining this type of computation with the binary strings technique in a novel approach, we could make a strong and flexible

pattern detection model that can recognize patterns regardless of their amplitude, shape and length.

3.4.3 Discussions and Comparison with the State-of-the-art

Based on historical observed data, none of the existing workload periodicity detection solutions answer the need to get a general-purpose, generic periodicity detection algorithm that fits different types of telecom and IT workloads. Further, the existing periodicity detection algorithms often require inputs from experienced individuals to tune certain parameters (e.g., specify the period size, select data points to detect repetitions in the dataset). Contrary to the literature, we automate the process of detection, thus enabling detection of patterns of any amplitude, shape and length, and without any human intervention. Additionally, none of the available periodicity detection solutions provide means to identify similar workload patterns in a dataset, which can provide valuable information to researchers aiming to improve load profiles for workload modeling aspects. Moreover, current approaches using auto-correlation to detect periodicity cannot offer a reliable workload periodicity detection solution, as auto-correlation is only useful for detecting periodicity at regular intervals in static time series.

In summary, our proposed approach differs from other existing solutions in the following aspects:

- Automated parameter selection: We propose an algorithm that evaluates the deviation patterns of the workload data to select the optimal pattern detection parameters.
- Detection of periodic patterns of any amplitude, shape and length: The proposed algorithm leverages on the prefix transreversal operations and a novel “digital print” extraction approach to screen the dataset and detect deviation patterns.
- Optimization of workload modeling approaches: The proposed algorithm enables the classification of cycles by amplitude, shape and length, thus allowing workload modeling specialists to define load profiles for various workload behaviors.

3.5 Problem Illustration

Among the many applications of workload periodicity detection, its ability to single out behavior and trends in time-series datasets is one advantage which can greatly benefit the workload modeling research area. Workload modeling could be an excellent alternative to evaluating Cloud systems' policies, for example. It enables the generation of large amounts of workload profile data that can be used as inputs for any organization's decision-making process, thus reducing its dependence on external sources for obtaining such data. Current workload modeling approaches, however, lack reliable tools to identify periodic cycles in datasets. Splitting periodic cycles and categorizing them by similarity following parameters such as amplitude, shape and length can be a great asset to researchers, allowing them to generate more accurate workload models for specific load profiles. Such issues motivate the need to create a workload periodicity detection algorithm which (1) enables the categorization of periodic cycles by amplitude, shape and length; (2) facilitates the detection of different periodic pattern types according to particular time periods (e.g., hourly, daily, weekly, etc.) or specific events (e.g., earthquake, emergency, entertainment event, etc.); (3) stores similar periods/cycles in a certain dataset and classifies them in the appropriate load profiles; and finally (4) adopts a generic approach and permits periodicity detection for large arrays of workload resources such as CPU rates, RAM utilization, disk I/O, and network traffic.

3.6 Architecture

In this section, we give an overview of the proposed workload periodicity detection approach. The novelty of our solution lies mainly in (1) the extraction of "digital print" values from workload data after having converted them into binary strings and sorted them in a prefix transreversal operation and (2) the pattern detection process which involves the evaluation of the digital prints deviation values through sliding windows alongside the dataset.

The workload periodicity detection algorithm is composed of two phases. In the first phase, called the “preparation phase”, the observed workload data from a certain dataset are adjusted and converted into easily classifiable parameters, which assists the pattern detection process in the second phase, called “decision phase”. An exhaustive description of the different steps of the preparation phase can be found in Chapter 3.7, and those of the decision phase can be found in Chapter 3.8. For the sake of clarity, a summary of each phase is provided below.

3.6.1 Preparation Phase

In the preparation phase, the first step is to read chronologic workload data in an offline fashion and to extract from this data the mean value from every batch of four sequential data samples to get uniform, noise-free discrete values. Next, to get a smooth and continuous mean function, we generate uniform quadratic spline curves by using the mean values obtained in the previous step. Thereafter, re-discretized values $\mu(t)$ from the spline are adjusted by multiplying them by 100 in order to get values ranging from 0 to 10,000. This step is necessary to get more significant values that will increase the accuracy of the pattern detection process in the detection phase. We then (d) convert each value from the previous step into binary strings, then (e) make prefix transreversal operations on each binary string. The final steps involve (f) extracting the “digital print” values from the transposed binary strings and (g) sorting the remainder of the binary strings to generate more parameters to be evaluated in the decision phase.

3.6.2 Decision Phase

Having obtained the different adjusted values and parameters from the preparation phase, the decision phase is employed to evaluate the digital print values for the sake of pattern detection. The pattern and periodicity detection methods proposed in this work involve (a) evaluating digital print values for deviations (see Chapter 3.8.1.), (b) comparing the fitness of the deviation values against “evaluation patterns” (sequential deviation values forming shapes, i.e.: sharp positive deviations, smooth negative deviations, etc.) over a sliding

window, (c) assembling pairs of positive deviation patterns and negative deviation patterns and identifying them as periodic patterns, and finally (d) counting the number of periodic patterns and identifying their shape, length and amplitude with the help of the digital print values and parameters obtained from the prefix transreversal operation.

3.7 Workload Data Estimation

In the following section, we give a brief discussion on the concepts of splines, prefix transpositions and digital prints, and show how we adapted them to our workload periodicity detection problem. The digital printing operation, in particular, is a novel approach derived from prefix transreversal operations, which fuels the pattern detection process and contributes greatly in solving several interesting challenges in our problem, namely detecting patterns of various lengths, amplitude and shapes.

3.7.1 Splines

In this phase, we generate splines for curve-fitting continuous mean *μ(t)* values. To achieve this goal, successive sequences of n workload data samples from a dataset are evaluated and the mean values are computed. This first step ensures that the noise in the data is removed in order to achieve smooth and continuous functions. Another advantage of this steps lies in the re-discretization of the values from the mean spline into uniform time-series values, thus fitting performance and/or accuracy requirements for the proposed algorithm.

$$\hat{f}(X_t(t)) = \begin{cases} a_i t_{i-1}^2 + b_i t_{i-1} + c_{i-1} = \hat{f}(X_{i-1}) \\ a_i t_i^2 + b_i t_i + c_i = \hat{f}(X_i) \\ a_i t_{i+1}^2 + b_i t_{i+1} + c_{i+1} = 0 \end{cases} \quad i = 1, \dots, n \tag{3.1}$$

Where,

$a_1 = 0$: (first linear spline)

$\hat{f}\big(X_t(t)\big) = \mu(t)$: Continuous mean

3.7.2 Prefix Transreversals

Transreversals is commonly used by molecular biologists to infer evolutionary and functional relationships in gnomes. The biological intuition behind this approach is that multiple "copies" of the same gene can appear at multiple places along a single genome. Its underlying mechanism is that the transposition distance between two permutations can be used to estimate the number of global mutations between genomes (Dutta et al., 2013). In this sub-section, we propose to adapt this concept to our periodicity detection problem.

