Proactive Resource Management to Ensure Predictable End-to-End Performance for Cloud Applications

---

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment
of the requirements for the degree

Doctor of Philosophy

by

In Kee Kim

May 2018

APPROVAL SHEET

This Dissertation
is submitted in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

Author Signature: _____

This Dissertation has been read and approved by the examining committee:

Advisor: Marty Humphrey

Committee Member: Alfred C. Weaver

Committee Member: Yanjun Qi

Committee Member: Hongning Wang

Committee Member: Byungkyu Brian Park

Committee Member: _____

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

May 2018

Proactive Resource Management to Ensure Predictable End-to-End Performance for Cloud Applications

---

A Dissertation

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Doctor of Philosophy

by

In Kee Kim

May

2018

APPROVAL SHEET

The dissertation

in submitted in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

In Kee Kim

AUTHOR

The dissertation has been read and approved by the examining committee:

Marty Humphrey (Advisor)

Alfred C. Weaver (Committee Chair)

Yanjun Qi

Hongning Wang

Byungkyu Brian Park

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, Dean, School of Engineering and Applied Science

May
2018

Abstract

Public IaaS clouds have become an essential infrastructure for enterprises and research organizations to run applications/services because of attractive capabilities from the public clouds. i.e., scalability, elasticity, resource diversity, and cost efficiency. Predictive resource management systems are developed to fully leverage such cloud infrastructures with two interrelated goals: maximizing SLA (Service Level Agreement) satisfaction and minimizing execution cost. However, existing predictive approaches are not sufficient to meet these two goals due to two uncertainties in public IaaS clouds – *workload uncertainty* and *performance uncertainty* –, and often show insufficient performance and adaptability in predicting future workloads and guaranteeing performance SLA. As a result, existing methods incur frequent SLA violations and require high execution cost.

This dissertation is to address such problems to achieve proactive resource management that assures end-to-end performance of cloud applications on public IaaS clouds. This research includes three important mechanisms for this direction.

The first mechanism is CloudInsight that addresses the workload uncertainty. CloudInsight is a novel prediction framework for forecasting real-world cloud workloads, leveraging the combined power of multiple workload predictors. More specifically, CloudInsight predicts future workload changes by creating an ensemble model with multiple workload predictors. The weights of the predictors are determined at runtime based on their accuracy for current workload using multi-class regression. We evaluated CloudInsight with various workload traces from real-world cloud applications. The results show that CloudInsight has 13% – 27% better accuracy than state-of-the-art predictors for all traces. Moreover, the results from a trace-based simulation with a representative resource management module show that CloudInsight has 15% – 20% less under-/over-provisioning periods, resulting in better cost efficiency and lower SLA violations than existing predictors.

The second mechanism is Orchestra that handles the performance uncertainty. Orchestra is a cloud-specific framework for controlling multiple cloud applications in the user space, aiming at meeting corresponding SLAs. Orchestra takes an online approach with lightweight monitoring and creates performance models for multiple cloud applications on the fly. It then optimizes the allocations of shared resources (e.g., CPU, memory, IO, network) and controls the resources to satisfy SLAs. We evaluated the performance of Orchestra on a production cloud (Amazon EC2) with a

diverse range of SLA requirements. The results show that Orchestra guarantees the performance of latency-sensitive/user-facing cloud applications (e.g., Web, DBMS) to meet the SLA requirements at all times. Moreover, we measured the accuracy of performance models in Orchestra framework, and the results often show less than 10% errors in estimating the performance of cloud applications.

In addition to the mechanisms that solve two main uncertainties in the public clouds, we present a new cloud simulator – PICS – that supports large-scale performance evaluation of cloud applications and resource management systems in a short amount of time. PICS enables the cloud end-users to evaluate the cost and performance of public IaaS clouds along with such dimensions like VM and storage service, resource scaling, job scheduling, and diverse workload patterns. We extensively validated PICS by comparing its results with the data acquired from real public IaaS cloud using real cloud-applications. We show that PICS provides highly accurate simulation results (less than 5% average simulation errors) under a variety of use cases. Furthermore, we evaluated PICS' sensitivity with imprecise simulation parameters. The results show that PICS still provides very reliable simulation results with inaccurate simulation parameters and performance uncertainty.

Acknowledgements

The six years at the University of Virginia were the best time of my life, but it was not so easy. I could not finish this long journey without support, advice, devotion, and sacrifice of many people. I would like to express my gratitude to them in this section.

First of all, my sincere gratitude goes to my adviser, Marty Humphrey. Without his guidance and help, I would never be able to finish my dissertation. I am fortunate to have him as my advisor. He gradually trained and taught me with insightful ideas and questions, rather than just providing answers. He was patient, always stood with me, and encouraged me. Working with Marty was very enjoyable and wonderful learning experience, and it made me an independent researcher. Marty is not only a research advisor but also a great mentor for my life. Now he is the role-model for rest of my life in academia.

I would like to thank Yanjun Qi for her help and contribution to this dissertation. This dissertation started with a small idea from her Advanced Machine Learning class. I am very lucky to have a long collaboration with her. She helped me and encouraged me to come up with new ideas for research problems I had.

I would like to thank Alfred C. Weaver for serving as the dissertation committee chair. I really enjoyed numerous hallway talks with Alf on the fifth floor at Rice Hall. Chatting with him was an excellent refreshment, and he gave me great and constructive advice for my research, life, career development.

I would like to thank Hongning Wang for being my committee. He gave me sharp questions and comments on my dissertation. I learned a lot from his respectful personality, research, and teaching. It was my honor to be his first TA at UVA.

I would like to thank Byungkyu Brian Park for being my dissertation committee. He gave me invaluable suggestions, supports, and questions when I had trouble to develop ideas for this dissertation project. Also, I respect his devotion to Korean Scientist Community at UVA.

I would express special thanks to Kevin Skardon, the chairperson of Computer Science department. Kevin is the best department chair and leader I have ever met, and his strong leadership keeps developing this department. He gave me great advice when I made the most difficult decision for my future career. Also, he was the committee chair for my qualifying defense. Without his support and encouragement, I would not complete my degree program.

It was my honor to contribute this department as TA for many semesters. Sometimes I complained it, but TA was a wonderful experience in understanding students from different cultures and learning how efficiently teach them. Also, I would like to thank my UVA teachers for giving me such opportunities. Thank you, Marty Humphrey, Hongning Wang, Mark Floryan, Aaron Bloomfield, Ahmed Ibrahim, Mark Sherriff, and David Edwards.

I would like to mention Wei Wang for his truthful friendship. Wei and I had shared the same office for years, and he is my best friend and mentor. I remember many days and nights we discussed and exchanged ideas about our research and life. We both have the same difficulty as an old graduate student with family and were struggling for the work and life balance. He was my comrade who fights with the same research problems as well. I hope he has a very successful career in academia.

I would like to say thank you to Jinho Hwang at IBM Research. I met him during my internship at IBM T.J. Watson Research Center in 2016 summer. We became a good friend and started a collaboration for Orchestra project. He is progressive, hard-working and brilliant.

I would also like to thank Sai Zeng at IBM Research. She was my mentor during my internship and gave me a very interesting and challenging research problem in enterprise cloud computing. Working with her was very pleasant because she is insightful, smart and goal-oriented.

For the last six years at UVA CS, I am so grateful to have many friends who are humorous, elegant, smart, and hard-working. They are Avinash, Ritambhara, Dezhi, Chong, Jack, Lin, Anindya, Sarah, Xinfei, Sourav, Nathan, Vanamala, Swaroopa, Abu Mondol, Asif, Jake, Yann, Samyukta, Akshay, Chi, Mohsin, Nipun, Ifat, Yong, Haoran, David, Rohan, Essex, Chunkun, and so many friends that I cannot list them all. Thank you all for your friendship, support, and peer pressure.

Also, I would like to thank my Korean friends here in Charlottesville. Thank you, Min-Kyu's family, Gwansik's family, Chaeyoon's family, Seungwook's family, Cholkyu's family, Changbum's family, Minjae's family, Taeyoung, Sung-In, Austin, Francis, and Hyewon Jin.

My friends in Korea should be mentioned here. They always encouraged me by Facebook and Kakao talk messages. Thank you, Ji Seong, Seokju, Boknam, Seongsoo, Kirk, Seungchan, Ji-ae, Sungho, Insik, Hyunghoon, James Song, Jungmin, and many others. And, I would like to thank my former advisor, Jong Sik Lee, for his help when I decided to start the doctorate program in the U.S.

I would never be able to accomplish anything in my life without endless support from my family and family-in-law. Thank you, my parents – Wan Young Kim and Jongrye Park –, for their love, understanding, support, and advice during this long journey. I would also like to thank my older sister's family – Hyejin Kim, Yongjae Kim, Chris –, and my younger sister's family – Haejoo, Wonwook, Eden –, for their endless love and encouragement.

My family-in-law always stood with me. Their love and support are always in my heart. Thank you, my parent-in-law – Duck-ki Moon and Youngsook Park –, for their love and encouragement. Especially, my father-in-law is now in heaven though, I am sure that he will be very pleased with this achievement. I would also like to thank sister-in-law – Dr. Yuri –, and brother-in-law's family – Jongil, Jeongmi, Jiwoo –, for their support, encouragement, and truthful advice.

Last but not least, my deepest gratitude goes to my wife Jihye and my daughter Yubin. 6 years ago, I made a crazy decision that I quit my job and started a doctorate degree in the U.S. I am still wondering. If I could go back to 2012, would I make the same decision? Jihye had never complained about this decision although she already knew our life would be in trouble. She just gave up her stable life in Korea and came to the U.S. with me and two years old baby. Our life in Charlottesville was tough and nothing was easy. She has endured all the difficulties. Thank you for her love, patience, and understanding during this long journey. Yubin is the most delightful part of my family. Yubin has cheered me up to finish this dissertation and found several typos in my papers. Without Jihye and Yubin, this achievement is meaningless.

To Yubin and Jihye

Contents

Chapter 1

Introduction

For many organizations today, the issue is not *whether to use* public IaaS (Infrastructure as a Service) clouds[1] [5, 67, 144, 167], but rather *how best to use* public IaaS clouds [171]. Public IaaS clouds offer a diverse type of virtualized computing and storage resources, cost efficient pay-as-you-go pricing models, and higher availability, scalability and elasticity of resources than other traditional computing platforms [16, 81]. These advantages make public IaaS clouds as the primary computing infrastructure for hosting diverse cloud applications[2] and services[3] in industry, research and academic institutions.

To effectively leverage public IaaS clouds, application service providers employ resource management systems that control virtualized cloud resources to handle dynamic changes of workloads. i.e, job[4] and request arrivals. Figure 1.1 illustrates a cloud application operation model with a resource management system. Three entities are involved in this model; 1) cloud end-users, 2) public IaaS clouds, and 3) a resource management system.

Cloud end-users generate workloads or resource demands to target cloud applications and include diverse groups with their interests in cloud applications. i.e., users for big data analytics, web services, scientific application, etc. Public IaaS clouds are the infrastructure that provides on-demand cloud resources such as VMs (Virtual Machines), containers, storage, network, and others. These on-demand resources host cloud applications that process jobs from the cloud end-users. Resource management

[1]The public IaaS clouds includes Amazon Web Services [5], Microsoft Azure [144], Rackspace [167], Google Compute Engine [67], etc.

[2]These cloud applications include business, scientific and big data analytics applications.

[3]The cloud services include web search (e.g. Microsoft Bing), social networks (e.g. Facebook), e-commerce services (e.g. eBay), and on-demand video streaming services (e.g. Netflix).

[4]In this dissertation, we use the term "job" and "request", *interchangeably.*

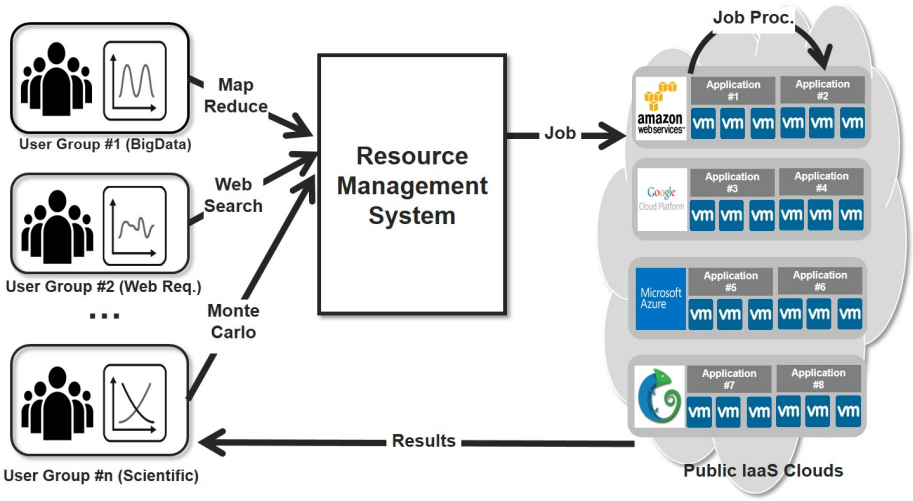


Figure 1.1: **Application Operation Model in Public IaaS Clouds.**

system plays the most important role in this cloud application operation model. This system is responsible for finding proper cloud resources (in public IaaS clouds) to execute the jobs (from cloud end-users) with the objectives of satisfying SLA (Service Level Agreement) targets and minimizing execution cost [6, 15, 60, 64, 68, 94, 140]. The real-world examples of SLA targets are maintaining a response time/tail latency of user requests or completing job execution within a user-defined deadline [40, 94, 95, 114, 122, 186]. With these two goals, the resource management system adopts various mechanisms that automatically scales clouds resources for the cloud applications to process the jobs from cloud end-users.

1.1 Background: Cloud Computing

Cloud computing is the latest evolution of distributed computing technologies, and it delivers various computing resources – *servers, storage, network, databases, containers, even applications* – as a service over the Internet [16, 135]. A collective power with the massive amount of computing resources provided by "cloud computing and its underlying data center technologies" handles the resource needs of diverse (and globally deployed) applications from enterprise and research organizations.

Elasticity is a key characteristic of cloud computing and differentiates clouds from existing large-scale computing infrastructure like grid or cluster computing [75]. This characteristic allows cloud applications to dynamically adjust the amount/type of resources according to the fluctuation of workloads to them. For instance, cloud

applications provision the minimal amount of up-front servers when the number of requests to the applications is very low. And, the applications provision potentially unlimited amount of servers when the user requests are extremely high. According to a recent report [184], Netflix (hosted by Amazon Web Services) occupied approximately 37% of Internet traffic during peak hours in 2015.

Moreover, the elasticity of cloud computing reduces TCO (Total Cost of Ownership)[5] for applications and service infrastructure because application service providers and users are only responsible for paying cloud cost while the resources are provisioned. This is called "pay-as-you-go" pricing model [16]. According to a cost analysis report [146], migrating applications and infrastructures to a cloud provider can save nearly 40% of TCO for three years. Due to these advantages of cloud computing, many enterprise and research organizations can dramatically save the up-front investment and management cost for on-premise infrastructures.

1.1.1 Cloud Service Models

Cloud computing provides different service models to cloud users and applications service providers. These services can be classified into four different models such as IaaS (Infrastructure-as-a-Service), PaaS (Platform-as-a-Service), SaaS (Software-as-a-Service), and XaaS (Everything-as-a-Service). Figure 1.2 illustrates key differences of legacy on-premise model, IaaS, PaaS, and SaaS cloud models. Among three cloud models, IaaS requires substantial management efforts to users and SaaS requires the least efforts to the underlying infrastructures. In other words, IaaS cloud users have more freedom to manage and control the underlying infrastructure and SaaS users least access to these infrastructures.

Infrastructure-as-a-Service (IaaS): The IaaS model offers diverse virtualized and infrastructure resources as a form of services to cloud users. The infrastructure resources include VM (Virtual Machine), storage, network, and other on-demand resources. Common types of cloud resources provided by IaaS model is VM. Users and customers for IaaS model are responsible for managing software stacks for running the resources. i.e., OS, software installation. The IaaS model is the closest to traditional on-premise infrastructure. The providers for this model are Amazon Web Services [5], Microsoft Azure [144], Google Compute Cloud [67], and Rackspace [167]. Moreover,

[5]TCO includes CapEX (Capital Expenses), OpEX (Operating Expenses), and indirect business/management costs.

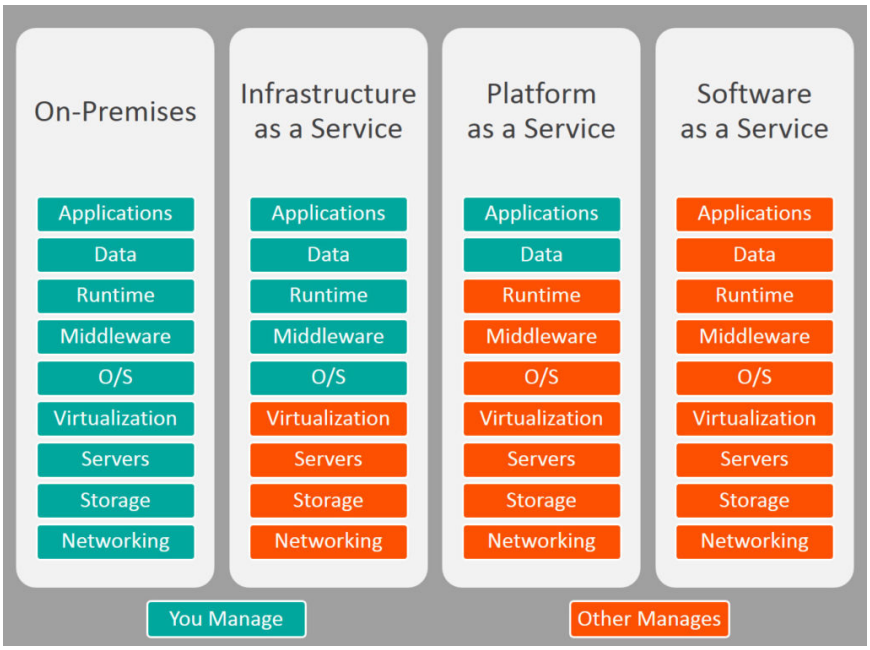
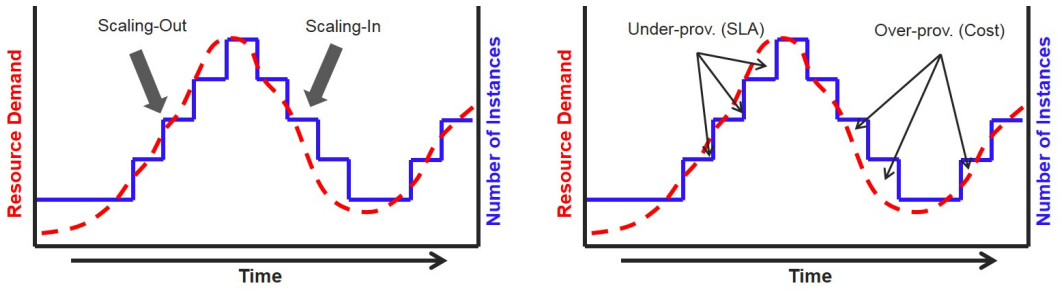


Figure 1.2: **The key differences between three cloud models (IaaS, PaaS, and SaaS) and on-premise model. Green boxes are users' responsibility, and orange boxes are cloud service providers' responsibility. Figure courtesy of BMC software [197].**

users, companies, and many other organizations can create IaaS infrastructure by using open-source softwares. OpenStack [158] and Apache CloudStack [14] are widely used to build IaaS infrastructure.

Platform-as-a-Service (PaaS): The PaaS model provides a platform service that allows users to develop, test, debug, and run applications without managing infrastructures and software stacks. The platforms commonly include OS, development/test environments, databases, storage, and middlewares for running applications. The most important characteristic of PaaS model is that users can develop and execute applications and have no needs for managing underlying infrastructure. However, because PaaS model is built on top of different infrastructures, it often limits the type of programming languages and software stacks (OS or databases). The providers of PaaS model are Google App Engine [65], IBM Bluemix[6] [84], and Heroku [79].

[6]As of October 2017, IBM Bluemix is changed to IBM Cloud [84].



(a) Autoscaling Operation (b) Under/Over-Provisioning

Figure 1.3: **Autoscaling Operation and Under/Over-Provisioning Problem. The dotted line (red) indicates the resource demands/number of requests from cloud end-users and the bold line (blue) means the number of cloud instances.**

Software-as-a-Service (SaaS): The SaaS model provides on-demand softwares over the Internet. Cloud users for the SaaS model have no requirement for software installation and development as well as infrastructure management. The SaaS model providers are Salesforce [175], Box [23], AWS EMR (Elastic MapReduce) [7], and others.

Everything-as-a-Service (XaaS): The XaaS models are a more specialized form of delivering platform or services over the Internet. Typical examples are Storage-as-a-Service, DB-as-a-Service, API-as-a-Service, Container-as-a-Service, Function-as-a-Service, and others. As the readers guess with the names, these services are more focused on specific delivery models and applications like databases, storages, containers, and even cloud functions. As clouds have become more dominant for IT industry and application delivery models, new XaaS model will appear.

In this subsection, we discussed four representative service models in cloud computing. While all these models are contributing to cloud evolution, this dissertation work is focused on application and resource management in IaaS clouds. The next subsection will discuss resource management problems in IaaS clouds.

1.2 Cloud Resource Management Problem

To fully utilize the elasticity of cloud computing, horizontal and vertical scalings have been proposed for the resource scaling and management operations for cloud applications. Horizontal scaling is a scale-in/out approach that changes the size/amount

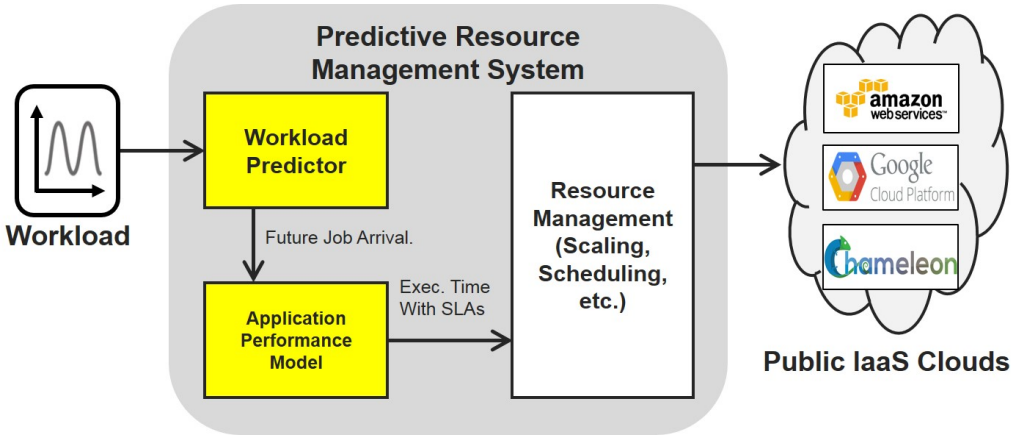


Figure 1.4: **Predictive Resource Management System.**

of cloud resources (e.g., VM) to handle fluctuation of workloads (job arrivals) [6, 68, 131, 132, 140]. Vertical scaling[7] is a scale-up/down approach that changes type of cloud resources with workload dynamics [15]. Between these two scaling approaches, horizontal scaling is more widely adopted [12, 35, 44, 50, 73, 78, 80]. Unfortunately, vertical scaling has narrowed applicability and can easily reach to the capacity limitation [173, 205, 211]. Vertical scaling thus is more likely used for limited resource management domains (e.g., DVFS [115], CPU [189] and memory scaling [185]) rather than general cloud application and resource management domains.

Due to the limitation of vertical scaling, autoscaling [6, 68, 140] is a *de facto* approach for horizontal scaling in cloud applications and is applied to various real-world cloud applications/services like Netflix's Fenzo [149] and Facebook [204]. Figure 1.3(a) illustrates a fundamental mechanism of this scaling approach. Autoscaling mechanisms and triggers monitor the utilization and behavior of current resources and adjust the size/amount of resources according to the fluctuation of workloads and user-defined rules (e.g., upper-/lower-bound of CPU usage). However, as shown in Figure 1.3(b), autoscaling can often be sub-optimal because of its *reactive* nature [150]. The *reactive* nature often results in over- and under-provisioning of cloud resources that causes low cost efficiency and high SLA (violations. To overcome such limitations, many predictive resource management systems have been proposed [21, 28, 31, 47, 60, 61, 64, 92, 154, 161, 162, 174, 181, 203, 207].

[7]Vertical scaling replaces a low-performance instance (eg., an instance with less CPU cores) with a high-performance instance (e.g., an instance with more CPU cores) when workload increases. Vertical scaling also changes an instance type from a high-performance VM to a low-performance VM when workload decreases.

Figure 1.4 depicts a typical predictive resource management system. The predictive resource management systems commonly consist of three modules; a *workload predictor*, *application performance model*, and *resource management module*. The workload predictor forecasts future job arrival time/rate from the cloud end-users. The application performance model estimates execution/processing time of the workloads. The last component – resource management module – allocates/deallocates cloud resources and maps user workloads to specific cloud resources. To achieve desired SLA satisfaction and cost efficiency of cloud applications, the decision and operation of the predictive resource management system are heavily relying on the results from two predictive modules. The workload predictor and application performance model provide crucial information to when or what to scale cloud resources. However, state-of-the-art approaches are still sufficient to achieve the desired performance for cost efficiency and SLA satisfaction due to deficient performance of these two predictive modules. The main causalities are *two uncertainties* in public IaaS clouds, namely *workload uncertainty* and *performance uncertainty*.

Therefore, the overall goal of this research is to mitigate such uncertainties and create new predictive resource management system for cloud application providers. This work first focuses on designing highly accurate workload predictor and application performance models, then combining them with a resource management module to assure predictable performance of cloud applications.

1.3 Research Challenges

To achieve the goal of predictive resource management, there are three research challenges. These three research challenges involve prediction of future workload patterns from real-world cloud applications, prediction of performance of cloud applications, as well as efficient evaluation of large-scale cloud applications. The following subsections describe the details of three challenges.

1.3.1 Challenge 1: Uncertainty in Future Workload Pattern

A widely used assumption, when developing/deploying cloud applications, is that cloud workload can be one of the following four patterns [58, 131, 193]. These four patterns are increasing/decreasing, on/off, cyclic bursty, and random (bursty) pattern. The increasing pattern is often observed for very successful start-up cloud applications. (the decreasing workload is opposite case.) The on/off pattern is modelled with

batch or scientific applications like Hadoop/MapReduce and MPI applications. The cyclic bursty represents cloud application scenarios of e-commerce and on-demand video-streaming services. i.e., Amazon, eBay, Netflix, Hulu. The random (bursty) can be shown in cloud applications when unexpected social events happen.

With the above assumption, cloud practitioners may determine the workload pattern to their applications. Although the workload pattern is known a priori, designing a workload predictor for predictive resource management is still difficult for many reasons. First, long-term observation and measurement efforts are required to correctly understand the behaviors of user workloads. Such observation and measurement steps are necessary for selecting a proper model and tuning parameters in the prediction model. Also, the user workload may contain a certain level of burstiness and randomness that can easily degrade the performance of the workload predictor [2, 88].

Furthermore, many large-scale cloud applications (e.g., SNS, web search, multimedia streaming, big data) are globally deployed and handle workloads coming from all around the world, so that their workload patterns are the mixture of individual patterns and can be continuously fluctuating over time [33, 34, 169, 170, 187, 200, 201]. Thus, the above assumption is no longer valid for such applications and the dynamics in cloud workloads makes this challenge even harder.

There are several approaches for predicting future workloads [28, 46, 76, 89, 125, 127, 154, 164, 181, 203]. However, these approaches are inaccurate, because they did not consider the dynamics and mixture of cloud workloads.

1.3.2 Challenge 2: Uncertainty in Cloud Application Performance

Performance uncertainty has become a *norm* for cloud applications in public IaaS clouds [62, 71, 85, 101, 120, 130, 160, 178]. This performance uncertainty can be caused by diverse factors. i.e., complex workloads to data centers, multi-tenancy on shared servers, and hardware heterogeneity in cloud data centers. In our previous work [104], it is observed that AWS (Amazon Web Services) has approximately 20% in performance uncertainty (variability) when a cloud (MapReduce) application executes the same jobs on the same type of VMs in the same data center. Due to the performance uncertainty, it is very challenging for a resource manager and cloud applications to satisfy SLA requirements from the cloud users.

Another causality of the performance uncertainty, which many cloud practition-

ers may neglect, is co-runners on the same VM instance. Due to the evolution of microservice and orchestration infrastructures, it is prevalent for VM instances to run multiple Docker containers [4, 51, 52, 69, 116, 137, 141]. The execution of these Docker container is not coordinated, and one or more Docker containers can suddenly consume the substantial portion of VM resources (e.g., CPU, memory, IO, network). Therefore, services/applications on other containers do not provide desired performance for satisfying their SLA requirements due to the lack of resources and performance interferences from the containers with high resource consumption.

There are several approaches to minimize this performance uncertainty to assure the performance of cloud applications. Unfortunately, these approaches rely on the host machine's system/HW-level statistics (e.g., program counter and cache miss rate) [43, 102, 148, 156, 208, 214, 216]. Therefore, it is not feasible for cloud service providers to apply such mechanisms to cloud resource/application management due to the unobservability problem [62].

1.3.3 Challenge 3: Difficulty in Large-scale Evaluation of Resource Management Systems and Cloud Applications

Cloud applications on public IaaS clouds and the resource management systems are typical large-scale distributed systems that should potentially handle the massive amount of jobs/requests from cloud end-users and diverse types/sizes of cloud resources. Unfortunately, evaluation approach to such large-scale cloud applications is still under-developed.

A typical approach for evaluating such large-scale cloud applications, taken by many organizations today, getting started is that a few cloud experts and application developers within the organization deploy a small-scale test cloud-application on the public cloud of choice. Then, the next steps are to scale-up the test cloud-application to better assess the capabilities and viability in the context of the organization's particular goals and requirements. However, this approach is time-consuming, too specific to one cloud, and hardly addressing longer-term issues.

Cloud simulators can be alternative to this approach and several cloud simulators [26, 111, 157] are proposed from the research community. These simulators are generally leveraged for testing diverse use-cases in data center's design/management and provide diverse capabilities for addressing infrastructure and data center issues. However, such simulators have little consideration for handling issues and concerns

of perspectives from the cloud service providers, so that these approaches are still insufficient to be used for evaluating large-scale cloud applications and the resource management.

To support the large-scale evaluation of cloud applications and resource management, the following capabilities are required for a new evaluation platform:

- Being able to answer performance, reliability, and cost concerns from application service providers.
- Being able to provide a trustworthy simulation/evaluation with performance and workload uncertainty in public IaaS clouds.
- Being able to simulate/evaluate of a broad range of resource management policies. i.e., horizontal/vertical scaling, job scheduling, and job failure policies.

1.4 General Approach

The dissertation presents a comprehensive solution that manages diverse cloud resources to assure end-to-end performance of cloud applications by addressing the challenges discussed in the previous section. More specifically, the approaches are:

1. A holistic evaluation of the performance of existing cloud workload predictors to determine the best workload predictors for well-known cloud workload patterns. This evaluation is performed with diverse performance metrics including model accuracy and overhead, cost efficiency and SLA satisfaction rate. Further evaluation is performed to determine the best predictive scaling styles (e.g., predictive scaling-out, predictive scaling-in, or both) that achieve best cost and performance benefits of cloud resource/application management.
2. A framework that predicts future variation for real-world cloud workloads, which have a variety of patterns and dynamic fluctuations. Due to the dynamic nature and variability of real-world cloud workloads, this framework employs a number of existing workload predictors and automatically ensembles the best predictors with their unique contribution. To determine the best predictors and the contribution, this framework employs a SVM (Support Vector Machine)-based multiclass regression mechanism.

3. A model that estimates the performance (e.g., response time, job execution time) of cloud applications with a set of resource utilization. Based on this model, a new application/resource control framework is developed to guarantee the performance SLA (e.g., response time SLA/constraint) of the cloud applications under the performance uncertainty. More specifically, this framework is designed to manage a cloud application's performance when it is running with other co-runners (e.g., different cloud applications, containers) on the same VM instances and is competing with them to consume shared resources.

4. A simulator that supports large-scale evaluation of cloud applications and resource management and is designed based on the perspectives from cloud application service providers. This simulator enables cloud practitioners and application designers to test diverse real-world use-cases with different resource management policies and cloud configurations as well as assess cloud cost and application performance in a short amount of time without actual applications' deployment. To provide trustworthy evaluation results, the simulation results are validated against the measurement results from the real public IaaS clouds.

1.5 Contributions

The most important contribution of this research is that we introduce an innovative approach for proactive resource management for assuring end-to-end performance of cloud applications on public IaaS clouds. To this end, we design and develop three essential components in the resource management system – *workload predictor*, *application performance model*, and *resource scaling and control model*, which address the workload and performance uncertainty with highly accurate prediction and resource control. More specifically, this research makes the following contributions.

- To determine the best workload predictor for different cloud workload patterns and best predictive scaling styles (chapter 3), we perform an extensive evaluation of 21 existing workload predictors in prediction accuracy for forecasting future workload patterns (e.g., job arrivals). Also, we conduct a comprehensive valuation of workload predictors regarding cost and deadline miss rates. This evaluation considers various workload patterns, three styles of scaling operations, and two different billing models. Finally, we determine the best workload predictor and the best style for predictive resource scaling operations.

- To address the workload uncertainty (chapter 4), we create a new workload prediction framework that is an online, multi-predictor based approach that performs highly accurate workload prediction with low overhead under dynamic cloud workloads with various patterns. We also introduce a novel online mechanism to create an ensemble workload predictor. This mechanism assigns weights to each predictor by accurately estimating that predictor's relative accuracy for the next time interval using multi-class regression.

- To address the performance uncertainty caused by co-runners (chapter 5), we create a cloud application control framework that employs multiple online prediction algorithms with multi-variant regression to proactively monitor the performance and execution of different cloud applications on the same cloud instance. This control framework leverages a user-space resource control framework that aims the fast resource adjustment to violated SLAs and a lightweight control loop to manage multiple applications/services with SLA constraints on cloud instances.

- To address the difficulty in large-scale evaluation of resource management systems and cloud applications (chapter 6), we create a new public IaaS cloud simulator that is versatile and satisfies cloud user's various needs of evaluating cloud-scale applications, resource management mechanisms as well as public IaaS clouds without actual cloud deployment. This is a first simulator that supports both horizontal and vertical cloud resource scaling, to the best of our knowledge. A thorough validation and analysis are performed against the actual measurement from the public IaaS clouds in order to confirm the correctness of the simulation results and the simulator's sensitivity/reliability to the performance uncertainty.

1.6 Thesis Statement

Proactive resource management to ensure the predictable end-to-end performance of public cloud IaaS applications can be achieved through improved cloud workload predictions and improved cloud application performance modeling.

1.7 Dissertation Organization

The rest of this dissertation is organized as follows: Chapter 2 discusses related work of this research. Chapter 3 presents the performance evaluation of existing workload forecasting techniques and mechanisms of cloud resource scaling. Chapter 4 presents CloudInsight framework for predicting diverse, real-world cloud workload patterns. Chapter 5 presents Orchestra framework to guarantee performance SLA for cloud application under performance uncertainty and interferences. Chapter 6 presents PICS simulation framework for evaluating resource management systems and cloud applications. Chapter 7 summarizes this research and provides directions for future exploration.

Chapter 2

Related Work

This chapter discusses related research work. First, we describe workload prediction techniques for cloud applications (Section 2.1). Then, we discuss research on building a performance model (Section 2.2) and addressing performance interference (Section 2.3) for cloud infrastructure and cloud application management. Lastly, this chapter describes evaluation platform focusing on cloud simulator (Section 2.4).

2.1 Cloud Workload Prediction

A large body of work has been conducted for developing workload predictor to address dynamic workload patterns in predictive cloud resource management. There are two major directions in workload predictor research. The first direction is to focus on predicting the future resource usages (e.g. CPU, memory, and I/O) based on past resource usage [18, 49, 64, 181, 206]. The second direction is to forecast next job arrivals to target cloud applications. This direction focuses on predicting the future job arrival rate/time by applying various workload predictors and is what this dissertation work is to address.

A wide variety of predictive models/approaches has been proposed to accurately forecast temporal and other properties (e.g., resource needs, data size) from user requests. Most notably, employing a *static* workload predictor is the main stream of such proposals. Various models from statistical and machine learning domains are used to design the workload predictor. These models are generally relying on regression, time-series, and machine learning [18, 46, 89, 153]. Among them, time-series approaches are the most popular approach (ES [21, 138, 154, 217], AR [30, 31, 49, 202], ARMA [55, 100, 161, 174, 190], ARIMA [28, 47, 213] and others [61,

113, 164]) because cyclic bursty is known as a typical workload pattern for cloud applications [1, 97]. Such time-series-based approaches are based on an assumption that cloud workloads are repeating over time. However, such static predictor based approaches are not sufficient to address the dynamics cloud workloads. In particular, a short-term burstiness [88] can easily degrade the accuracy/performance of such predictors, so that static approaches often show insufficient performance for unknown or dynamic workload patterns.