3.7.2.1 Prefix Transreversals on Binary Strings

A string s is called k-ary (or having arity k) if the number of symbols occurring in it is $[k]$. Since, in this work, we apply prefix transreversal on CPU data samples converted into binary values, the set of strings is said to have arity 2 (we only use symbols {0,1}, "binary"). Moreover, a transreversal operation involves swapping two adjacent substrings. Hence, a prefix transreversal operation $f_p(1, x, y)$ on a data sample converted into a binary string $s = [s[n], s[n-1] \dots, s[2], s[1]]$ (since the power of numeric values increase from right to left) of length n, where $1 < x < y \leq (n+1)$, is a rearrangement event that transforms s into $[s[n], \dots, s[y], s[x-1], \dots, s[1], s[y-1], \dots, s[x]]$.

For example, let $s = 100101101$ and we want to apply the prefix transreversal operation $f_p(1,3,5)$. Now, $s[n], \dots, s[y] = s[9], \dots, s[5] = 10010$, $s[x-1], \dots, s[1] = s[2], \dots, s[1] = 01$, $s[y-1], \dots, s[x] = s[4], \dots, s[3] = 11$. Therefore, we get $s = 10010\mathbf{0111}$.

3.7.2.2 Estimation of *x* and *y*

The process of estimating parameters x and y in the operation $f_p(1, x, y)$ is entirely automated and is given by the following equations:

$$\sigma^2 = Var(|X_t - \mu(t)|) \tag{3.2}$$

$$x = \lfloor log_2(\sigma) \rfloor \tag{3.3}$$

$$\delta X_{max} = max(|X_{t+1} - X_t|) \quad \forall\, t = 1,2, \dots, n-1 \tag{3.4}$$

Where,

X_t : Observed data sample

n : Size of the data set

$$y = x + \left\lfloor log_2 \left(\left\lceil \frac{\delta X_{max}}{2^{x-1}} \right\rceil \right) \right\rfloor + 2 \tag{3.5}$$

3.7.2.3 Digital Print

Digital printing is a novel approach that has been proposed as an adjustment of the typical transreversal processes. It plays a key role in our workload periodicity detection problem, since it helps us determine the maximum number of step deviations as well as the grid size.

The "digital print" is a sub-string defined by $\rho_t = [s[y-1], \dots, s[x]]$ in the prefix transreversal operation $f_p(1, x, y)$. The digital print is extracted following the prefix transreversal's transposition process. The grid size and maximum number of step deviations are obtained using the following equations:

Grid size:

$$2^{y-x} \tag{3.6}$$

Max step:

$$2^{y-x-1} - 1 \tag{3.7}$$

As such, the maximum step helps determines if each digital print deviation is either positive or negative by ensuring that each successive value stay within reach. For example, with a

grid size of 32 and a max step of 15, a digital print at time $t = 8$ and a digital print at time $t+1 = 25$, we subtract digital print value t to value $t+1$: 25-8=17. The result, 17, is greater than the maximum step, 15, meaning that the deviation from t to $t+1$ is negative.

3.7.2.4 L-transreversal

The remaining string after a digital print extraction operation is then processed into a l-transreversal operation. This operation consists of reducing the string length by eliminating adjacent identical symbols, thus enabling the reduction of a string's length by l to obtain a normalized string.

Since binary values can only start with a value of $s[n] = 1$ (with the only exception of $s = 0$), there may be only two types of normalized binary strings in our approach:

1. Strings with odd length start and end with 1's (ex: 10101)
2. Strings with even length start with 1, end with 0 (ex: 1010)

For example, let $s = 1001001$. We find two adjacent identical symbols. Therefore, after applying the l-transreversal operation, we get the normalized string $s = 1\mathbf{00}1\mathbf{00}1 = 10101$ with a reduction length of $l = 2$.

3.7.2.5 Grouping Distance

The grouping distance $d_g(s)$ is the minimum number of prefix transreversal operations needed to reduce the length of the string be k. Since we are dealing with binary strings (i.e., $k = 2$), the two only possible binary pairs are '01' or '10'. As strings are normalized after an l-transreversal operation, only two kinds of binary strings are possible: 101…10 of even length or 101…01 of odd length.

The distance for normalized binary strings of size $n > 2$ is given by the following equation:

$$d_g(s) = \left\lceil \frac{n-2}{2} \right\rceil \tag{3.8}$$

We can always have a 2-transreversal if string $|s|$ is even. However, if $|s|$ is odd, we need an extra 1-transreversal. For example, let the normalized string $s = 10101$. We can apply a 2-transreversal as follows: $s = 10101 = 10\mathbf{1} \circlearrowleft \mathbf{0}1 = 10011 = 101 = \mathbf{1} \circlearrowleft \mathbf{0}1 = 011 = 01$. Here, we need one 2-transreversal and one 1-transreversal to group this string. So, $d_g(s) = [1,1]$.

3.7.3 Preparation Phase Algorithm

Algorithm 3.1 shows the proposed process of the preparation phase. It starts by generating continuous splines by using the workload data as an input, then returns discretized $\mu(t)$ values (Line 3). The next step is conditional to the type of workload data being evaluated. If it is CPU rate, the $\mu(t)$ values are multiplied by 100 to get discretized values, called *cpuRate[]*, ranging from 0 to 10,000 (Line 4 to 6). Afterwards, the *cpuRate[]* array is converted into binary strings (Line 7), and evaluated along with the original observed data to get *x* and *y* values for prefix transreversal (Line 8 to 10). The next step is to select positive and negative evaluation patterns by evaluating the binary strings, the *x* and *y* parameters (Line 11). The algorithm then proceeds by getting transposed strings and digital prints from prefix transreversal and digital print extraction operations (Line 12). The normalized strings and reduction length values are then obtained from a l-transreversal process (Line 13). The grouping distance of the normalized strings is then processed to get binary pairs and the number of 2-transreversal and 1-transreversal operations necessary to reduce the normalized strings into binary pairs (Line 14). Finally, the algorithm returns the following values (Line 15): *x* and *y* parameters, positive and negative evaluation patterns, the digital prints, reduction lengths, binary pairs and grouping distances. All of these values will further be processed in the evaluation phase, described in the next section.