Furthermore, several custom predictive approaches are developed to address dynamic cloud workload patterns. PRESS [64] and CloudScale [181] employ a custom predictor that consolidates FFT (Fast Fourier Transform) and Markov model. FFT is used to detect a signature of workload patterns, and Markov model is to address a short-term change of the workloads. However, in practice, it is challenging for cloud users to determine the transition probability of the Markov model correctly. Wood et al. [203] developed a hybrid approach, combining robust linear stepwise regression and the model refinement. This technique requires an offline profiling for the initial linear model creation.

To overcome the limitations of above approach, multi-predictor approaches are also proposed. These methods are in the spirit similar with our work in chapter 4 because they are designed to provide more generic and adaptive nature to their target applications and resource management infrastructure. Khan et al. [103], Herbst et al. [76], and Liu et al. [125] proposed an adaptive model for workload prediction. These approaches employ a classification and clustering (e.g., decision tree) of incoming workloads and statically allocate the best predictor for the particular type of workloads to increase the performance of workload prediction. However, for real workloads without clear seasonality and trend, it is hard to enumerate all possible classes *a priori*. Therefore, these approaches are not general enough to handle unknown, varying and/or non-typically-patterned real workloads. Our work in chapter 4, on the other hand, makes no assumption about the class/type/patterns, and thus can handle these real workloads. Moreover, the extra step of classification and clustering adds additional overhead to each job arrival prediction.

ASAP [96] and Vadara [127] are two ensemble approaches with multiple workload predictors. However, these two approaches use a simple assumption to determine contributions from each individual predictor, i.e., recent the best predictors (e.g., the lowest cumulative error during the previous monitoring interval [60]) will perform the best for the near-future. However, we observe that this assumption is not always true.

Especially, the workloads with short-term burstiness [88] can break this assumption. Unlike these approaches, we plan to utilize a much longer history, and employ multi-class regression model to predict the future accuracy of local predictors. Therefore, our work aims at providing more robust weights and more accurate predictions.

2.2 Cloud Performance Model

Modeling and estimating performance of cloud application and infrastructure are critical factors for managing cloud resources. i.e capacity planning, resource allocation, job scheduling, cost optimization, and system anomaly detection. Existing works have tried to create performance models that predict various performance aspects (e.g. I/O, CPU, memory usage, disk usage) of target cloud systems by utilizing various techniques such as SVM [36, 119, 165, 207], regressions (e.g. linear, logistic, quantile) [139, 155, 196], queueing theory [24], and ANN [118, 119]. These models are also adopted to predict the performance of various cloud application. i.e. Big Data processing framework (Hadoop [41], Spark [212], Storm [191], Heron [117]), Web (n-tier), DBMS, even holistic VM performance. However existing works have the following limitations.

First, the existing works often require both detailed knowledge and huge profiling effort on cloud infrastructure [118, 119]. However, cloud users have very limited access to cloud infrastructure and host machines/servers (e.g. holistic workload to data center, I/O usage, hardware specification, hypervisor), it is infeasible to build a performance model, which requires infrastructure information details.

Second, these approaches are often too specific to certain applications [77, 94, 194, 215], so that these approaches do not reflect general cloud applications/infrastructure. Moreover, some other works are only concentrated on modeling small parts of execution steps/pipelines rather than modeling an end-to-end performance of whole execution step of application pipelines. These works may not be a useful indicator to cloud resource manager. Last, the existing works still not enough to address performance uncertainty in cloud applications and infrastructure due to lack of information on multi-tenancy [24, 36, 48, 118, 119, 139, 155, 165, 196, 207].

2.3 Cloud Performance Interference

2.3.1 Cloud/Infrastructure provider-centric approach

There are significant works from the research community to detect, prevent, mitigate, and manage the performance interference caused by multiple co-located tasks and applications on the same physical hardware.

Designing an intelligent QoS management framework: This direction to detect the performance interference and schedules multiple tasks to avoid SLA violation from foreground (FG) applications. Q-Cloud [148] is a QoS-aware scheduler that mitigates resource contention and performance interference. Q-Cloud predicts a SLA violation of a task by an online feedback controller with a discrete time MIMO (Multi-Input and Multi-Output) model and it allocates more resources if SLA violation is expected. DejaVu [192] is a framework to detects performance interference with performance index by comparison of the performance from a production VM and the sandbox execution. Once SLA violation from the interference is detected, DejaVu mitigates it by adding more resources. DeepDive [156] detects and manages performance interference with a warning system. If interference is detected, it performs VM cloning with workload duplication and migrates the most aggressive VMs to another physical machine. Dirigent [216] is a lightweight QoS control framework for co-running of multiple tasks. Dirigent is designed to support cloud-offloaded tasks from user devices. Its control policy is to improve the tail latency of SLA (e.g., 95% latency) rather than minimizing the execution time of FGs. Dirigent also considers improving throughputs of backgrounds (BGs) and co-running tasks/processes rather than employing simple management policies (stopping or killing the BGs).

Determining the safe task placement: Bubble-up [134] is a mechanism to find a safe colocation of FG and BG, running on the same physical machine. Bubble-Up performs an offline/static profiling of FG tasks to create a sensitivity curve, representing how FG's QoS degrades as resource contention is increased. Bubble-Flux [208] is a dynamic and online QoS management mechanism and addresses the limitations from off-line profiling-based approach (e.g., Bubble-Up). Dynamic Bubble is to create the current sensitivity curve for FGs by generating a short-term burst of memory-intensive workloads. Online Flux Engine is a feedback controller with PIPO (Phase-in/Phase-out) that manages BGs when the FG violates SLA. Paragon [43] is an interference/QoS-aware scheduler for data centers. Paragon relies on minimal

offline profiling for the new workload (e.g., 1 min. on two servers) and a collaborative filtering to place tasks on the particular hardware (HW) platform where incurs low interference of co-located tasks. CPI^2 [214] is a mechanism that suggests CPI (cycle-per-instruction) as a performance indicator to detect performance interference of victim FG tasks. If an antagonist (causing the performance interference) is identified, CPI^2 forcibly reduces CPU usage of it by CPU hard-capping. Heracles [126] is a feedback controller that supports safe collocation of multiple tasks relying on modern performance isolation mechanism like Intel's cache allocation and *qdisc*.

However, the approaches mentioned above have the following limitations and differences; they often require 1) an on/off-line profiling to model the performance interference, 2) HW information (e.g., CPI, cache miss rate.), and 3) an expensive VM cloning and sand-box executions. On the other hand, our work in chapter 5 does not perform/require such profiling; rather our approach employs an online model to determine the optimal resource allocation to both FG and BGs on the same VM. Moreover, we aim at not collecting HW information, but periodically gathers system/application-level statistics. i.e., CPU, IO, and RT. This policy is because we consider that our target users are consumers for public IaaS clouds where provide limited access to the hypervisor layer and host machines [62].

2.3.2 Cloud user-centric approach

There are several attempts to address the performance interference from different applications in the user space. IC^2 [129] and DIAL [93] mitigate the performance interference for web server clusters. The performance interference is detected by resource monitoring or statistic inference model. Once the interference is identified, the approaches change application configurations or reduce request flows to the low-performed web nodes. However, these approaches are application-specific (web service), but our work in chapter 5 aims at supporting diverse types of FGs. Moreover, our approach does not change any application configuration. Such modifications could result in high overhead and only mitigate a short-term performance interference [128]. Stay-away [168] manages process containers (LXC or Docker) and mitigates the interference by throttling BGs. However, the control mechanism depends on a *diurnal* pattern of user-workloads, having a clear period of low intensity. This may not be true for modern cloud applications [11, 88] and, more importantly, we aim at designing an agnostic approach/framework to user workload patterns.

2.4 Cloud Simulation

Cloud simulations (e.g. CloudSim [26], iCanCloud [157], GreenCloud [111]) are widely used in cloud research community and many industry organizations to evaluate various aspects[1] of cloud infrastructures and applications. Since cloud applications and data center infrastructures are typically large-scale, actual deployment-based evaluation for these cloud systems are often technically infeasible (time and cost). To facilitate this evaluation and development process, cloud simulations have become a promising alternative for many cloud users and builders because the cloud simulations allow to evaluate various configurations for cloud applications and infrastructures in repeatable and controllable manners [26].

CloudSim [26] is the most successful simulation framework for cloud computing and is designed to support various simulation cases across three major cloud service models (e.g. SaaS, PaaS, and IaaS). CloudSim offers various capabilities for cloud simulations such as VM allocation and provisioning, energy consumption, network management, and federated clouds. It also has several extensions [25, 32, 63, 183, 198] due to its extensible simulation architecture. These extensions support simulations for large-scale cloud applications according to the geographical distribution of users and data centers [198], a network-based application model [63], complex scientific workflows [32], resource failure model during task executions [25], and SDN (Software-Defined Network) for data centers [183].

iCanCloud [157] is a holistic simulation platform for cloud computing and offers various simulation functionalities including resource provisioning, energy consumption, and user-defined resource broker model. The goal of iCanCloud is to predict a tradeoff point between cost and performance of cloud applications. iCanCloud offers a unique feature to configure various storage systems and a pre-defined IaaS model based on Amazon EC2 [5]. Moreover, iCanCloud supports large-scale simulation on distributed environments by employing MPI/POSIX-based API.

GreenCloud [111] is a packet-level simulator for data centers focusing on data communication and energy cost in cloud computing. This simulator is implemented based on NS-2 [87] and offers extensive network and communication capabilities (e.g. full implementation of TCP/IP model) to simulate data center operations. However, due to its narrow focus of simulation for cloud systems, it lacks many simulation

[1] Cloud simulators are often used to evaluate infrastructure/application performance, cost efficiency, and resource management policies of cloud systems.

capabilities. i.e. cloud resource management evaluations.

Even though existing cloud simulators offer many capabilities to facilitate cloud users and builders' evaluation tasks to cloud infrastructure and application, these existing works still have the following limitations.

First, existing cloud simulators tend to employ "bottom-up" approach to create simulation model, building the simulation model from low-level components (e.g. cloud infrastructures – network, storage, shared physical servers) [166]. Although, this bottom-up approach has several advantages (e.g. flexibility and extensibility), it is very hard for cloud users to build accurate simulation models for cloud infrastructure because cloud providers [5, 67, 144, 167] do not reveal their infrastructure details.

Table 2.1 summaries the representative capabilities of three major cloud simulators. We categorize the capabilities according to the layers of cloud infrastructure (data center) and applications. As Table 2.1 shows, existing predictors mostly support test cases and evaluations for data centers (e.g., physical and network resource management, power consumption management) and have less support for cloud applications and resource management. We believe that the less support for application layer is a disadvantage of bottom-up style simulation approaches.

Moreover, the results from the existing simulators are not validated with measurement from real-world cloud environment. When using existing cloud simulators, it is very hard to verify that simulation environments and configurations correctly represent the actual clouds (infrastructures and applications) [19]. Cloud simulations often generate different results with evaluation on actual clouds and user mechanisms (evaluated by cloud simulations) often cannot be reproducible in the actual cloud environments.

Table 2.1: **Simulation Capabilities of Existing Cloud Simulators.**

Simulation Capability		CloudSim [26]	iCanCloud [157]	GreenCloud [111]
Data center/ Infrastructure Layer	VM management (allocation, provisioning, scheduling, migration)	Yes	Yes	No
	Physical resource management and scaling	Yes	Yes	Limited
	Network resource management	Yes	Yes	Yes
	Power consumption management	Yes	Yes	Yes
	Federated cloud management	Yes	No	No
	Data center workload management	Yes	No	Yes
Application Layer	Horizontal VM autoscaling (scaling-in/out)	Limited	No	No
	Vertical VM autoscaling (scaling-up/down)	No	No	No
	Storage service management	No	Limited	No
	Job/application scheduling	Yes	Yes	No
	Billing management (cost optimization)	Limited	Limited	No
	Application/job failure management	No	No	No
	Performance uncertainty	No	No	No

Chapter 3

Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling

This chapter provides a detailed analysis of the performance of existing workload prediction techniques for cloud workloads. We begin this chapter with fundamental concerns/questions from cloud practitioners when designing/developing cloud applications with a predictive resource management system. And, we discuss the detail mechanisms of existing workload predictors applied to previous predictive approaches. We then evaluate the performance (e.g., statistical and cloud metrics) of these techniques with well-known workloads, and determine the best workload predictors for different cloud workloads as well as best predictive scaling styles through performance study with predictive resource management model.

3.1 Introduction

When and how to scale-out and scale-in a cloud application can be one of the most difficult challenges for a cloud application designer/deployer. Fundamentally, basing this decision on the current state of resources (e.g., VMs) for a given cloud application is usually simple and can be effective but is limited due to its reactive nature. For example, a standard use of Amazon Web Services' autoscaling mechanism is to configure it to add more "worker VMs" when the average current CPU utilization stays above a certain threshold for a short period time. However, fluctuating or generally unpredictable workload patterns can quickly eliminate any potential performance or

monetary improvements of such a policy/mechanism. In other words, intuitively any changes to the cloud application system state/configuration assume that the near-past workload will continue for the near-future.

Next-generation scaling approaches attempt to move beyond the limitations of such reactive systems by instead attempting to predict the near-future workloads. Over the past years, many studies have proposed predictive scaling approaches [21, 28, 31, 47, 60, 61, 64, 92, 154, 161, 162, 174, 181, 203, 207]. In general, due to the complexity of cloud environments, these approaches were often forced to make significant limiting assumptions in either the operating conditions/requirements or expected workload patterns. As such, it is extremely difficult for cloud users to know *which – if any – existing workload predictor will work best for their particular cloud activity*, especially when considering highly-variable workload patterns, non-trivial billing models, variety of resources to add/subtract, etc.

The goal of this chapter is to comprehensively evaluate existing cloud workload predictors, holistically, and often in a broader context than in the authors' evaluation methodology. Because the most common metric to evaluate workload predictors is accuracy for the future job arrivals, the first question we seek to answer is:

Question #1: *Which existing workload predictor has the highest accuracy for job arrival time prediction, when applied for different workload patterns?*

After considering the workload predictor in isolation, we evaluate it in combination with the resource scaling component. Therefore, the second question we seek to answer is:

Question #2: *Which existing workload predictor has the best cost efficiency and deadline miss rate (which represents performance and SLA requirements), when applied for different workload patterns, different scaling styles and different pricing models?*

Furthermore, some previous work employed predictive scaling-out, some employed predictive scaling-in, and some employed both. Given these choices of applying predictive scaling, the third question we seek to answer is:

Question #3: *Which style of predictive scaling (predictive scaling-out, predictive scaling-in, or both) achieves the best cost and performance benefit for particular cloud configurations (e.g. billing model, job deadline)? And how much benefit can be achieved?*

To answer these questions, we conducted comprehensive evaluations with a wide

range of configurations of predictive cloud resource scaling using 15 existing workload predictors [21, 22, 28, 45, 47, 112, 123, 133, 138, 154, 174, 176, 180, 190, 209]. We also included 6 well-known machine learning predictors that have not been used for the predictive scaling before. In total, we examined 21 predictors, covering naive, regression, temporal (time series) and non-temporal methods. We used 24 randomly generated workloads covering four common types of job arrival patterns [58, 193], which are growing, on/off, bursty and random. We also examined scaling operations including **RR** (Scaling-out: **R**eactive + Scaling-in: **R**eactive), **PR** (Scaling-out: **P**redictive + Scaling-in: **R**eactive), **RP** (Scaling-out: **R**eactive + Scaling-in: **P**redictive) and **PP** (Scaling-out: **P**redictive + Scaling-in: **P**redictive). We also considered both hourly and minutely pricing models. In our experiments, each configuration then covered one workload predictor, one workload, one scaling operation and one billing model. As a result, more than 4K ($21 \times 24 \times 4 \times 2$) configurations were examined. We run each configuration using the PICS [104] simulators to evaluate each configuration's cost and deadline miss rate. We chose PICS because it can accurately simulate real public IaaS clouds in short amount of time. Without PICS, it is both timely and financially infeasible to conduct such extensive experiments on real IaaS clouds.

Based on the experimental results, we successfully answer the three questions posed previously. Here we summarize our findings and answers to each question:

To find the best workload predictor in terms of the accuracy for diverse workload patterns (Question #1): the accuracies of different workload predictors vary considerably. The best workload predictors in terms of statistical accuracy are usually orders of magnitudes more accurate than the worst ones. However, no workload predictor is universally the best for all workload patterns. Each workload pattern has its own best workload predictor. We show the best workload predictors for each workload pattern in Section 3.4.1.

To find the best workload predictor in terms of cloud metrics such as cost efficiency and SLA satisfaction (Question #2): the workload predictor with the highest accuracy is not necessarily the best in terms of cost and deadline miss rate. Additionally, no workload predictor is universally the best for any workload pattern and billing model. However, we observe that the best predictor (in terms of cost and deadline miss rate) for a particular workload pattern is always one of the top 3 most accurate predictors (in terms of statistical accuracy) of that workload pattern. We also show the best workload predictors for each workload pattern in terms of cloud metrics in Section 3.4.2.

To find the best style of predictive resource scaling in terms of providing the best cost and performance benefits (Question #3): both predictive scaling-out (**PR**) and predictive scaling-in/out (**PP**) significantly reduces cost and deadline miss rate over purely reactive scaling (**RR**). However, predictive scaling-in (**RP**) performs similarly to **RR**. Overall, **PP** always provides the lowest cost and deadline miss rate for all workload patterns and billing models. On average, **PP** provides 30% less cost and 80% less deadline miss rate compared to **RR** or **RP**, and **PP** offers 14% less cost and 39% less deadline miss rate compared to **PR**.

A key finding from the answering those questions is that users, who want to design new algorithm for predictive resource scaling, should consider top 3 workload predictors depending on workload patterns, and use **PP** for their scaling operations in order to archive better cost efficiency and deadline miss rate.

3.1.1 Chapter Organization

The rest of this chapter is organized as follows: Section 3.2 describes workload predictors used in this work. Section 3.3 contains the experimental design of this work. Section 3.4 provides evaluation results for all predictors. Section 3.5 provides a discussion and future work. Section 3.6 summarizes this chapter.