Algorithm 3.1: preparationPhase(data[])

```
11: Input: data[] := workload data from a dataset
12: Output: x, y := prefix transreversal parameters,
    evalPatterns[] := positive and negative evaluation patterns,
    digitalPrint[] := digital print binary strings,
    l[] := reduction length of l-transreversal operations,
    binPair[] := binary pair following grouping distance operations,
    twoTrans[] := number of 2-transreversal operations made to reduce a normalized string to
    a binary pair,
    oneTrans[] := number of 1-transreversal operations made to reduce a normalized string to
    a binary pair.
13: Generate spline and get discretized μ(t) values:
    mu[]=generateSpline(data[])
14: If CPU rate, multiply by 100:
    for each i in mu[]
15:     cpuRate[i] = mu[i] × 100
16: end for
17: Convert to binary strings:
    binStr[] = dec2bin(cpuRate[])
18: Get x,y and evaluation patterns:
    x = floor(log2(std(abs(100×data[]-cpuRate[]))))
19: maxDelta = max(abs(cpuRate[i+1]-cpuRate[i]))
20: y = x + floor(log2(ceil(maxDelta/(2^(x-1)))))+2
21: evalPatterns[] = getEvalPatterns(x,y,binStr[])
22: Get transposed binary strings and digital prints:
    (transposedStr[],digitalPrint[]) = prefixTransreversal(x,y,binStr[])
23: Apply l-transreversal to get normalized strings and reduction length:
    (normStr[],l[]) = lTransreversal(transposedStr[])
24: Compute the grouping distance to get a binary pair, 2-trans and 1-trans distances:
    (binPair[],twoTrans[],oneTrans[]) = groupingDist(normStr[])
25: return x, y, evalPatterns[], digitalPrint[], l[], binPair[], twoTrans[], oneTrans[]
```

3.8 Workload Periodicity Detection

In this section, we discuss the decision phase of our workload periodicity detection solution. This process can be summarized in the following steps: (1) compute the digital print deviation $\rho_{t+1} - \rho_t$ for each element of the dataset; (2) define evaluation patterns (shapes) for positive deviations and negative deviations; (3) compare evaluation patterns with digital

print deviation values through a sliding window; (4) for each successive sequences of identified positive and negative patterns, a periodic pattern is detected; (5) the decision process then returns the number of cycles, their length, shape and amplitude.

3.8.1 Computing Digital Print Deviations

The step of computing digital print deviations of a dataset is useful in finding significant trends and patterns. This is achieved through analyzing these values which enables the pattern detection in our approach. We evaluate the deviations by subtracting each digital print value at time $t + 1$ from its predecessor at time t as given below:

$$\delta\rho_t = \rho_{t+1} - \rho_t \quad \forall\, t = 1,2,\dots,n-1 \tag{3.9}$$

Figure 3.1 shows an example of digital print deviation values for a dataset containing CPU workload data.

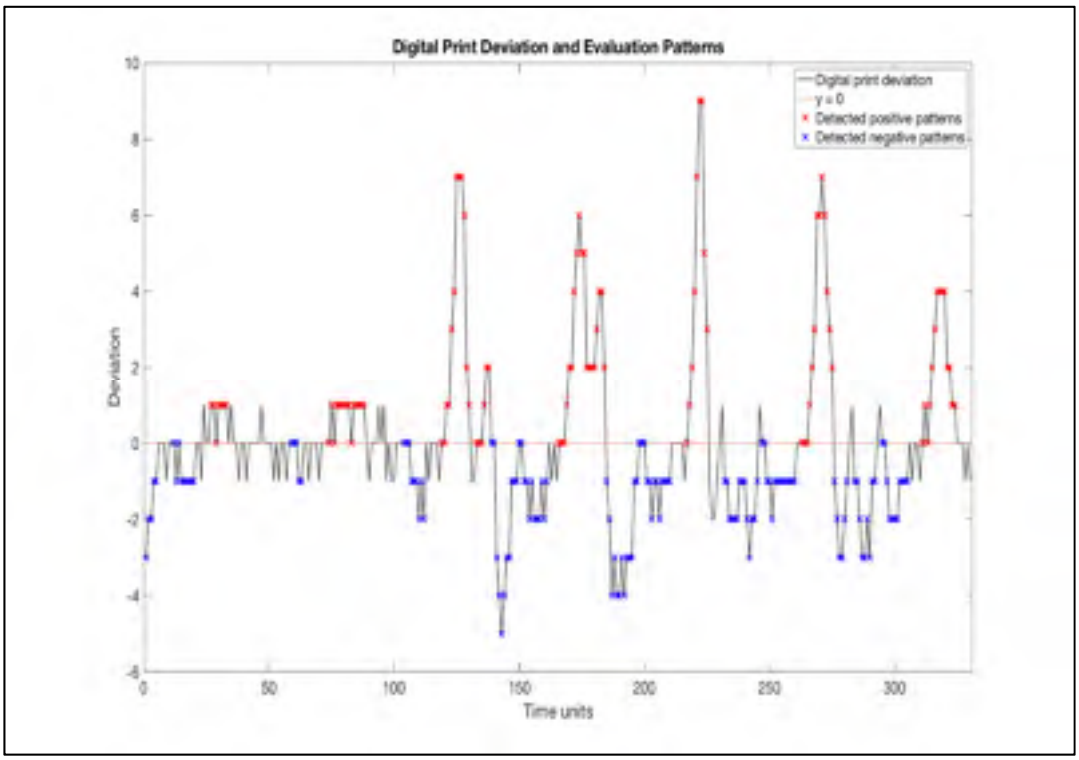

Figure 3.1 Digital Print Deviations from Web App Dataset

3.8.2 Evaluation Patterns and Sliding Windows

An evaluation pattern is a string that defines a deviation shape (or trend) that is compared, through a sliding window of the same size, with the digital print deviations of a dataset. A set of evaluation patterns are defined following a number of parameters, such as:

1. Slope sharpness (ex: sharp slope: smaller string size, sharper deviations, smooth slope: longer string size, smoother deviations);
2. Positive or negative deviation;
3. Proximity of all digital print deviation values to those of the evaluation pattern in a sliding window (ex: all digital print deviation values must be greater or equal to those of an evaluation pattern).