3.2 Workload Predictors

We collect a total of 21 workload predictors via an extensive survey of previous research on predictive resource scaling. Each predictor is categorized into one of the following classes: 1) naive, 2) regression, 3) temporal, and 4) non-temporal predictor.

3.2.1 Naive Workload Predictors

There exist two types of naive workload predictors that are mean and recent mean-based (kNN) methods.

1. The **mean job arrival time-based predictor** forecasts a next job arrival time based on a mean arrival time of all previous jobs. For the scaling-out operation, the cloud application prepares cloud resources as if the next job will be arrived at the predicted result based on mean. For the scaling-in operation, the cloud application waits until the predicted next job's arrival time when a VM running

by the cloud application is idle in order to increase a possibility of reuse of this VM.

2. The **recent mean job arrival time-based predictor (kNN)** is a similar approach with mean-based predictor, but this uses the arrival time of recent k jobs and predicts the next job's arrival time based on a mean arrival time of those recent jobs.

3.2.2 Regression-based Workload Predictors

Regression-based approaches can be split into category of global and local regressions. Each category can include linear (1-degree) or polynomial (2 or more degrees) models. In total, we use 6 regression-based predictors, which are global and local regression with linear, quadratic, and cubic models.

1. **Global regression predictors – linear and polynomial regression models** forecast the next job arrival time by creating a linear or polynomial regression model [72] using features including all previous job arrival time. Here we only consider job arrival time as the main variable. Therefore, these approaches are a single variable regression models.

2. **Local regression predictors – local linear and polynomial regression models** use locally weighted regression models [72] to estimate the next job arrival time. These approaches consist of two steps: 1) applying a kernel function to select job arrival samples and 2) creating linear or polynomial regression model based on the samples. In this chapter, we use kNN (k Nearest Neighbor) as the kernel function for the local regression models to select proper samples. kNN calculates a distance between a target object (e.g. next job arrival time) and all possible samples (e.g. past job arrival time) by using absolute or Euclidean distance function. kNN then selects proper local samples (e.g. k recent jobs). Based on the selected samples from the kNN, a linear or polynomial regression model is created, and predicts the next job arrival time. The major difference between the global and local regression is the size and similarity of job arrival samples used for creating a regression model. Local regression uses smaller number of samples that are more similar to the prediction target. The local regression models often have less overhead for model creation and workload prediction.

3.2.3 Temporal (Time-Series)-based Workload Predictors

There exist various temporal (time-series) methods for the future workload prediction because these predictors [21, 28, 30, 45, 47, 112, 123, 133, 138, 154, 174, 176, 180, 190] are widely used to analyze workload patterns for cloud computing as well as other domains of computer systems research. We use four categories of temporal models: 1) **ES** (Exponential Smoothing), 2) **AR** (Autoregressive), 3) **ARMA** (Autoregressive and Moving Average), and 4) **ARIMA** (Autoregressive Integrated Moving Average).

1. **ES** includes four methods: WMA (Weighted Moving Average), EMA (Exponential Moving Average), and DES (Double Exponential Smoothing).

   - **WMA** [206] is a weighted sum of observed dataset (e.g. past job arrival information) and sum of weight for each observed data is 1.

$$WMA = \sum_{n=1}^{k} w_n x_{t+1-n}, s.t. \sum_{n=1}^{k} w_n = 1 \tag{3.1}$$

   - **EMA** is a similar approach as WMA, but it gives more weight to the most recent observation of job arrivals. We modeled EMA predictor based on [176]. EMA predicts the future job arrival time by $s_t = \alpha x_t + (1-\alpha)s_{t-1}$. α is a smoothing factor ($0 < \alpha < 1$). If α is close to 1, EMA has less smoothing effect and gives more weight to the recent data, and vice versa.
   - **DES** is used when the observed data show clearer seasonal trend. We use Holt-Winters and Brown's DES.
     (a) **Holt-Winters DES** predicts the next job arrival time by capturing a smoothing value at time t ($s_t = \alpha x_t + (1-\alpha)(s_{t-1} + b_{t-1}),\ where\ s_1 = x_1$) and estimating the trend at time t ($b_t = \beta(s_t - s_{t-1}) + (1-\beta)b_{t-1},\ where\ b_1 = x_1 - x_0$). x_0 is the first observation of raw data, α is a smoothing factor ($0 < \alpha < 1$) and β is a trend smoothing factor ($0 < \beta < 1$). Then, the next job arrival time is calculated by $s_t + b_t$.

$$s_t = \alpha x_t + (1-\alpha)(s_{t-1} + b_{t-1}),\ where\ s_1 = x_1 \tag{3.2}$$

$$b_t = \beta(s_t - s_{t-1}) + (1-\beta)b_{t-1},\ where\ b_1 = x_1 - x_0 \tag{3.3}$$

(b) **Brown's DES** predicts the next job arrival time by calculating $(2 + \frac{\alpha}{1-\alpha})s_t^{'} - (1 + \frac{\alpha}{1-\alpha})s_t^{''}$. $s_t^{'}$ is the first order exponential smoothing model and is expressed by $s_t^{'} = \alpha x_t + (1-\alpha)s_t^{''}$. x_t is current job arrival and α is a smoothing factor $(0 < \alpha < 1)$. $s_t^{''}$ is double-smoothed statistics and is expressed by $s_t^{''} = \alpha s_t^{'} + (1-\alpha)s_{t-1}^{''}$.

$$T_{NextJobArrival} = (2 + \frac{\alpha}{1-\alpha})s_t^{'} - (1 + \frac{\alpha}{1-\alpha})s_t^{''} \tag{3.4}$$

2. **AR** is a linear combination of previous data of the target object (e.g. job arrival time). [49] we model AR(p) based on [21]. AR(p) model is expressed in $X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-1} + \varepsilon_t$. p is the order of AR model, φ_i is the set of parameters of the model, c is constant, and ε_t is white noise.

$$X_t = c + \sum_{i=1}^{p} \varphi_i X_{t-1} + \varepsilon_t \tag{3.5}$$

3. **ARMA** is a combined model of AR and MA (Moving Average) and ARMA(p, q) is expressed in $X_t = \sum_{i=1}^{p} \varphi_i X_{t-1} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + c + \varepsilon_t$. The first term is AR($p$) model with the order of p. The second term is MA(q) model with the order of q. We modeled ARMA for workload prediction using [45, 123, 174, 180, 190].

$$X_t = \sum_{i=1}^{p} \varphi_i X_{t-1} + \sum_{i=1}^{q} \theta_i \varepsilon_{t-i} + c + \varepsilon_t \tag{3.6}$$

4. **ARIMA** is a generalization of ARMA and provides a reliable prediction for non-stationary time-series data by integrating AR and MA models. ARIMA is expressed as ARIMA(p, d, q), where p is the order of AR, q is the order of MA, and d is the order of differencing model. We modeled ARIMA for workload prediction based on [28, 47].

3.2.4 Non-temporal Workload Predictors

Non-temporal workload predictors have not been applied to the cloud resource scaling before. These predictors, however, have provided accurate prediction results within a deterministic amount of time. We consider several non-temporal approaches to

predict the next job arrival time and select three categories of non-temporal prediction approaches: SVM (Support Vector Machine), decision tree, and ensemble methods.

1. **SVM** is an optimal margin-based classifier that tries to find a small number of support vectors (data points) that separate all data points of two classes with a hyperplane in a high-dimensional space [72]. With kernel tricks, it can be extended as a nonlinear classifier to fit more complex cases. SVM can be applied to the case of regression as well which contains all the main features that characterize the maximum margin based algorithm. At testing time, the (positive or negative) distance of a data point to the hyper-plane is output as the prediction result. We consider both linear and non-linear SVM. We use linear kernel for Linear-SVM and Gaussian kernel for non-linear-SVM (Gaussian-SVM). Linear-SVM is to focus on the workloads that have relatively clear trend factors and Gaussian-SVM is to predict the workloads with non-linear characteristics.

2. **Decision tree** is a non-parametric learning algorithm and it has also been used for both classification and regression problems [72]. Decision tree creates a classification or regression model by applying decision rules derived from features of dataset. Decision tree is known as its simple (tree-based) structure and fast execution time for numerical data.

3. **Ensemble prediction methods** employ multiple numbers of predictors to obtain better generalizability and increase performance. Ensemble methods use bagging or boosting approaches to reduce variance (bagging) or bias (boosting) on prediction results. We select three ensemble methods including gradient boosting, extra-trees, and random forest [72].

3.3 Experiment Design

3.3.1 Design of Cloud Resource Management System

To evaluate all workload predictors, we designed a cloud management system for resource scaling as shown in Figure 3.1. This system consists of three modules: job portal, resource management module (RMM) and predictive scaling module (PSM). The job portal is an entry for the workloads (jobs from end-users). A job's arrival triggers two other modules. A newly arrived job is passed to the RMM. The RMM selects a proper VM based on the job's duration and deadline. More specifically, the

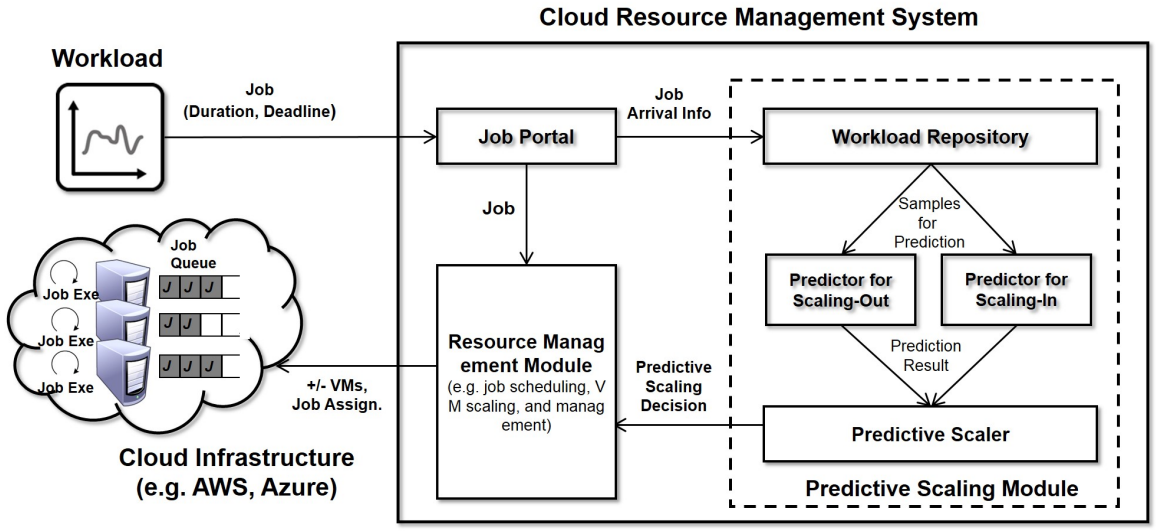


Figure 3.1: **Cloud Resource Management System with Predictive Scaling.**

Algorithm 1 Predictive Scaling-Out

Require: A new job arrives

```
samples ← get_samples_for_prediction ()
next_job ← predict_next_job_arrival (samples)
vm_type ← select_proper_vm_type (next_job)
vm_list ← current_running_vms (vm_type)

if vm_list is empty then
    create_vm (vm_type, time_to_start)
else
    do_nothing ()
end if
```

RMM creates a list of VMs that meets the deadline of the job by comparing the performance of different VM types with the job's duration and deadline. the RMM then selects the most cost efficient VM (i.e., cost/performance-ratio [107]) from the candidates. Note that the algorithm, used in this cloud resource management, focuses primarily on improving the job deadline satisfaction than reducing the cloud cost. Once a proper VM is selected, the RMM schedules the job to the selected VM via "Earliest Deadline First" scheduling. The selected VM is then used to execute the job. A new job's arrival activates the PSM as well. The job's arrival information is stored in the workload repository. The workload information from this repository is used by both predictive scaling-out and scaling-in.

1. **Predictive Scaling-out Operation (Algorithm 1)** is triggered when a job

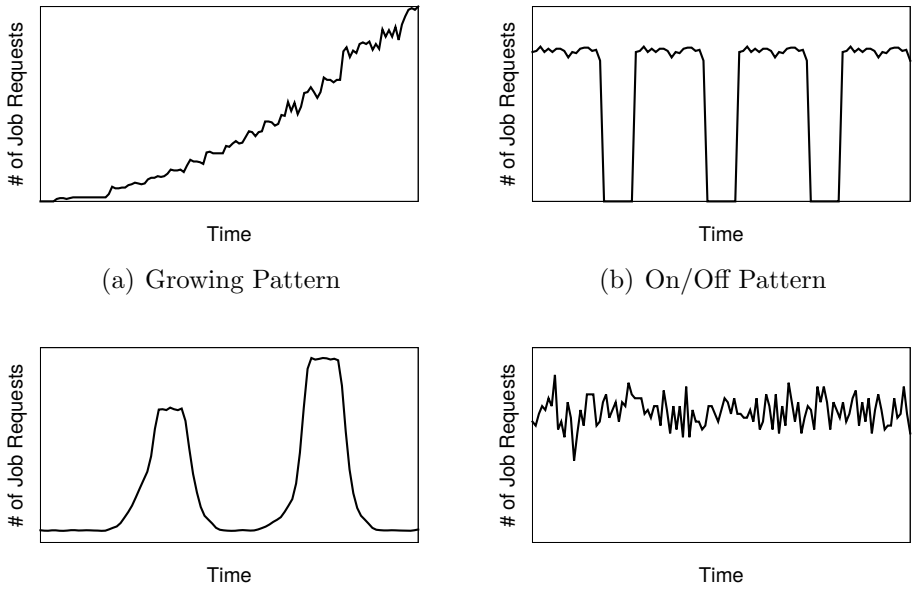
arrives. A prediction obtains proper amount of job arrival samples for prediction (**line 1**) and forecasts the next job's arrival time in the future (**line 2**). Based on the prediction result, this operation chooses a proper type of VM for the future job (**line 3**) as explained in the previous paragraph (In Algorithm 1, we assume that the duration and deadline of the future job are known). This operation selects a list of currently running VMs to execute the next job (**line 4**). If the list is empty (**line 6**), then a new VM will be created for the next job (**line 7**).

Algorithm 2 Predictive Scaling-In

Require: *vm* is idle

```
1:  samples = get_samples_for_prediction (vm)
2:  next_job = predict_next_job_arrival (samples)
3:
4:  if next_job arrival ≤ max_startup_delay then
5:     scale_in_time ← next billing boundary after next_job arrival
6:  else
7:     scale_in_time ← this billing boundary
8:  end if
9:
10: repeat
11:    if next_job arrives then
12:       go to new_job_processing_routine()
13:    end if
14: until scale_in_time
15:
16: terminate (vm)
```

2. **Predictive Scaling-in Operation (Algorithm 2)** is triggered when a VM is idle – no jobs in both processor and work queue. The workload predictor for scaling-in operation estimates the next job arrival time to the idle VM (**line 1–2**). Then we compare the predicted job arrival time with maximum VM startup delay [130]. If the job arrival time is smaller than the max startup delay, we choose to keep this VM for at least max startup delay time; otherwise, we choose to terminate this VM (**line 4–8**). The rationale behind this choice is explained as follows. For any new job, starting a new VM for it takes (startup time + job duration) to execute it. However, if we use existing idle VM, it takes (job arrival time + job duration) to execute it. Therefore, if job arrival time is smaller than startup time, it is faster and cheaper to use the existing VM; otherwise, it



(a) Growing Pattern (b) On/Off Pattern

(c) Bursty Pattern (d) Random Pattern

Figure 3.2: **Cloud Workload Patterns. X-axis means time and Y-axis means the number of requests (e.g. the number of jobs)**

is cheaper to use new VM. Moreover, we choose to terminate a VM only at nearest billing boundary, because we have already paid for this billing cycle. If the next job arrives within the predicted time, the job is assigned/processed by this VM (**line 12**). If there is no more jobs until the predicted time, this VM is terminated (**line 16**).

3.3.2 Cloud Workload Patterns

We generate synthetic workloads based on four well-known cloud workload patterns (shown in Figure 3.2). We create six workloads for each workload pattern with different mean and standard deviation of job arrival time/duration to reflect various and realistic cloud usage scenarios. (in total, we use 24 workload patterns for the evaluation.) The detail of each dataset is described in Table 3.1. For the growing workload pattern, we first generate a quadratic base function and then apply Poisson distribution to randomize the arrival time of a particular job. The on/off workload pattern has four active periods and three inactive periods. For the active periods of the on/off workload pattern, we use growing and declining quadratic functions. The bursty workload pattern has 6 – 7 peak spikes periods and other stable periods. To

Table 3.1: **Workload Dataset Information**

Workload	# of Jobs	Mean Job Arrival Time	Std. Dev. of Job Arrival Time	Job[1] Duration	Job Deadline
Growing	10K	25s	60.5	Average: 450s, Std.Dev.: 270	Average: 500s, Std.Dev.: 250
On/Off			375.5		
Bursty			35.3		
Random			270		

generate the random workload pattern, we use Poisson distribution for the random job arrivals.

3.3.3 Implementation Details

We implemented the cloud resource management system on top of PICS [104], a Public IaaS Cloud Simulator. In addition to the simulation model, we implement all the predictors (described in Section 3.2) and scaling-in/out mechanisms using numpy[2], Pandas (Python Data Analysis Library)[3], Statsmodels[4], and scikit-learn machine learning packages[5].

Choosing the parameters and the training sample size are very crucial to all workload predictors in order to provide the best possible prediction performance. For the decision of the training sample size for predictors, it is obvious that a predictor should use as many sample as possible to maximize the accuracy of the prediction. However, large size of training samples increases the overhead of prediction. A constraint for the prediction is that the predictors should be able to forecast the next job arrival time before the actual job comes to our cloud application. To this end, we tested a wide range of sample size and determine the size based on a tradeoff between the prediction overhead and the prediction accuracy. In this chapter, all predictors (except global regression approach) uses 50 – 100 of most recent job arrival samples for forecasting the future job arrival time prediction.

For the parameter selection of the workload predictors, we use either a performance-based or a grid search approach [20]. For AR, ARMA, and ARIMA model, we employ

[1]This is job duration on smallest (worst performance) VM instance (small EC2 instance [5] in our experiment design). By using the job duration and deadline, the cloud resource management system (Section 3.3.1) determines a proper VM type that can meet deadline.

[2]http://www.numpy.org/

[3]http://pandas.pydata.org/

[4]http://statsmodels.sourceforge.net

[5]http://scikit-learn.org/

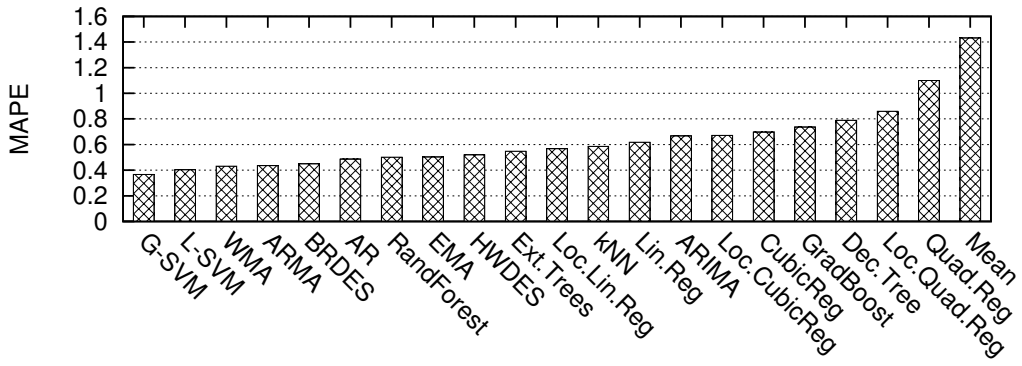


Figure 3.3: **MAPE Results of All 21 Predictors.**

the performance-based parameter selection, and we choose the first order of these three models. i.e., AR(1), ARMA(1, 1), ARIMA(1, 1, 1) Higher-order of these models is not desirable because these three workload predictors require high computation time. It is often impossible for the higher order of these models to predict the next job arrival time before the actual job arrives. For other temporal-based workload predictors (e.g. EMA and DES), we leverage a grid search approach for the parameter selection that tries every possible parameter within its range constraints (e.g. $0 < \alpha < 1$ and $0 < \beta < 1$ for Holt-Winters DES). Moreover, SVM predictors require soft and kernel parameters (e.g. Gaussian-SVM needs both parameters and Linear-SVM requires only the soft margin parameter). We choose these parameters that result in best prediction performance. The range we have considered is from $10e^{-6}$ to $10e^{3}$ for both parameters.

3.4 Evaluation

3.4.1 Evaluation of Accuracy for Workload Predictors

As the first evaluation of this chapter, we measure all 21 workload predictors' accuracy for predicting the future job arrival time under four different workload patterns. We employ MAPE (Mean Absolute Percentage Error) [72] to statistically measure the prediction accuracy.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{T_{actual,i} - T_{predicted,i}}{T_{actual,i}}\right| \tag{3.7}$$

Figure 3.3 shows MAPE results of all 21 predictors. Average of the MAPE for all

21 predictors is 0.6360. Overall, two SVM approaches have the best MAPE results. Other three best predictors are WMA, ARMA, and Brown's DES. The MAPE value of Gaussian-SVM predictor (the best predictor) is 0.3665, which is 42% less result than average of all predictors. However, the best predictors in overall do not necessarily mean the best predictor for each workload pattern, so we also present the performance of all predictors for each workload pattern.

Table 3.2: **The MAPE Results of Workload Predictors Under Four Different Workload Patterns. (WL: Workload, GR: Growing, OO: On/Off, BR: Bursty, RN: Random)**

WL	Rank	Predictor	MAPE	WL	Rank	Predictor	MAPE
GR	**1**	**L-SVM**	**0.28**	OO	**1**	**G-SVM**	**0.22**
	2	AR	0.29		2	ARMA	0.30
	3	ARMA	0.30		3	L-SVM	0.44
	Avg.	–	0.51		Avg.	–	0.69
	Worst	Qua.Reg	2.75		Worst	Loc.Cub.Reg	1.25
BR	**1**	**ARIMA**	**0.38**	RN	**1**	**G-SVM**	**0.45**
	2	BRDES	0.41		2	Lin.Reg	0.46
	3	L-SVM	0.43		3	L-SVM	0.46
	Avg.	–	0.75		Avg.	–	0.52
	Worst	mean	3.35		Worst	Dec.Tree	0.62

Table 3.2 shows the MAPE results for each workload pattern. Due to page limitation, Table 3.2 only contains the best 3 predictors, average results, and the worst predictor. As shown in Table II, different workload patterns have different best predictors: **Linear-SVM** (growing), **Gaussian-SVM** (on/off), **ARIMA** (bursty), and **Gaussian-SVM** (random). Table 3.2 also shows that the MAPE results of the top three predictors are very similar to each other for growing and bursty workloads. These workloads have clear trend patterns, and many good workload predictors can successfully detect these patterns when using enough job arrival samples. The MAPE results of random workload are lower compared to other workloads, indicating random workload is the most difficult to predict. This difficulty is primarily caused by the fact that job arrival times in random workloads do not have clear trend pattern for predictors to discover.

Moreover, obtaining the prediction results in a deterministic amount of time is a critical issue for the predictive resource scaling. We also measured computation overhead of predictors (Figure 3.4). kNN, WMA, and EMA are the fastest predictors.

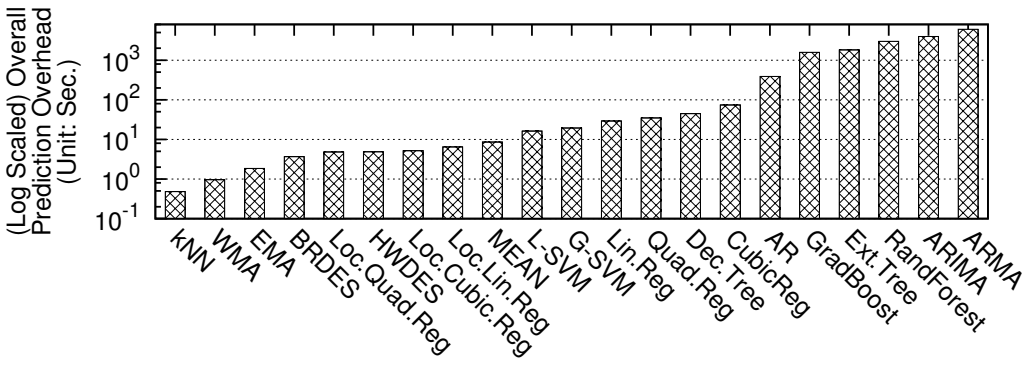


Figure 3.4: **Prediction Overhead (Log-Scaled Sum of Total Prediction Time) for All the 21 Workload Predictors. (Sample Size: 50 recent jobs)**

The overall prediction time for 10K jobs of these three predictors are 0.48 (kNN), 0.96 (WMA), and 1.86 seconds (EMA). However, some temporal approaches (AR, ARMA, and ARIMA) and ensemble methods (extra trees, gradient boosting, and random forest) have longer prediction time. The highest overhead predictor is ARMA, which takes 6031.52 seconds for 10K jobs.

3.4.2 Performance of Different Styles of Predictive Scaling

We measure the performance of four different styles of scaling operations for cloud resource management:

- **Baseline**: **RR** (Scale-Out: **R**eactive + Scale-In: **R**eactive)
- **PR** (Scale-Out: **P**redictive + Scale-In: **R**eactive)
- **RP** (Scale-Out: **R**eactive + Scale-In: **P**redictive)
- **PP** (Scale-Out: **P**redictive + Scale-In: **P**redictive)

PR is the most common style of predictive scaling for cloud application. **PR** predictively scales out cloud resources and reactively scales in cloud resources. **RP** is another way of predictive scaling, and it uses a reactive way for scaling-out and a predictive approach for scaling-in. **PP** is a combination of **PR** and **RP**, and it leverages a workload predictor for both scaling-out and scaling-in operations. For this evaluation, we use **RR** (Scaling-Out: **R**eactive + Scaling-In: **R**eactive) as a baseline. **RR** adds a new VM (scaling-out) when a new job needs an extra VM, and

terminates an idle VM (scaling-in) at its billing boundary. (i.e., hourly bound or minutely bound).

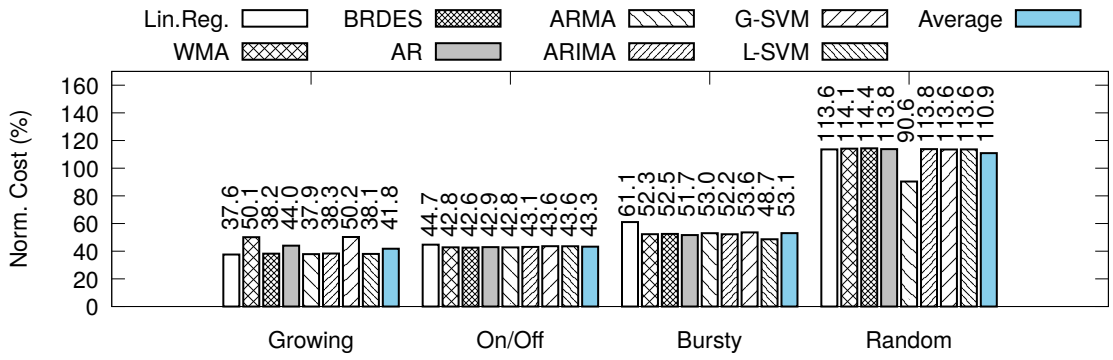
Due to page limitation, we only show the results of the predictive scaling operations with the most accurate 8 workload predictors from Section 3.4.1. These predictors are Linear-SVM, Gaussian-SVM, ARMA, AR, WMA, ARIMA, Brown's DES, Linear regression, which cover the overall best 5 predictors and the best 3 predictors for each workload pattern.

To evaluate the predictive scaling operations, we use two common cloud metrics (cost and job deadline miss rate) and two different billing models (hourly and minutely pricing model). Cost is to evaluate each scaling operations' cost efficiency, and job deadline miss rate represents the SLA-satisfaction requirement. We use two different billing models for cloud infrastructure, because major commercial IaaS clouds employ either hourly (e.g. AWS) or minutely (e.g. MS Azure) pricing model. We also use four different workload patterns (growing, on/off, bursty, and random workload).

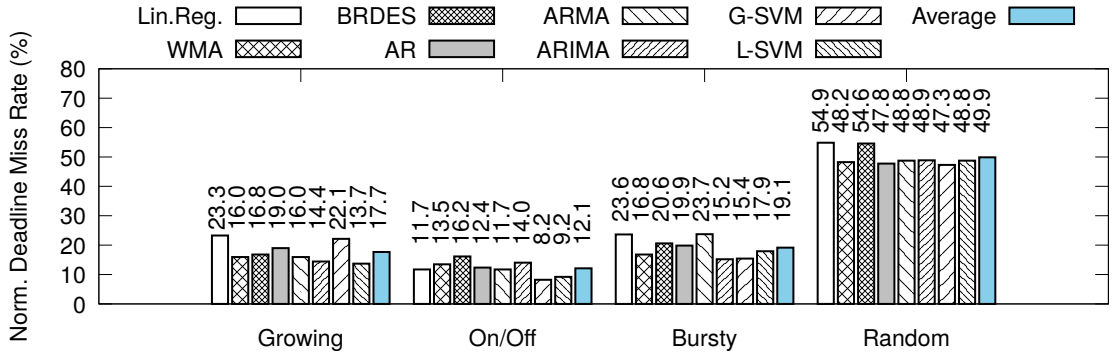
The goals of this evaluation are:

- Measuring the actual benefits from predictive scaling.
- Determining the best style of predictive scaling.
- Finding the best workload predictor for each workload pattern in terms of cloud metrics.

Case #1 – PR (Scale-Out: **P**redictive + Scale-In: **R**eactive): Figure 3.5 shows a normalized cost and job deadline miss rate (all results are normalized to **RR**) of **PR** for hourly pricing model. The results show that **PR** can improve 47%–58% of cost efficiency for growing, on/off, and bursty workloads. However, for random workload, **PR** has 11% of worse cost efficiency over the baseline. In terms of job deadline miss rate, **PR** has 50%–88% of less job deadline miss rate over the **RR** (baseline). For the hourly pricing model, **PR** shows relatively deficient performance for random workload in both cost efficiency and job deadline miss rate. This is because random workload is harder to predict than the other workload patterns. We rank the workload predictors for the **PR** based on the deadline miss rate, because it is more important to ensure that jobs meet their deadlines. Only after the deadline requirements are met, our cloud resource manager in Section 3.3.1 optimizes for cost efficiency. The best workload predictors for **PR** are: **Linear-SVM** (13.7%) for growing, **Gaussian-SVM** (8.2%) for on/off, **ARIMA** (15.2%) for bursty, and **Gaussian-SVM** (47.3%) for random workload.



(a) Normalized Cost.

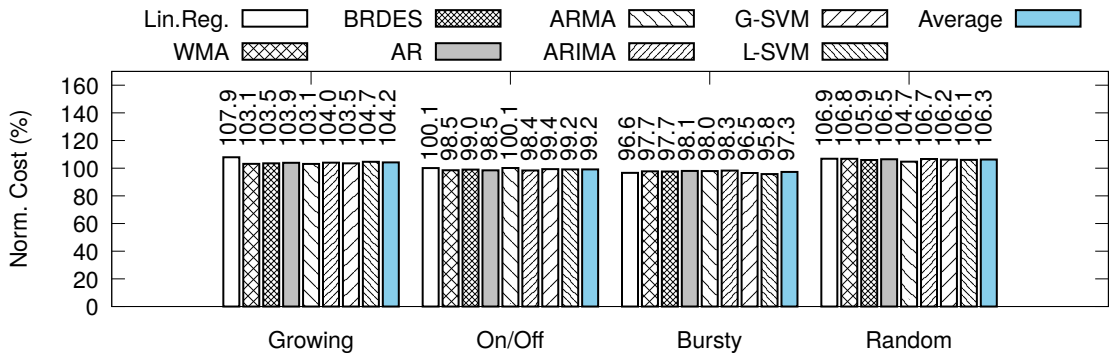


(b) Normalized Job Deadline Miss Rate.

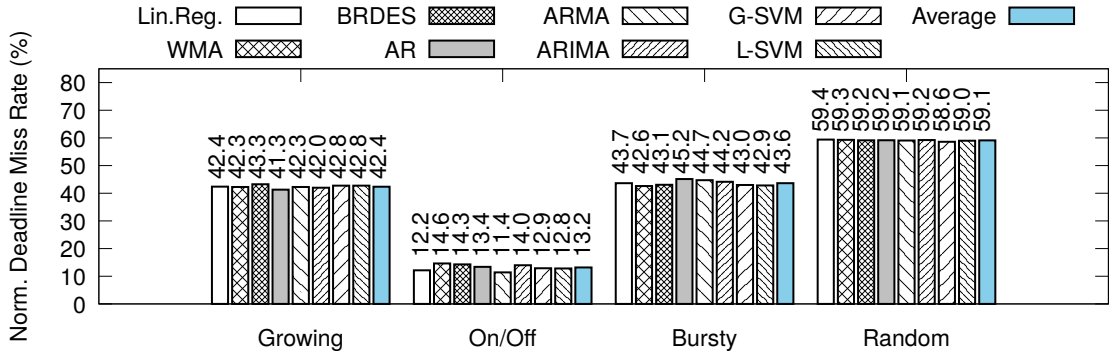
Figure 3.5: **Case #1 – Normalized Cost and Job Deadline Miss Rate of PR (Scaling-Out:Predictive + Scaling-In:Reactive) – Hourly Pricing Model.**

Figure 3.6 shows evaluation results of **PR** for minutely pricing model under four workload patterns. **PR** has similar cost efficiency with **RR**, but it has 41%–87% of less job deadline miss rate than the baseline. Thus, **PR** provides better job deadline satisfaction without dramatically increasing cost. The reason that **PR** has similar cost efficiency with **RR** is that the minutely pricing model is designed to provide better cost efficiency than hourly model to the user. So it is very hard to improve the cost efficiency for the minutely pricing model even though we have a good predictor. The best workload predictors for **RP** with minutely pricing model are: **AR** (41.3%) for growing, **ARMA** (11.4%) for on/off, **WMA** (42.6%) for bursty, and **Gaussian-SVM** (58.6%) for random workload.

Case #2 – RP (Scale-Out: **R**eactive + Scale-In: **P**redictive): Figure 3.7 and 3.8 show the evaluation results of **RP** for both pricing models under four workload patterns. The results indicate that **RP**'s benefit to the cloud system is not as much



(a) Normalized Cost.



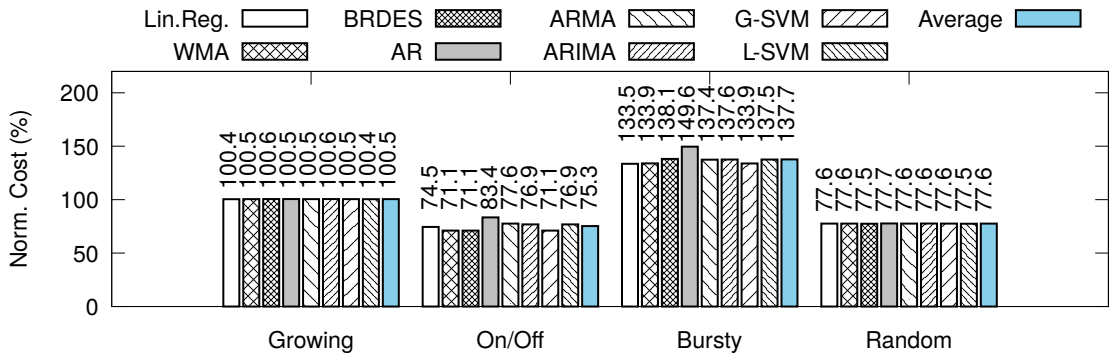
(b) Normalized Job Deadline Miss Rate.

Figure 3.6: **Case #1 – Normalized Cost and Job Deadline Miss Rate of PR (Scaling-Out:Predictive + Scaling-In:Reactive) – Minutely Pricing Model.**

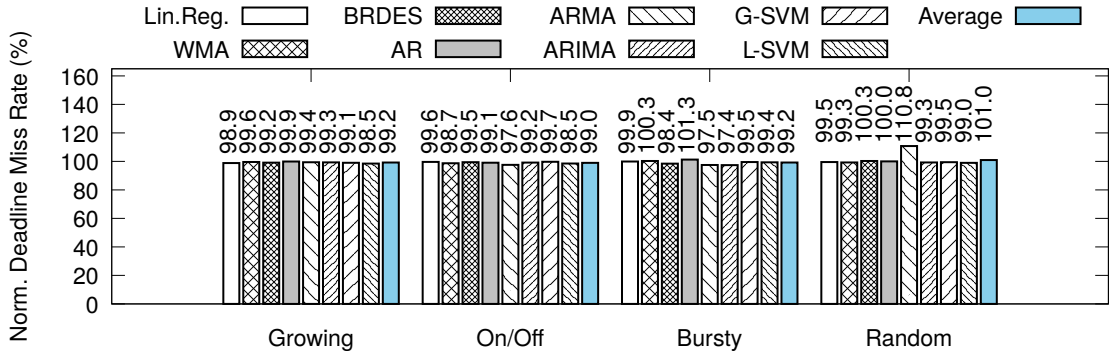
as the benefits from **PR**. The only benefit from the **RP** is the improved cost efficiency (on/off and random workloads for hourly pricing model, random workload for minutely pricing model). The improvement of cost efficiency is 12%–25%, but it has no benefits for job deadline miss rate.