A simple manner to detect periodic patterns is to find consecutive pairs of positive and negative deviation patterns. We can, for example, define two evaluation patterns; one to detect smooth positive deviations $s_1 = [0,0,1,1]$ and one to detect sharp negative deviations $s_2 = [-1,-3]$. We then screen the dataset with two sliding windows such that:

$$[\delta\rho_t, \delta\rho_{t+1}, \delta\rho_{t+2}, \delta\rho_{t+3}] \geq s_1 \quad \forall\, t = 1,2,\dots,n-3 \tag{3.10}$$

And,

$$[\delta\rho_t, \delta\rho_{t+1}] \leq s_2 \quad \forall\, t = 1,2,\dots,n-1 \tag{3.11}$$

Each consecutive pair of similar patterns $[s_1, s_2]$ forms a cycle that can be further analyzed to obtain its length, amplitude and shape.

3.8.3 Algorithm of the Evaluation Phase

Algorithm 3.2 shows the proposed evaluation phase algorithm. The algorithm starts by initializing the grid size and maximum step (Line 3 and 4). It then proceeds to convert the digital prints' binary values into decimals (line 5). Next, a deviation matrix is generated by evaluating each consecutive digital print values (Line 6 to 19). The next step splits the *evalPatterns[]* array into two separate attributes: one for positive evaluation patterns and one for negative evaluation patterns (Line 20 and 21). The algorithm then proceeds with the pattern detection process. That process is realized by evaluating the digital print deviations in two distinct sliding windows. The first sliding window evaluates the digital print deviations with positive evaluation patterns and a proximity parameter (Line 22 to 28), and the other sliding window evaluates the same digital print deviations and proximity parameter with negative evaluation patterns (Line 29 to 35). After the pattern detection process, we evaluate pairs of positive and negative deviations previously detected by evaluating their location in the time series (Line 36). Matching pairs are then identified as sequential cycles and return their respective starting and ending data points, their length and shape (Line 37). Finally, the algorithm returns an array with detected cycles in numerical order, containing their position, length and shape (Line 38).

Algorithm 3.2: evalPhase(evalPatterns[],digitalPrint[],proximity)	
1:	**Input:** evalPatterns[] := positive and negative deviation evaluation patterns, digitalPrint[] := digital print binary strings.
2:	**Output**: cycle[] := array map of cycles including information about position, shape and length.
3:	Initialize grid size and max step number: gridSize = 2^(y-x)
4:	maxStep = (gridSize / 2) - 1
5:	Convert digital prints to decimal: decPrint = bin2dec(digitalPrint[])
6:	Build deviation matrix: **for each** i in decPrint[]-1
7:	deltaPrint[i] = decPrint[i+1] – decPrint[i]
8:	**end for**
9:	**for each** i in deltaPrint[]-1

Algorithm 3.2: evalPhase(evalPatterns[],digitalPrint[],proximity)

```
10:     if abs(deltaPrint[i]) > maxStep
11:        if deltaPrint[i] > 0
12:           deviation[i] = deltaPrint[i] – gridSize
13:        else
14:           deviation[i] = deltaPrint[i] + gridSize
15:        end if
16:     else
17:        deviation[i] = deltaPrint[i]
18:     end if
19:   end for
20:   Prepare sliding window:
      windowUp = even(evalPatterns[])
21:   windowDown = odd(evalPatterns[])
22:   j=1
23:   for i = 1 in size(deviation[])
24:     if deviation[i] >= windowUp + proximity
25:        detectedDeviationUp[j] =  I + size(windowUp) -1
26:        j++
27:     end if
28:   end for
29:   j=1
30:   for i = 1 in size(deviation[])
31:     if deviation[i] >= windowDown - proximity
32:        detectedDeviationDown[j] =  i + size(windowDown) -1
33:        j++
34:     end if
35:   end for
36:   Evaluate pairs of positive and negative deviations
      (posC[], lengthC[], shapeC[]) = evalDeviations (detectedDeviationUp[],
      detectedDeviationDown[])
37:   cycle[] = [posC[], length[], shapeC[]
38:   return cycle[]
```

3.9 Experimental Results

In this section, we present the results of our experiments.

3.9.1 Use cases

In our experiments, we use CPU workload datasets from two different domains, namely those of IT and telecom. Both domains were selected because they differ significantly from each other in terms of load behavior, general CPU rate variations attained in any given period of time, and periodicity type (hourly, daily, etc.). Moreover, this variability is important to ensure that we cover different situations and evaluate its efficiency in detecting cycles of different lengths, shapes and amplitude. For instance, the IT CPU workload data used in this work show sharper variations in short bursts while the telecom CPU workload data, on the other hand, show flatter, erratic and continuous loads under normal customer demands. This is mainly due to the fact that IT systems distribute the computing of jobs and tasks across many virtual machines and CPUs, as opposed to the telecom CPU loads, which generally handle a stack of individual call setup activities. To cover as many scenarios as possible, we employ four different datasets; two from the telecom domain (IMS1, IMS2) which have similar configurations, but also having variations in the number of customer calls per second (CPS) generated, and two from the IT domain (Web, Google). IMS1 and IMS 2 datasets show easily identifiable cycles of varying lengths, but of similar shape and were used in the first phase of our approach's design, mainly for tuning the pattern detection parameters of the algorithm.

The Web App dataset originates from a virtual machine hosting a Web server and shows a periodicity (daily cycles) over one week of normal utilization. The Google dataset comes from a single server in a Google Cloud environment and shows standard behavior of a CPU workload under normal utilization in a Cloud environment. For this dataset, periodicity is less obvious and varies greatly in length and shape.

3.9.2 Configuration

To demonstrate the proposed workload periodicity detection approach, we evaluated the observed CPU workload datasets of virtualized IP Multimedia Subsystems (IMS) and Google clusters. The experiments were performed on a server with a 2.3 GHz Intel Core i7 quad-core processor, 16 GB of RAM, and using Matlab R2017b. The following describes the configurations and scenarios used in our experiments.

a. IMS1 Configuration: The IMS1 dataset is a collection of CPU workload data obtained through stressing an IMS virtualized server with call setups. Table 3.1 describes the number of calls generated in a given time length, for the first cycle.

Table 3.1 IMS1, Cycle 1 Load Profile

Phase	Starting CPS	Variation	Duration
1	150	+50 CPS/10 s.	50 s.
2	400	-	100 s.
3	600	-	300 s.
4	200	-50 CPS/50 s.	150 s.

The next 5 cycles are variations of the first cycle as shown in Table 3.2, with different Calls per second (CPS) rates and durations.

Table 3.2 IMS1, Load Profile Variations, Cycles 2-6

Cycle	Variation from 1st cycle
2	+50 CPS
3	-50 CPS
4	-100 CPS
5	-25 CPS
6	+300 CPS

b. IMS2 Configuration: The IMS2 dataset is another collection of CPU workload data obtained through stressing an IMS virtualized server with call setups. Table 3.3 describes the number of calls generated in a given time length, for the first cycle.

Table 3.3 IMS2, Cycle 1 Load Profile

Phase	Starting CPS	Variation	Duration
1	150	+50 CPS/50 s.	150 s.
2	500	+50 CPS/50 s.	100 s.
3	900	+50 CPS/50 s.	150 s.

The next 5 cycles are the variations of the first cycle as shown in Table 3.4, with different CPS rates and durations.

Table 3.4 IMS2, Load Profile Variations, Cycles 2-6

Cycle	Variation from 1st cycle
2	-100 CPS
3	+275 CPS
4	-25 CPS
5	+400 CPS
6	+25 CPS

c. Web App Configuration: CPU load collected every 30 minutes from a virtual machine hosting a Web server.

d. Google Configuration: CPU load of a single machine in a cluster captured and made available by Google.

3.9.3 Results

The aim of our set of experiments is to evaluate the efficiency of the proposed workload periodicity detection approach in terms of the following metrics: 1) efficiency of the automated parameter selection process, 2) performance comparison with the auto-regression approach, and 3) ability to accurately detect lengths, shapes and amplitudes of cyclic patterns.

3.9.3.1 Automated Parameter Selection Process

The automated parameter selection process is performed in the evaluation phase of the algorithm to obtain the x and y parameters for prefix transreversal and digital print extraction. They are useful as well to get the optimal evaluation patterns for the decision phase. Table 3.5 shows the different parameters obtained out of the four considered datasets.

By analyzing Table 3.5 and comparing the results with the observed CPU workload from each data set as shown in Figure 3.2, Figure 3.3, Figure 3.4 and Figure 3.5, we observe that the x and y parameters vary from a data set to another. Specifically, we notice that these parameters have higher values for IMS1 and IMS2, as they reach higher CPU rates, have sharp CPU deviations and, also, high variance. For IT CPU workloads such as the case in Web App and Google, on the other hand, we observe that the x and y parameters have lower values. Lower CPU rates achieved as well as lower variance in the workload data explain these results.

Table 3.5 Automated Parameter Selection

Dataset	x	y	Grid size	Max step	Positive Eval. Pattern	Negative Eval. Pattern
IMS 1	8	14	64	31	[0,0,0,1,1,1]	[0,0,0,-1,-1]
IMS 2	9	14	32	15	[0,0,0,1,1,1]	[0,0,0,-1,-1]
Web App	7	12	32	15	[0,0,0,1,1,1]	[0,0,0,-1,-1]
Google	7	11	16	7	[1,1]	[-1,-1]

We can also observe that the “positive” evaluation patterns and “negative” evaluation patterns selected for three of the four datasets are identical. This is explained by the large amount of values in each of these datasets and the relatively low deviation of the digital print patterns necessary to detect positive and negative pattern deviations. Google stands out as the only data set with 2-value strings for its evaluation patterns. This is explained by its low amount of workload data and sharp deviations in its digital print values.

3.9.3.2 Workload Periodicity Detection vs. Autocorrelation

The next experiment aims to compare the efficiency of the workload periodicity detection approach with the auto-regression approach, as shown in Table 3.6. Here, we observe that all periodic patterns are detected by the workload periodicity detection algorithm. It is interesting to note that our algorithm detected 8.5 cycles in the Google data set (8 full cycles and one half-cycle of negative deviation), due to the sensibility of the algorithm to deviations in the digital print and to the cycles generated from the continuous mean splines of the evaluation phase. By observing the auto-correlation figures of each data set (see sample auto-correlation functions at Figure 3.2, Figure 3.3, Figure 3.4 and Figure 3.5), we notice that this method is less effective in detecting cycles with periodicity of different lengths. Auto-correlation could detect all the cycles for IMS1 and some of the cycles for IMS2 and Web App, but none of the cycles for Google.

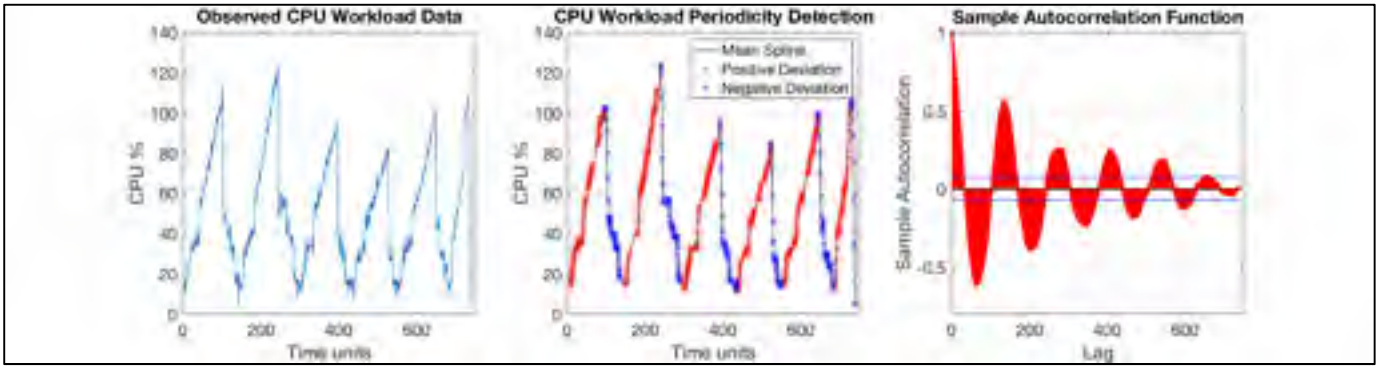

Figure 3.2 IMS1: Observed Data, Periodicity Detection and Autocorrelation Function