Case #3 – PP (Scale-Out: **P**redictive + Scale-In: **P**redictive): Figure 3.9 shows normalized costs and job deadline miss rates of **PP** for hourly pricing model under four workload patterns. The results show that **PP** improves 48%–69% of cost efficiency for the four workloads over **RR**, and has 78%–93% of less job deadline miss rate than **RR**. The best workload predictors for the **PP** with hourly pricing model are: **Linear-SVM** (5.5%) for growing, **Gaussian-SVM** (7.8%) for on/off, **Brown's DES** (5.1%) for bursty, and **Linear-SVM** (18.1%) for random workload.

Figure 3.10 shows evaluation results of **PP** for minutely pricing model under four workload patterns. The results show that **PP** slightly improves cost efficiency ($\leq$



(a) Normalized Cost.



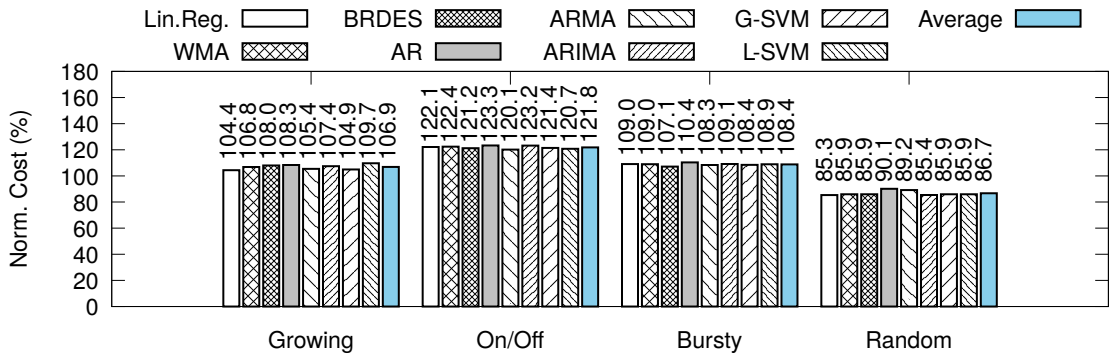
(b) Normalized Job Deadline Miss Rate.

Figure 3.7: **Case #2 – Normalized Cost and Job Deadline Miss Rate of RP (Scaling-Out:Reactive + Scaling-In:Predictive) – Hourly Pricing Model.**

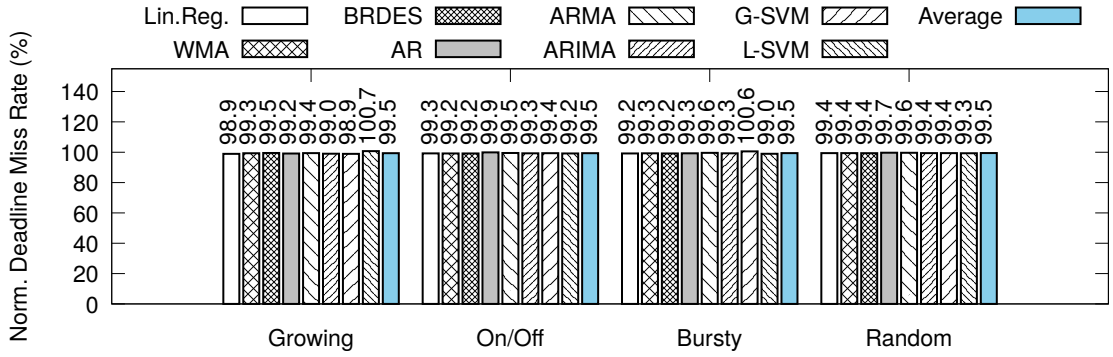
11%) of the cloud system and has huge improvement (60%–87%) for the job deadline miss rate. The best workload predictors of **PP** with minutely pricing model for each workload pattern are: **AR** (27.5%) for growing, **ARMA** (11.4%) for on/off, **Brown's DES** (19.7%) for bursty, and **Gaussian-SVM** (39.5%) for random workload.

Comparison of Three Predictive Scaling Styles: So far, we have separately evaluated three predictive scaling styles of the cloud resource management. In the following paragraphs, we present the overall benefit of predictive scaling. And we compare the results of these three predictive scaling styles to determine the best one for cloud resource management. Figure 3.11 shows that the comparison of average results of normalized cost and job deadline miss rate for **PR**, **RP**, and **PP** in both hourly and minutely pricing models.

For the hourly pricing model (Figure 3.11(a)), we found that **PP** is the best style for cloud resource management in terms of better cost efficiency and less job deadline



(a) Normalized Cost.

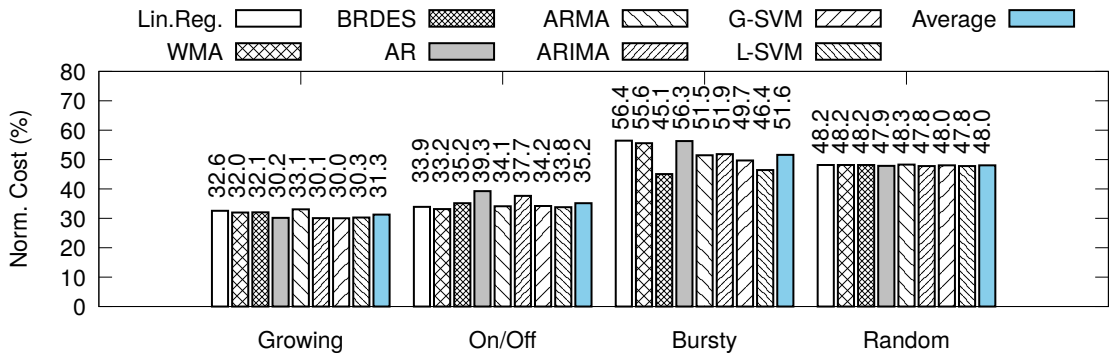


(b) Normalized Job Deadline Miss Rate.

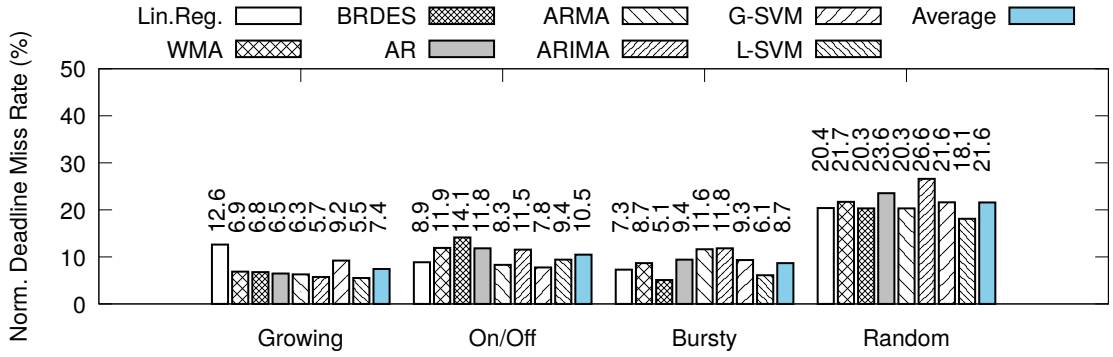
Figure 3.8: **Case #2 – Normalized Cost and Job Deadline Miss Rate of RP (Scaling-Out:Reactive + Scaling-In:Predictive) – Minutely Pricing Model.**

miss rate. **PP**'s cost efficiency is 20% (compared to **PR**) and 56% (compared to **RP**) better than other two approaches. Moreover, **PP**'s job deadline miss rate is 13% (compared to **PR**) and 88% (compared to **RP**) lower than others. This result is interesting, because although predictive scaling-in does not improve cost and deadline miss rate by itself (as shown in Case #2), it provides considerable improvement for both metrics when combined with predictive scaling-out. These results show that predictive scaling in/out approach (**PP**) (with a good workload predictor) helps to improve the performance of the cloud resource management.

For the minutely pricing model (Figure 3.11(b)), the job deadline miss rate of **PP** outperforms other two styles of predictive scaling operations. **PP** has 12% (compared to **PR**) and 72% (compared to **RP**) of less job deadline miss rate. **PP** also improves cost efficiency over **PR** and **RP**. These results suggest **PP** can significantly reduce deadline miss rate without cost overhead.



(a) Normalized Cost.

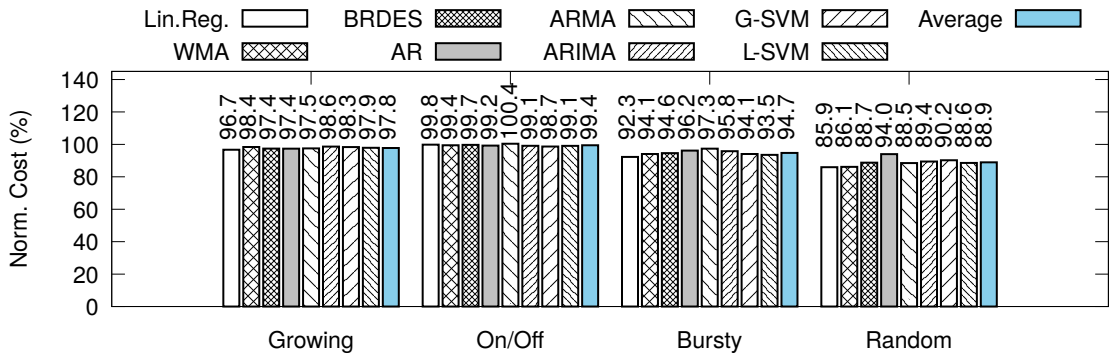


(b) Normalized Job Deadline Miss Rate.

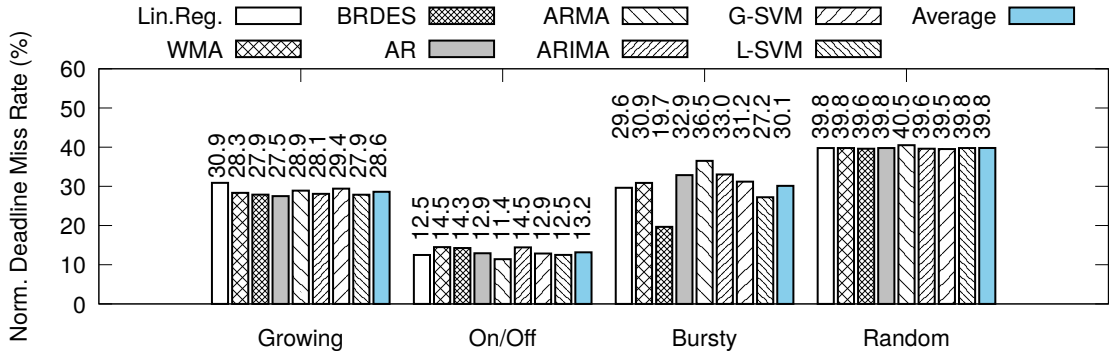
Figure 3.9: **Case #3 – Normalized Cost and Job Deadline Miss Rate of PP (Scaling-Out:Predictive + Scaling-In:Predictive) – Hourly Pricing Model.**

To understand the reasons of 1) **PP** significantly improves cost efficiency (hourly pricing model) and deadline miss rate (both pricing model) and 2) **RP** does not improve the performance by itself, we analyze the number of created VMs and VM utilization of three styles. Figure 3.12 represents the VM numbers and utilization of three scaling styles for both pricing models. For the both pricing models, **PP** creates the less number of VMs and has higher utilization than others. The reasons that **PP** has high VM utilization and lower number of created VMs are:

- Predictive scaling-out of **PP** uses more currently running VMs for the (near) future jobs, and creates less VMs for the (near) future jobs.
- Predictive scaling-in of **PP** keeps VMs alive for (further) future jobs, which further reduces the new VM creations, and increases the utilizations of existing VMs.



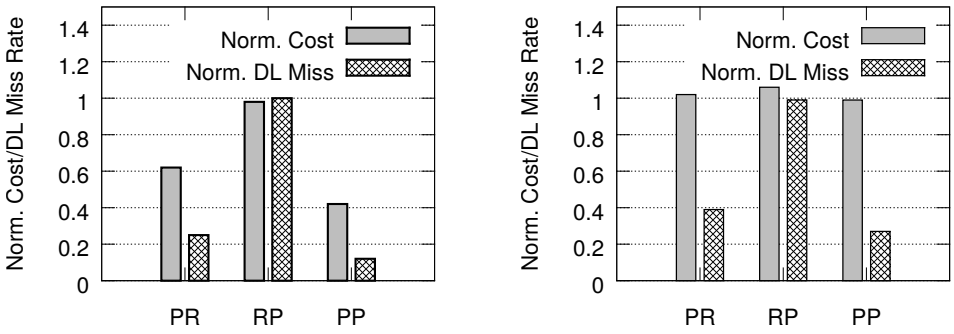
(a) Normalized Cost.



(b) Normalized Job Deadline Miss Rate.

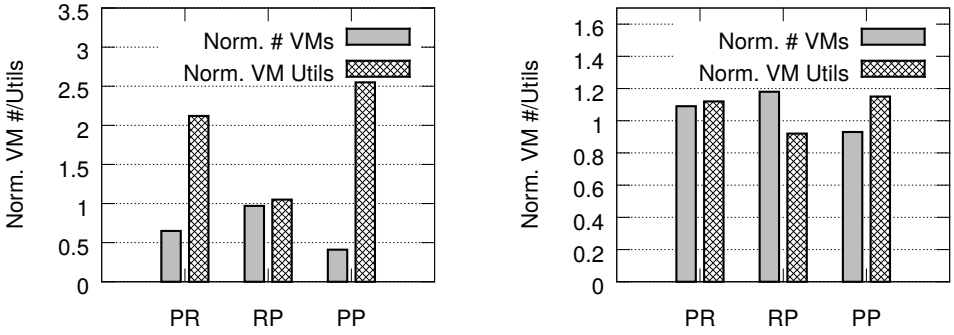
Figure 3.10: **Case #3 – Normalized Cost and Job Deadline Miss Rate of PP (Scaling-Out:Predictive + Scaling-In:Predictive) – Minutely Pricing Model.**

Moreover, the reason that **RP** cannot improve the cloud metrics is related to reactive scaling-out of **RP**. Reactive scaling-out operation creates VMs when jobs actually arrive, so **RP** has to create better performance VMs (more expensive VMs) in order to meet the jobs' deadline. This is because **RP** has no advance preparation for eliminating the overhead of the VM creation (e.g. startup delay). Also, most of VMs should be terminated after processing the current job because they are not used for future jobs. So, predictive scaling-in of **RP** does not help in this case because most of VMs should be destroyed.



(a) Hourly Pricing Model (b) Minutely Pricing Model

Figure 3.11: **Comparison of Average Results of Normalized Cost and Job Deadline Miss Rate of Three Scaling Styles (PR, RP, and PP).**



(a) Hourly Pricing Model (b) Minutely Pricing Model

Figure 3.12: **Comparison of VM Numbers/Utilization of Three Scaling Styles.**

3.5 Discussion

3.5.1 Mixing Workload Predictors

In Section 3.4.2, we observed that the predictive scaling should be used for both scaling-out and scaling-in for the best results. However, this observation raises another question: *Should we always use the same workload predictor for both scaling-in and scaling-out? Or, is there any benefit of using two different workload predictors for scaling-in and out?* Because scaling-in and scaling-out use different standard to manage VMs, they may prefer different workload predictors.

To answer these questions, we conducted additional experiments for **PP** using the 8 best workload predictors mentioned in Section 3.4.2. As we vary the workload

predictors for scaling-in and scaling-out, there are 64 possible combinations of predictors. Limited by space, we cannot provide results for all 64 combinations. Instead, we present the results of the best combinations for each workload pattern in Table 3.3(a) and 3.3(b). As these Tables show, except for the growing and random workloads under hourly pricing model, mixing two workload predictors has better results than using only one predictor. However, it is still unclear how to properly choose the best combination. We plan to solve this problem in the future.

3.5.2 Ensemble of Top Predictors

As shown in Section 3.4, no workload predictor is universally the best for every workload pattern and pricing model, in terms of both prediction accuracy and cloud metrics. Furthermore, the most accurate workload predictor does not necessarily provide the lowest cost and deadline miss rate. These two observations, along with the fact that mixing workload predictor is better than a single workload predictor, indicate that predictive scaling cannot simply rely on one workload predictor to achieve the best result. Instead, an ensemble method that combines top workload predictors may be the best for predictive scaling. For best results, this ensemble method may also have to dynamically weight its workload predictors based on workload pattern. We plan to investigate this ensemble method in the future.

3.6 Chapter Summary

To design proactive cloud resource management system, it is non-trivial for cloud users to determine the best workload predictor of predictive resource scaling due to the enormous number of predictor proposed, diverse workload patterns, and various cloud configurations.

In order to help the cloud users select the best workload predictor, we comprehensively evaluated 21 (existing) workload predictors using statistical and cloud metrics. For the performance evaluation of all workload predictors, we simulated a cloud resource management system with all 21 workload predictors using PICS simulator. We then evaluated the prediction accuracy of all workload predictors, and cost/deadline miss rate of three styles of predictive scaling (**PR**, **RP**, and **PP**) for diverse workload patterns and different cloud configurations.

The evaluation results show that:

Table 3.3: **Normalized Cost and Deadline Miss Rate of Mixed Workload Predictors.**

(a) Hourly Pricing Model.

Workload	**Mixed Workload Predictor**		**Normalized Cost**	**Normalized DL Miss Rate**
	Scale-Out	**Scale-In**		
Growing	L-SVM	L-SVM	30%	4%
On/Off	G-SVM	L-SVM	30%	7%
Bursty	BRDES	ARIMA	47%	4%
Random	L-SVM	L-SVM	48%	18%

(b) Minutely Pricing Model.