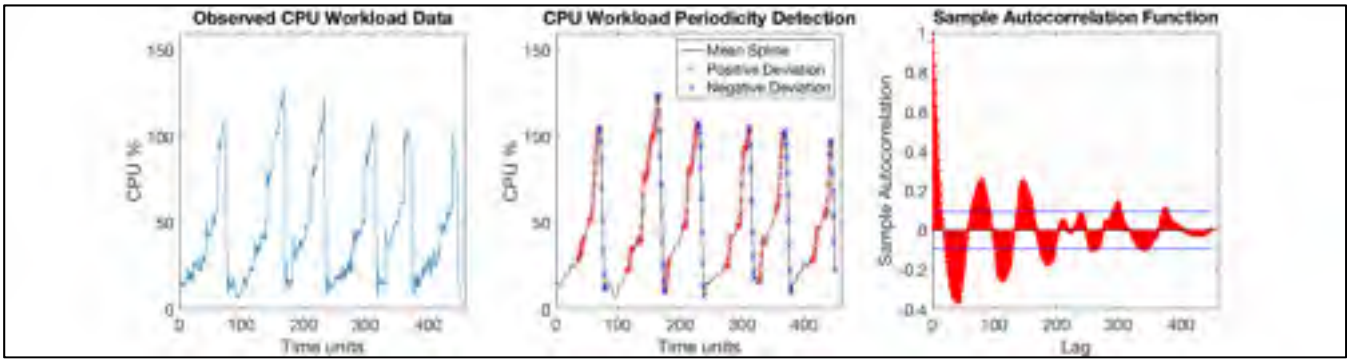

Figure 3.3 IMS2: Observed Data, Periodicity Detection and Autocorrelation Function

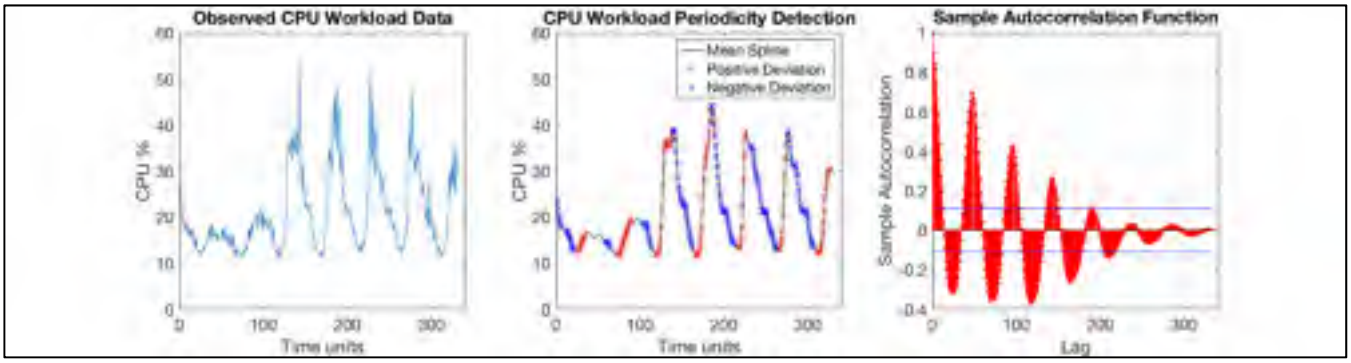

Figure 3.4 Web App: Observed Data, Periodicity Detection and Autocorrelation Function

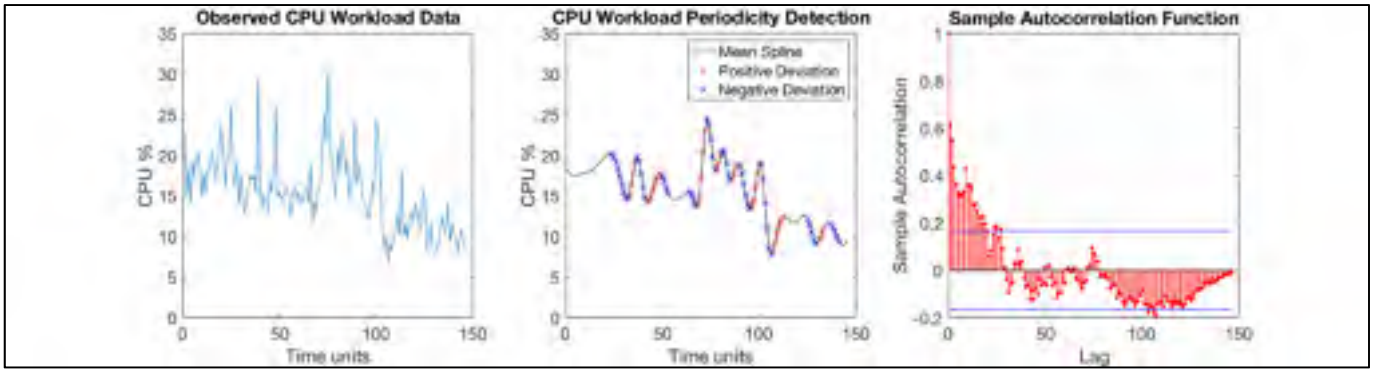

Figure 3.5 Google: Observed Data, Periodicity Detection and Autocorrelation Function

Table 3.6 Periodicity Detection Comparison

Dataset	Periodicity Detection Algorithm	Auto correlation	Improvement PDA vs Auto correlation
IMS 1	6	6	0%
IMS 2	**6**	5	20%
Web App	**7**	4	75%
Google	**8,5**	0	100%

3.9.3.3 Length, Shape and Amplitude

Table 3.7, Table 3.8, Table 3.9 and Table 3.10 show the length, starting and ending positions of each cycle, for each dataset. By observing these results, we find that the workload periodicity detection algorithm returns accurate estimations of each cycle's length. We also notice that these results go in line with the evaluation pattern estimations of Table 3.5, since the cycles found in Google have low lengths, explaining the low string size of the evaluation patterns and the sensibility required to detect periodicity.

Results show that the current implementation of the workload periodicity detection approach permits detection of general positive and negative deviations.

Table 3.7 IMS1 - Cycle length, starting point and ending point

Cycle	Length	Start	End
1	129	6	135
2	142	147	289
3	135	300	435
4	113	435	548
5	126	555	681
6	48	686	734

Table 3.8 IMS2 - Cycle length, starting point and ending point

Cycle	Length	Start	End
1	42	37	79
2	62	113	175
3	63	175	238
4	45	274	319
5	53	325	378
6	35	412	447

Table 3.9 Web App - Cycle length, starting point and ending point

Cycle	Length	Start	End
0,5	21	-	21
1	37	27	64
2	39	74	113
3	43	119	162
4	45	166	211
5	44	217	261
6	42	263	305
6,5	20	311	-

Table 3.10 Google - Cycle length, starting point and ending point

Cycle	Length	Start	End
0,5	32	-	32
1	8	33	41
2	24	43	67
3	9	68	77
4	7	78	85
5	9	86	95
6	11	95	106
7	22	107	129
8	10	131	141

To summarize, these experiments show that the workload periodicity detection approach is very efficient in finding generic periodic cycles in a large set of use cases and shows better performance compared to the auto-regression approach. Moreover, this approach offers a fully automated parameter selection, thus fitting patterns of any shape, amplitude and length.

3.10 Conclusion and Future Work

In this paper, we proposed a workload periodicity detection approach for cloud computing systems. The main advantages of the proposed approach lie in its (1) generic nature which enables its adoption for any type of discrete time-series datasets, and (2) aptitude to detect workload periodicity cycles when the lengths, amplitude, and shapes of the data patterns change. The proposed solution is based on the prefix transreversal technique which has shown its effectiveness in deducing evolutionary relationships among genomes in the field of molecular biology. Experiments carried out on IT and Telecom datasets reveal that our solution can maximize the accuracy of detecting workload periodicity cycles even in extreme cases wherein the lengths, shapes, and amplitude of the data patterns continuously vary, compared to the auto-correlation technique. Such promising results open the door for a

valuable track to examine the proposed workload periodicity detection approach on other workload attributes such as RAM and network traffic. We also plan to extend this work to online periodicity detection by using machine learning techniques and add a training phase.

3.11 Acknowledgment

This work has been supported in part by Natural Science and Engineering Research Council of Canada (NSERC), in part by Ericsson Canada and in part by Rogers Communication Canada.

CHAPITRE 4

DISCUSSION DES RÉSULTATS

Au fil des articles présentés dans les chapitres précédents, nous avons proposé deux approches innovantes, pertinentes dans le domaine infonuagique, en voie d'être brevetées et avec un potentiel intéressant de commercialisation. Mais qu'en est-il de leur interdépendance? Ce sujet n'est évidemment pas mentionné dans les pages précédentes, mais nous devons soulever ici les points reliant ces deux mécanismes, qui, dans les faits, ne forment qu'un seul outil.

D'abord, le mécanisme de détection de périodicité est un processus faisant partie intégrante du mécanisme de modélisation. En réalité, l'une des splines générées par le mécanisme de modélisation (la spline continue à intervalles fixes) est également utilisée par le mécanisme de détection de périodicité. Les valeurs re-discrétisées $\mu(t)$ ainsi générées sont donc utilisées pour chacune des deux approches. Cette solution fût privilégiée d'entrée de jeu lors de la réalisation de notre deuxième axe de travail, vues la précision des valeurs moyennes $\mu(t)$ générées avec ce modèle en rapport avec les données observées et la direction que nous désirions donner à notre modèle de détection de périodicité. La réutilisation du modèle basé sur des splines continues à intervalles fixes signifie donc que globalement, cette étape a un coût nul en termes de performance au niveau du mécanisme de détection de périodicité.

Finalement, il est nécessaire de mentionner que le mécanisme de détection de périodicité permet la classification de comportements de charges de trafic provenant d'un jeu de données. Cela signifie donc que les modèles générés par notre mécanisme de modélisation sont basés sur les profils de charge identifiés en amont. Cela permet donc de générer des modèles basés non pas sur une séquence de données sur une échelle de temps de 1 à n, mais bien sur des comportements de charges de trafic possédant des caractéristiques d'amplitude, de forme et/ou de longueur similaires. Cela s'avère une propriété très intéressante de la combinaison de ces deux approches, car c'est cette dernière qui offrira la possibilité de

détecter, par exemple, des anomalies dans le comportement de charge de trafic grâce à des outils de prévision et d'estimation de charges de trafic.

CONCLUSION

Dans ce document, nous avons proposé un outil de génération de charges de trafic dans des environnements de communication virtualisés. Cet outil est composé de deux approches innovantes, chacune ayant mené à deux applications de brevets provisoires et deux articles de revues. La première approche, qui est un mécanisme de modélisation de trafic de charge basé sur un processus hybride Hull-White-GA, a démontré une précision supérieure à d'autres solutions de modélisation existantes ainsi que des temps d'exécutions intéressantes pour un algorithme évoluant avec des jeux de données statiques. La seconde approche, qui est un mécanisme de détection de périodicité basé sur des opérations de transréversion par suffixe et sur l'analyse « d'empreintes numériques », s'avère une méthode prometteuse pour détecter et classifier des comportements de charges de trafic ayant des caractéristiques d'amplitude, de forme et de longueur similaires. Cette dernière a surclassé la méthode conventionnelle basée sur l'autocorrélation pour détecter des cycles, plus particulièrement dans des situations où ces cycles ont des variations significatives dans certaines de leurs caractéristiques. La combinaison de ces deux mécanismes ainsi proposés répond aux objectifs fixés, soient d'offrir un outil (1) générique de modélisation de trafic de charge, capable de modéliser n'importe quel type de ressource, (2) se concentrant sur le comportement des ressources systèmes, (3) comportant un mécanisme de détection de périodicité, (4) étant entièrement automatisé et (5) permettant de générer des charges de trafic synthétiques fidèles aux charges de trafic réelles. L'outil résultant offrira donc un moyen efficace d'analyse et de gestion de ressources, permettant aux administrateurs de systèmes virtualisés d'ajuster de manière proactive les paramètres de redimensionnement et de migration des machines virtuelles, pour ainsi respecter les politiques de SLA et QoS imposées par une entreprise.

Les améliorations recommandées à notre outil seraient d'en faire l'adaptation pour parvenir à faire la prédiction de charges de trafic ainsi que la détection d'anomalies en temps réel dans des environnements virtualisés de production. Cela permettrait d'ajuster de manière dynamique les paramètres de redimensionnement et de migration de ressources tout en respectant les politiques mises en place. Cette tâche nécessiterait toutefois de porter une

attention particulière sur la complexité de l'algorithme et le temps d'exécution, puisque la latence a un impact significatif sur la prise de décision dans de tels environnements.

BIBLIOGRAPHIE

M. Ahdesmäki, H. Lähdesmäki, R. Pearson, H. Huttunen, and O. Yli-Harja, "Robust detection of periodic time series measured from biological systems," *BMC Bioinformatics*, vol. 6, 2005.

C. An, J. Zhou, S. Liu, and K. Geihs, "A multi-tenant hierarchical modeling for cloud computing workload," *Intell. Autom. Soft Comput.*, vol. 22, no. 4, pp. 579–586, 2016.

A. Bahga and V. K. Madisetti, "Synthetic Workload Generation for Cloud Computing Applications," *J. Softw. Eng. Appl.*, vol. 4, no. 7, pp. 396–410, 2011.

S. Brunner and A. A. Ali, "Voice Over IP 101 - Understanding VoIP Networks," *Juniper Networks*, pp. 1–24, 2004.

L.-J. Cao and F. E. H. Tay, "Support vector machine with adaptive parameters in financial time series forecasting," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1506–1518, 2003.