Workload	**Mixed Workload Predictor**		**Normalized Cost**	**Normalized DL Miss Rate**
	Scale-Out	**Scale-In**		
Growing	AR	BRDES	98%	27%
On/Off	ARMA	AR	102%	9%
Bursty	BRDES	Lin. Regression	93%	16%
Random	G-SVM	BRDES	88%	38%

- In terms of statistical metrics, the accuracies of different workload predictors vary considerably. However, no predictor is universally the best for all workload patterns. Each workload pattern has its own best predictor.
- The most accurate workload predictor is not necessarily the best in terms of cloud metrics. Moreover, no predictor is universally the best for all workload patterns and billing models. However, we observe that the best predictor for a particular workload pattern is always one of the top 3 most accurate predictors (with statistical metric) of that workload.
- **PR** and **PP** approaches significantly reduce cost and deadline miss rate over **RR**. Overall, **PP** always provides the lowest cost and deadline miss rate for all workload patterns and billing models.

In summary, to design a new predictive cloud resource scaling, the cloud users should consider top 3 workload predictors depending on their workload patterns, and use **PP** approach for their scaling operations in order to maximize the scaling performance. Moreover, we examined the benefit of using different predictors for scaling-out and scaling-in in **PP**. Our results show that mixing workload predictors has lower cost and deadline miss rate than using a single predictor. These results further suggest that ensemble of top workload predictors may be the best for predictive scaling.

Chapter 4

CloudInsight: Addressing Dynamic and Multi-Type Cloud Workloads in Predictive Scaling

4.1 Introduction

As discussed in Chapter 3, different cloud workload patterns have different best predictors, and top three predictors often show similar performance in predicting future workload changes. Therefore, cloud application designers/deployers may develop a predictive resource management system by using a single *static* (or "*one-size-fits-all*" style) workload predictor with the simplifying assumption that their workload has a stable pattern (e.g., increasing, cyclic bursty, and on-and-off) over time. And the similar approach is widely leveraged by many existing predictive systems [21, 28, 31, 47, 55, 61, 100, 113, 138, 154, 161, 164, 174, 190, 202, 203, 213, 217].

However, the "*one-size-fits-all*" approach is *insufficient* to address real-world cloud workloads because the patterns of real-world cloud workloads are usually unknown a priori because the real workloads are complicated in that they may experience dynamic pattern-shift/fluctuation over time. This dynamic nature of real workloads can be confirmed with traces from real-world cloud applications [33, 34, 169, 170]. Figure 4.1 illustrates the job/request arrival rate of the traces within Google [200] and Facebook [187]. Neither trace follows regular patterns and both traces are more likely to be composed of interleaving short-lived patterns that have different characteristics. This dynamic fluctuation makes it difficult for a single static predictor to achieve high prediction accuracy through the lifetime of a cloud application. As

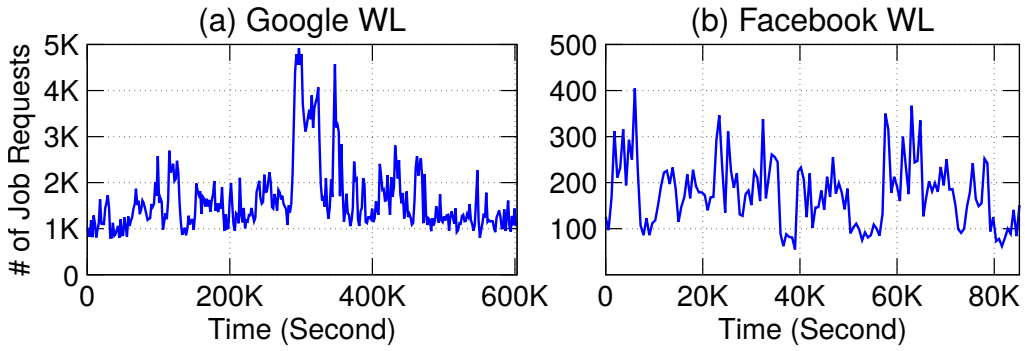


Figure 4.1: **Job arrival rate from cloud workload traces; (a) Google cluster trace [200] with 30 min. of time interval and (b) Facebook Hadoop trace [187] with 5 min. of time interval.**

Figure 4.1 shows, real workloads may not have clear and stable trend/seasonality. The lack of trend/seasonality implies that time-series models may not be the best choice for the cloud workloads in practice.

Consequently, a new approach is required to *improve the accuracy of workload prediction for real-world workloads that have a variety of workload patterns and dynamic fluctuations*. To this end, we have created the CloudInsight framework, inspired by a "mixture of experts" problem [90, 91]. Observing that different predictors excel at predicting different workload patterns, CloudInsight leverages the combined power of a diverse set of workload predictors. More specifically, CloudInsight dynamically creates an *ensemble model* that combines multiple predictors to predict the job arrival rate for the next time interval in the future. The weight of a predictor in the ensemble model is calculated at runtime based on the predictor's accuracy for the current workload at previous time intervals. To determine the weights, we design a novel evaluation approach based on a SVM (Support Vector Machine) multiclass regression model. The ensemble model is recreated periodically to handle the dynamic fluctuations in a workload. Since CloudInsight is an open-architecture, any different predictors from users' choice can be used in it.

We have conducted comprehensive evaluations of the performance of CloudInsight with diverse real-world workload traces collected from cluster [187, 200], HPC (High-Performance Computing) [86], and web applications [17]. The experiment results show that CloudInsight outperforms existing time-series, machine learning, and specific custom predictors in every workload. CloudInsight has 13% to 27% better prediction accuracy with low overhead. Moreover, we also perform a trace-based simulation in combination with a representative resource management module. The results

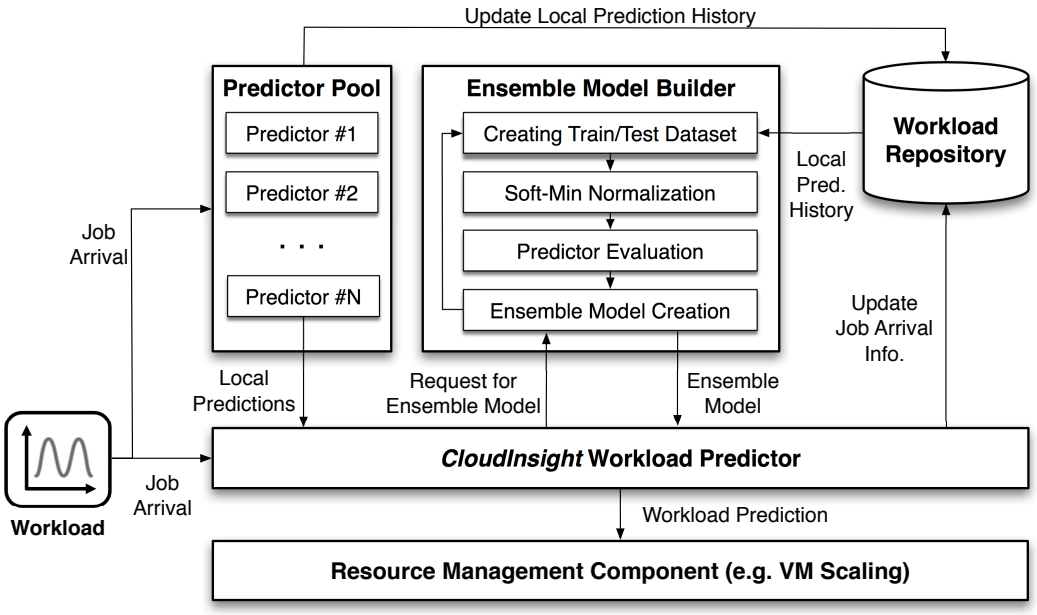


Figure 4.2: **Architecture of CloudInsight.**

from the simulation study show that CloudInsight incurs 15% – 20% less under-/over-provisioning, resulting in 16% better cost efficiency and 17% fewer SLA violations.

4.1.1 Chapter Organization

We structure the rest of the chapter as follows. We present the framework details and implementation of CloudInsight in Section 4.2. In Section 4.3, we evaluate CloudInsight with real-world workload traces. Section 4.4 provides sensitivity analysis of CloudInsight with more predictors. Lastly, Section 4.5 summarizes this chapter.

4.2 Approach: CloudInsight

This section gives a detailed description of CloudInsight. Figure 4.2 illustrates an overall architecture of CloudInsight. This framework consists of four main components: 1) a *predictor pool*, 2) a *workload repository*, 3) a *model builder* and 4) CloudInsight workload predictor. The input of this framework is the actual/current workloads (e.g., job arrivals) and the output is the prediction for a near-future workload. The *predictor pool* is a collection of workload predictors. The *workload repository* stores the job history of the workload and the prediction history of all local predictors in *predictor pool*. The *model builder* is responsible for creating an ensemble prediction

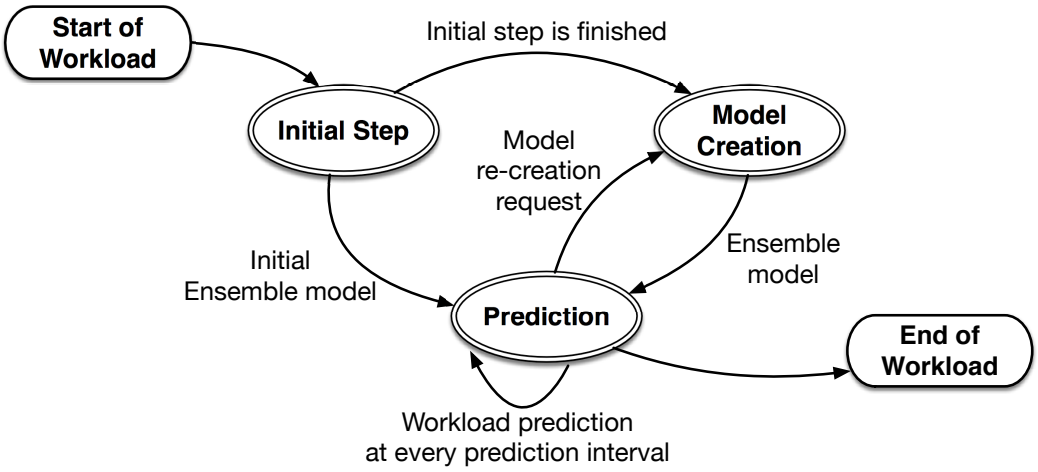


Figure 4.3: **Workflow of CloudInsight.**

model by evaluating the performance of the predictors in the *predictor pool*. CloudInsight workload predictor provides the forecast for the near-future workload using an ensemble model created by the *model builder*. This prediction will be utilized by resource managers for predictive resource (e.g., VM) scaling.

4.2.1 Workflow of CloudInsight

Figure 4.3 depicts the workflow of CloudInsight. The workflow of CloudInsight runs through the following steps; 1) initial prediction and measurement ("initial step" in Figure 4.3), 2) ensemble model creation ("model creation" in Figure 4.3), and 3) workload prediction ("prediction" in Figure 4.3). Once the first step is finished, CloudInsight repeats the second and the third steps until the end of workload.

CloudInsight has two temporal interval parameters: 1) a prediction interval and 2) a model re-creation interval. The former is the interval to make a new future workload prediction, and the latter is the interval to re-create the ensemble model used by CloudInsight. Users set both intervals when CloudInsight starts; normally the prediction interval is shorter than the model creation interval.

Initial Prediction and Measurement: when jobs begin to arrive in a cloud application, a prediction for the future workload is also initiated. But, for the initial period (e.g., the first 30 minutes or 1 hour), CloudInsight either uses a simple ensemble workload prediction model that all local predictors[1] have the equal contribution (weight)

[1]The local predictors indicate the predictors in *predictor pool*. Section 4.2.2 explains the local predictors in detail.

or relies on user's selection of the weights (the user can allocate a higher weight for a particular predictor). The use of simple ensemble model here is because, during the initial period, CloudInsight does not have enough accuracy history for the local predictors. It is impossible to meaningfully assign weights to the predictors without this history.

Ensemble Model Creation: once the initial (measurement) step finishes and initial accuracy history is collected, CloudInsight creates an ensemble prediction model based on the procedure described in Section 4.2.4. This ensemble model is used to predict future workload. After the expiration of the model re-creation interval, the ensemble model will be re-created.

Workload Prediction: the workload prediction is performed at every pre-defined prediction interval with the ensemble model, which is created from the previous step. The ensemble model combines the predictions from the local predictors in the *predictor pool*. This prediction can then be used by a resource management component for resource scaling.

4.2.2 Predictor Pool

The *predictor pool* contains a variety of workload predictors, called "local predictors." CloudInsight is designed to be generic and does not have any special dependency with a particular predictor. Consequently, its *predictor pool* can contain any predictors as long as those predictors can provide predictions for future workloads (e.g., job arrival rates). We have experimented with various workload predictors, including time-series, regressions, and machine-learning models. As shown later in Section 4.3, CloudInsight can properly handle all these types of predictors and considerably improve workload prediction accuracy. Other types of workload predictor (e.g., Neural Network [89], Markovian Arrival Process [64, 181], Bayesian Model [46]) can also be used for the *predictor pool*. Because of CloudInsight's generality, users can add any workload predictors to the *predictor pool*. Note that, more local predictors may increase the overhead of workload prediction and ensemble model creation. Although CloudInsight's overhead is negligible in our evaluation with 21 predictors, users may want to limit the size of *predictor pool* when there are hundreds of potential local predictors and the overhead becomes non-trivial.

When actual workload comes to our target applications, all local predictors in the *predictor pool* make their predictions for future job arrival rates. However, because

different local predictor works best for different workload patterns (or a particular part of dynamic real-world workloads), the *model builder* would selectively consider the best ones and combine them with different contributions (weights). The *model builder* is responsible for properly selecting and combining them. We will describe how we address this issue.

4.2.3 Workload Repository

Workload repository contains the prediction history of the all local predictors in the *predictor pool*. This history is represented as a normalized performance vector, which is described in the following paragraphs.

Performance Vector (PV): the PV is a fundamental element of training and prediction input datasets for the evaluation step and is a feature matrix composed of prediction errors of all local predictors for past prediction history. The performance vector is an $n \times m$ matrix as formulated below:

$$PV = \begin{bmatrix} PE_{1,1} & PE_{1,2} & \cdots & PE_{1,m-1} & PE_{1,m} \\ PE_{2,1} & PE_{2,2} & \cdots & PE_{2,m-1} & PE_{2,m} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ PE_{n-1,1} & PE_{n-1,2} & \cdots & PE_{n-1,m-1} & PE_{n-1,m} \\ PE_{n,1} & PE_{n,2} & \cdots & PE_{n,m-1} & PE_{n,m} \end{bmatrix} \tag{4.1}$$

, where n is the number of the local predictors and m is the past (consecutive) prediction points. $PE_{i,j}$, an element of PV matrix, is the prediction error of ith local predictor at jth prediction point. $PE_{i,j}$ is measured by squared error $((Prediction_{i,j} - Actual_{i,j})^2)$. A single PV represents a set of prediction errors of all n local predictors for past m prediction points. (In our evaluation, CloudInsight uses 50 for m of PV. This configuration works well for our case.)

Soft-Min Normalization: an issue of PV is that each element (PE) of the PV matrix is the absolute squared error of each local predictor at certain prediction point. Since a PE represents the absolute prediction errors at a particular time interval, two PEs from two different time intervals cannot be directly compared to determine which is more accurate (i.e., which has smaller error). Therefore, we normalize PEs so that they can be directly compared. To normalize all PEs in a PV, we use a soft-min normalization function that transforms each element (PE) into a real number

between 0 and 1. The soft-min function is shown in Equation (4.2).

$$Soft - Min(PE_{i,j}) = 1 - \frac{e^{-PE_{i,j}}}{\sum_{k=1}^{n} e^{-PE_{k,j}}} \tag{4.2}$$

The input of Equation (4.2) is an element ($PE_{i,j}$) of PV. The numerator of the function is the exponentially inverse transform of the PE that we want to normalize. The dominator is the sum of exponentially inverse transforms for all PEs at a particular prediction point (a single column in the PV.) Also, this normalized value is subtracted from 1 so that higher values mean better performance (smaller prediction errors) of the local predictors. The upper bound of the normalized soft-min value is 1, while the lower bound is 0. A local predictor always has a soft-min value between 0 and 1. After this normalization, the sum of each column in a PV is 1. Intuitively, the soft-min value for a local predictor at particular prediction point can be viewed similarly as the probability of it being the best predictor for this prediction point.

4.2.4 Ensemble Predictor Builder

The *model builder* evaluates the local predictors, determines the best predictors among them, and creates an ensemble prediction model of top predictors with different weights. This *model builder* is inspired by a mixture of experts (MOE) problems [90, 91]. The essential insight of the MOE is that a collective result (e.g., ensemble) of all local experts is often better than a decision from a single expert [83]. In our case, each local predictor can be considered as a local expert in the MOE problem. Unlike a general MOE approach, which leverages a simple linear combination of all local experts, we create an ensemble model combined by the local predictors with different weights, because different local predictors work best for different workload patterns (or a particular part of dynamic real-world workloads). A higher weight of a local predictor indicates better performance for the current workload, and thus potentially better for the future.

Evaluating the local predictor is the most important step of the *model builder* to create an ensemble prediction model. We formulate this evaluation as a multiclass regression problem and use Gaussian SVM regression model [72]. The following paragraphs give a detailed description of how the ensemble model is trained and used to make workload predictions.

Training Dataset and Prediction Inputs: both training dataset and the predic-

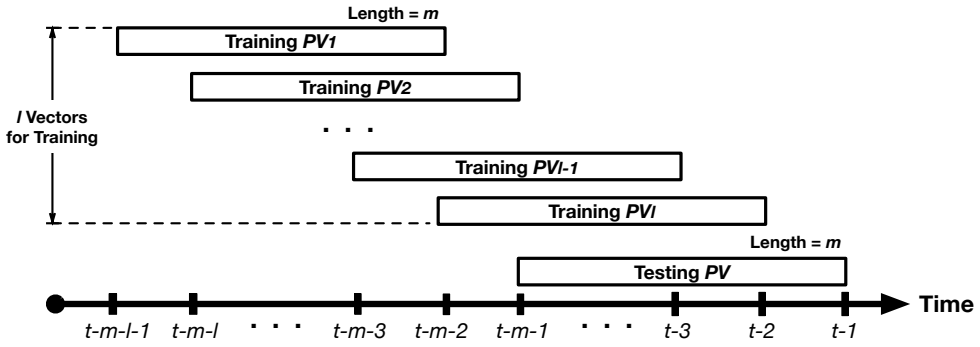


Figure 4.4: **Temporal coverage of PVs for training and prediction input dataset. (m: length of row for PV, meaning m temporal points for past predictions, l is the size of training dataset, meaning the number of PVs in training dataset.**

tion input dataset are represented as a collection of PVs as discussed in Section 4.2.3. However, these two datasets use separable PVs that cover different temporal windows. Suppose time t indicates current prediction point, l is the size of training dataset, and m means the length of columns in PVs. The training dataset covers the history of the local predictors' performance between at $t-m-l-1$ and $t-2$. The training dataset is expressed as $\{PV_{t-m-l-1}, PV_{t-m-l}, ..., PV_{t-3}, PV_{t-2}\}$. The prediction input data-set, which is used to predict the job arrival rate at time t, is the PV at $t-1$ prediction point and is expressed as $\{PV_{t-1}\}$. Figure 4.4 illustrates the temporal coverage of training data set and prediction input data set.