A. K. Chanda, S. Saha, M. A. Nishi, M. Samiullah, and C. F. Ahmed, "An efficient approach to mine flexible periodic patterns in time series databases," *Eng. Appl. Artif. Intell.*, vol. 44, pp. 46–63, 2015.

A. K. Chanda, C. F. Ahmed, M. Samiullah, and C. K. Leung, "A new framework for mining weighted periodic patterns in time series databases," *Expert Syst. Appl.*, vol. 79, pp. 207–224, 2017.

Chen, X., Zheng, J., Fu, Z., Nan, P., Zhong, Y., Lonardi, S., & Jiang, T. (2005). Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *2*(4), 302–314. https://doi.org/10.1109/TCBB.2005.48

C. Cortes and V. Vapnik, "Support-Vector Cortes, C., & Vapnik, V. (1995). *Support-Vector Networks. Machine Learning*, 20(3), 273–297. http://doi.org/10.1023/A:1022627411411Networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

Z. Dias and U. Dias, "Sorting by Prefix Reversals and Prefix Transpositions," *Discret. Appl. Math.*, vol. 181, pp. 78–89, 2015.

A. K. Dutta, M. Hasan, and M. S. Rahman, "Prefix transpositions on binary and ternary strings," *Inf. Process. Lett.*, vol. 113, no. 8, pp. 265–270, 2013.

Z. Dziong, “ATM network resource management,” 1997.

P. Z. Dziong, “Modélisation, Estimation et Contrôle, pour les Réseaux de Télécommunication,” pp. 1–67, 2010.

M. G. Elfeky, W. G. Aref, and A. K. Elmagarmid, “Periodicity detection in time series databases,” *IEEE Trans. Knowl. Data Eng.*, vol. 17, no. 7, pp. 875–887, 2005.

D. Feitelson, *Workload modeling for computer systems performance evaluation*, vol. 6. 2014.

Hasan, M., Rahman, A., Rahman, M. K., Rahman, M. S., Sharmin, M., & Yeasmin, R. (2015). Pancake flipping and sorting permutations. *Journal of Discrete Algorithms*, *33*, 139–149. https://doi.org/10.1016/j.jda.2015.03.007

R. Hu, J. Jiang, G. Liu, and L. Wang, “Efficient resources provisioning based on load forecasting in cloud,” *Sci. World J.*, vol. 2014, 2014.

R. Hu, J. Jiang, G. Liu, and L. Wang, “Cpu load prediction using support vector regression and kalman smoother for cloud,” in *Distributed Computing Systems Workshops* (ICDCSW), 2013 IEEE 33rd International Conference on. IEEE, 2013, pp. 88–92.

S. Islam, J. Keung, K. Lee, and A. Liu, “Empirical prediction models for adaptive resource provisioning in the cloud,” *Futur. Gener. Comput. Syst.*, vol. 28, no. 1, pp. 155–162, 2012.

W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, “Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: Towards performance modeling,” *Futur. Gener. Comput. Syst.*, vol. 29, no. 5, pp. 1254–1264, 2013.

D. Magalhães, R. N. Calheiros, R. Buyya, and D. G. Gomes, “Workload modeling for resource usage analysis and simulation in cloud computing,” *Comput. Electr. Eng.*, vol. 47, pp. 69–81, 2015.

D. A. Menasce, V. A. Almeida, L. W. Dowdy, and L. Dowdy, “Performance by design: computer capacity planning by example,” *Prentice Hall Professional*, 2004.

I. Minei, “MPLS DiffServ-aware Traffic Engineering,” 2000.

I. S. Moreno, P. Garraghan, P. Townend, and J. Xu, “An approach for characterizing workloads in google cloud to derive realistic resource utilization models,” in *Proceedings - 2013 IEEE 7th International Symposium on Service-Oriented System Engineering, SOSE 2013*, 2013, pp. 49–60.

M. A. Nishi, C. F. Ahmed, M. Samiullah, and B. S. Jeong, “Effective periodic pattern mining in time series databases,” *Expert Syst. Appl.*, vol. 40, no. 8, pp. 3015–3027, 2013.

M. Khaledur Rahman and M. Sohel Rahman, “Prefix and suffix transreversals on binary and ternary strings,” *J. Discret. Algorithms*, vol. 33, pp. 160–170, 2015.

F. Rasheed and R. Alhajj, “STNR: A suffix tree based noise resilient algorithm for periodicity detection in time series databases,” *Appl. Intell.*, vol. 32, no. 3, pp. 267–278, 2010.

F. Rasheed and R. Alhajj, “A framework for periodic outlier pattern detection in time-series sequences,” *IEEE Trans. Cybern.*, vol. 44, no. 5, pp. 569–582, 2014.

“RBF SVM parameters,” http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html, accessed: 2017-10-16.

S. B. Shaw and A. K. Singh, “Use of proactive and reactive hotspot detection technique to reduce the number of virtual machine migration and energy consumption in cloud data center,” *Comput. Electr. Eng.*, vol. 47, pp. 241–254, 2015.

Sharmin, M., Yeasmin, R., Hasan, M., Rahman, A., & Rahman, M. S. (2010). Pancake flipping with two spatulas. *Electronic Notes in Discrete Mathematics*, *36*(C), 231–238. https://doi.org/10.1016/j.endm.2010.05.030

R. Tahmasbi and S. M. Hashemi, “Modeling and forecasting the urban volume using stochastic differential equations,” *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 1, pp. 250–259, 2014.

W. Wang, X. Huang, X. Qin, W. Zhang, J. Wei, and H. Zhong, “Application-level cpu consumption estimation: Towards performance isolation of multi-tenancy web applications,” in *Cloud computing (cloud)*, 2012 ieee 5th international conference on. IEEE, 2012, pp. 439–446.

T. Wang, J. Wei, W. Zhang, H. Zhong, and T. Huang, “Workload-aware anomaly detection for web applications,” *J. Syst. Softw.*, vol. 89, no. 1, pp. 19–32, 2014.

Z. Wei, T. Tao, D. ZhuoShu, and E. Zio, “A dynamic particle filter-support vector regression method for reliability prediction,” *Reliability Engineering & System Safety*, vol. 119, pp. 109–116, 2013.

H. Yang, Z. Luan, W. Li, and D. Qian, “MapReduce workload modeling with statistical approach,” *J. Grid Comput.*, vol. 10, no. 2, pp. 279–310, 2012.

D.-J. Zhang-Jian, C.-N. Lee, and R.-H. Hwang, “An energy-saving algorithm for cloud resource management using a kalman filter”, *International Journal of Communication Systems*, vol. 27, no. 12, pp. 4078–4091, 2014.