Evaluation of Local Predictors: as we mentioned previously, we reduce the "evaluating local predictors" problem to the "multiclass regression" problem. A multiclass regression problem gives the probabilities of whether an observation belongs to a set of categories. Consequently, with a "multiclass regression" model, we can evaluate the probability that a local predictor is the most accurate predictor for the future workload. More specifically, we employ Gaussian SVM model for this classification problem. The evaluation with the SVM model follows a typical machine learning process; training and prediction. The SVM model is trained with the aforementioned training dataset. After training, this model can provide its projection for all local predictors. The output vector of this model is shown below.

$$Y = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_{n-1} \\ \omega_n \end{pmatrix} \tag{4.3}$$

The output of this SVM model is a $n \times 1$ matrix. Thanks to the soft-min normalization, all items in this output matrix are real numbers (ω) between 0 and 1. A higher value of ω_i (close to 1) indicates that ith predictor has higher possibility to be the best predictor for workload prediction. Likewise, lower value of ω_j (close to 0) suggests that jth predictor has a lower probability of being the best.

Creating an Ensemble Model for Workload Prediction: the ensemble workload predictor directly uses the output from the evaluation results of the local workload predictors. This ensemble model is constructed with Equation (4.4).

$$\text{Ensemble Model: } \frac{\sum_{i=1}^{n} \omega_i p_i}{\sum_{i=1}^{n} \omega_i} \tag{4.4}$$

ω_i is the output from the previous step and p_i is the prediction from a local predictor. By employing ω_i, this ensemble model gives higher weights to potentially more accurate local predictors and lowers weights to potentially less accurate local predictors. The result from this ensemble model is the prediction of the current and near-future job arrive rates, which can be utilized by other resource management components (e.g., VM scaling). Also, the predictions of the local predictors used in this ensemble model will be updated to the *workload repository* to be used for further evaluation of the local predictors. Note that the ensemble model is not necessarily created at every prediction point since this model requires the entire process of evaluating the local predictors, which is time-consuming. The ensemble model will be re-created periodically with a predefined time interval. In our evaluation (Section 4.3), we recreate the ensemble model after finishing every five predictions, which works well for our case.

4.2.5 Implementation of CloudInsight

We implemented CloudInsight with Python 2.7 on Ubuntu 16.04 LTS. To implement the local predictors in the *predictor pool* and the evaluation step of the *model builder*, the following statistics and machine learning libraries are used; NumPy, Statsmodels,

Pandas, and scikit-learn.

For implementing the local predictors, while our goal is to improve/maximize the prediction accuracy, deterministic processing time of the local predictors is desirable. This requirement is because CloudInsight collaborates with a resource manager that should adequately prepare cloud resources before the actual job arrives. We use a grid search [20] to determine the parameters for the local predictors with a tradeoff between the accuracy and the prediction overhead. We consider parameters of $0 < \alpha < 1$ for BDES, 1st to 3rd order for other time-series models (AR, ARMA, ARIMA) and $10e^{-3}$ to $10e^{3}$ for soft margin and kernel parameters in SVMs.

For the implementation of the ensemble predictor builder (the SVM multi-regression model), we aim more at improving the performance of an ensemble model. To this end, we also take the same approach (grid search) with the way of tuning the local predictors, but we examine broader range for soft margin and kernel parameters of SVM model $10e^{-6}$ to $10e^{6}$ to yield better results.

We use various synthetic workloads [58, 193] to guide the above two parameter selection processes. To ensure fair evaluation and avoid over-fitting, we did not use real workloads in parameter selection. Real workloads [17, 34, 86, 169] are only used to evaluate CloudInsight.

4.3 Performance Evaluation

4.3.1 Evaluation Setup

Workload Datasets: to evaluation CloudInsight, we used three categories of workload traces from real-world cloud applications: 1) Cluster workload traces from Google [200] and Facebook [187], 2) Scientific/HPC workloads from the Grid Workloads Archive [42] and 3) Wikipedia web traces from WikiBench [199]. These three groups of workload datasets allow us to evaluate CloudInsight with diverse scenarios of application deployment on clouds. We provide details on the characteristics of these workloads in Section 4.3.2.

Local Predictors: in this evaluation, the *predictor pool* has *eight* well-known workload predictors; linear regression [22, 125, 147, 162, 210], WMA [61, 113, 164], BDES [21, 138, 154], AR [30, 31, 49, 202], ARMA [55, 100, 161, 174, 190], ARIMA [28, 47, 213], and two SVMs (both linear and Gaussian models) [18, 153, 207]. These eight predictors are chosen because they have shown decent performance for four different

simulated workloads from the measurement in Chapter 3. These predictors are described in Section 3.2.

Evaluation Goals: our goal is to evaluate four properties of *Cloud-Insight*. Firstly, we measure the accuracy of CloudInsight regarding the forecasting future job arrival rate in above datasets (Section 4.3.3). Secondly, we evaluate the overhead of CloudInsight since prediction within deterministic time is a prerequisite of any workload predictors (Section 4.3.4). We then will perform a trace-based simulation of CloudInsight in combination with resource scaling components to measure how much benefit it can bring to cloud resource management (Section 4.3.5).

Performance Metrics: to measure the prediction accuracy of job arrive rate, we employ RMSE (Root Mean Square Error), which is formulated as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Predicted_i - Actual_i)^2} \tag{4.5}$$

Once we obtain RMSE results, all results are normalized over the result from CloudInsight to see the relative differences clearly. 1.0 means the results from CloudInsight. Lower RMSE results mean the better performance and vice versa.

To evaluate the overhead of CloudInsight, we define the processing overhead as the *time* for "job arrival rate prediction process" and "ensemble model recreation process." We measure the actual processing time on a Linux Server with 8 CPUs (AMD Opteron Processor 4386) and 16G RAM.

For the evaluation with a representative resource management module, we employ general cloud metrics, including cloud cost, job deadline miss rate, and over-/under-provisioning time.

Baselines: we compare CloudInsight against four predictors; ARIMA, SVM, FFT (Fast Fourier Transform), and RSLR (Robust Stepwise Linear Regression). We choose ARIMA and SVM from the local predictors because they are widely used in many predictive approaches [18, 28, 47, 153, 207] as well as the two best "*one-size-fits-all*" predictors that we have experimented with. We choose FFT [64, 181] and RSLR [203] from state-of-the-art approaches, which provide a robust and accurate prediction for cloud resource scaling.

4.3.2 Evaluated Workloads

While the workload datasets contain various characteristics, this chapter focuses on its temporal characteristics. i.e., job arrival rate. We extract temporal behaviors of job submissions in the workloads and Table 4.1 describes the summary of the characteristics of the workloads. We also choose workloads with variable length of duration (lifetime) and density of job arrivals to show the generality of CloudInsight. The following paragraphs outline the backgrounds of such workloads.

Table 4.1: **Statistics of evaluated workloads.**

	Workload	**Duration**	**# Jobs**	**Predictor Setting**	
				Prediction Interval	**Model Recreat. Interval**
Cluster	Google	1 month	2M	30 to 1200 sec.	At every 5 Predictions
	Facebook #1	1 day	5.9K		
	Facebook #2		6.6K		
	Facebook #3		24K		
	Facebook #4		25K		
Web	Wiki Global	3 days	823K	30 to 1200 sec.	At every 5 Predictions
	Wiki Germany		76.5K		
	Wiki Japan		51K		
HPC	Grid 5000	22 days	62.5K	30 to 1200 sec. or 1 to 12 hrs (AuverGrid)	At every 5 Predictions
	NorduGrid	60 days	122K		
	AuverGrid	365 days	2.3M		
	SHARCNet	11 days	188K		
	LCG	33 days	435K		

Cluster Workloads: these workloads represent the behaviors of cloud applications for cluster and big data analytics (e.g., Hadoop). We use several workloads from Google Cluster trace [200] and Facebook Hadoop traces [187]. Google workload contains 2 millions of job arrival data for a one-month period. Facebook dataset contains 1 millions of job submissions. Particularly for the Facebook workload, we use 4 sample traces (two traces from 2009 and two traces from 2010), each represents 1-day job submissions. The examples of visualizing such workloads are shown in Figure 4.1 in Section 4.1. We only show two of cluster workload, but other three workloads from Facebook have similar characteristics with Figure 4.1(b). The cluster workloads are varying significantly and have dynamic nature of job submissions.

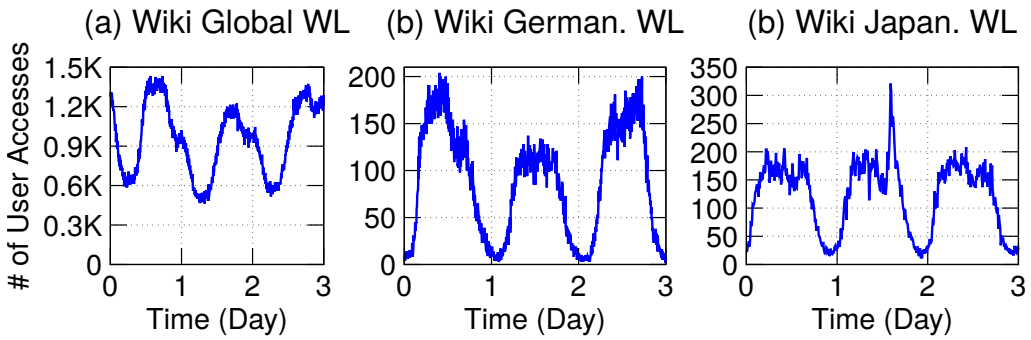


Figure 4.5: **Wikipedia traces with 5 min. of time interval.**

Web Workloads: these workloads represent the behaviors of web applications, which is a common application model on clouds. We use three days of Wikipedia traces in September 2007. We focus on access log for (global) Wikipedia pages[2], German[3], and Japanese main page[4] of Wikipedia. The datasets have 823K (global), 76.5K (German), and 51K (Japanese) of user accesses. These Wikipedia workloads are illustrated in Figure 4.5. Wikipedia workloads generally show strong seasonality and trend characteristics, but Japanese main page has an unexpected spike of user access.

HPC Workloads: the clouds are also actively used by many HPC applications to support diverse areas of scientific computing. Five workloads are used for HPC scenarios on the clouds. i.e., Grid5000[5], NorduGrid[6], AuverGrid[7], SHARCNet[8], and LCG (LHC Computing Grid)[9]. These workloads respectively contain 62.5K jobs, 122K jobs, 2.3-million jobs, 188K jobs, and 435K jobs for various periods. These workloads are illustrated in Figure 4.6. HPC workloads have similar characteristics with two previous workloads but have more dynamic natures.

4.3.3 Prediction Accuracy of CloudInsight

To compare the accuracy of CloudInsight, with the baselines, we use various time interval for the workload prediction as shown in Table 4.1. This prediction interval

[2]https://www.wikipedia.org
[3]http://de.wikipedia.org
[4]http://jp.wikipedia.org
[5]https://www.grid5000.fr
[6]http://www.nordugrid.org
[7]http://www.auvergrid.fr
[8]https://www.sharcnet.ca
[9]http://wlcg.web.cern.ch

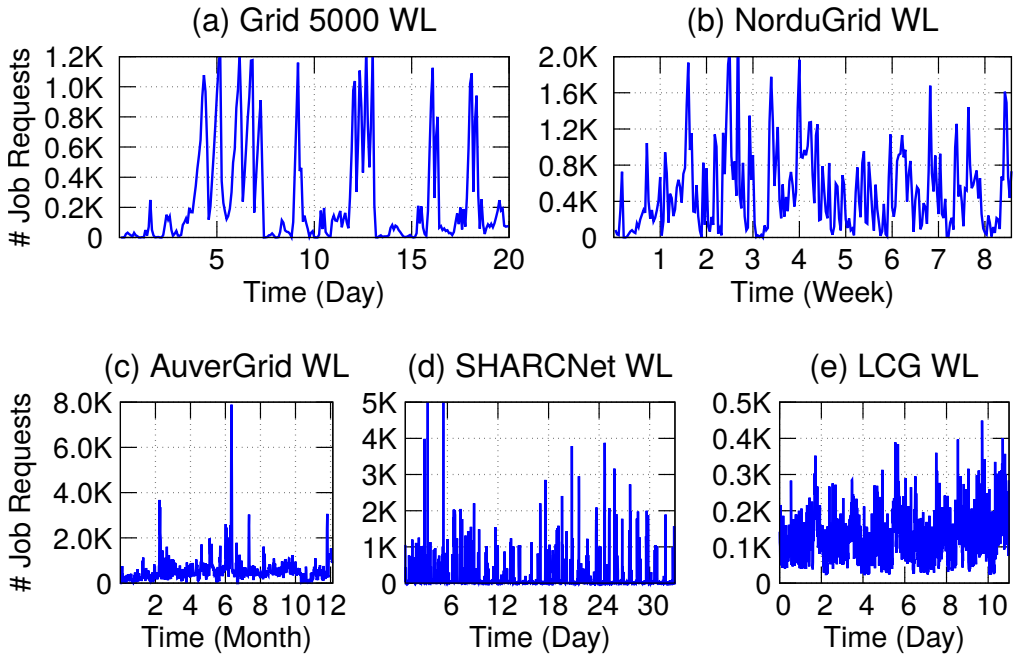
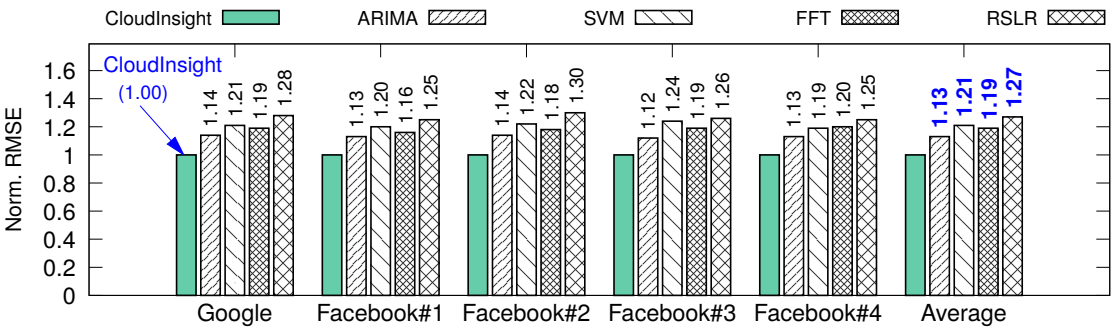


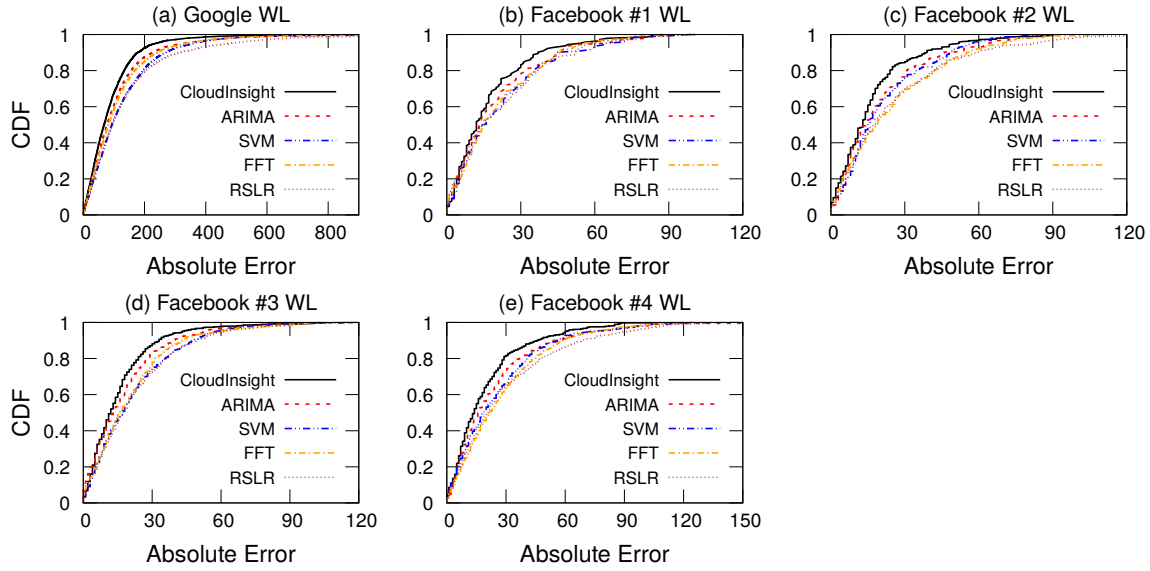
Figure 4.6: **HPC workload traces. (Grid 500 with 3600 sec. of time interval, NorduGrid with 7200 of time interval. AuverGrid with 12 hrs. of time interval, LCG with 600 sec. of time interval and SHARCNet with 900 sec. of time interval.)**

may affect its prediction accuracy because a longer prediction interval may provide a smoothing effect on the workload patterns. Evaluation with the various time interval minimizes this impact and averaging them offsets the variation. In general, we use the prediction intervals from 30 seconds to 1200 seconds with a step of 30 seconds. Especially for AuverGrid with one-year period, we use the prediction intervals with a range from 3600 seconds (1 hour) to 86400 (24 hours) with a step of 3600 seconds. For the model re-creation interval, we recreate and update the SVM model for evaluating the local predictors at every five predictions of future workloads.

Cluster workload: the RMSE results of job arrival rate predictions of the five approaches are shown in Figure 4.7(a) (all results are normalized to CloudInsight). Overall, CloudInsight is 13% – 27% more accurate than the four baselines. Because the cluster workloads do not have a stable seasonality and a trend, it is difficult for a single model (ARIMA, SVM, FFT, or RSLR) to accurately detect certain patterns from the cluster workloads to predict future changes. However, CloudInsight can keep adjusting the weights for each predictor and create new ensemble model periodically to

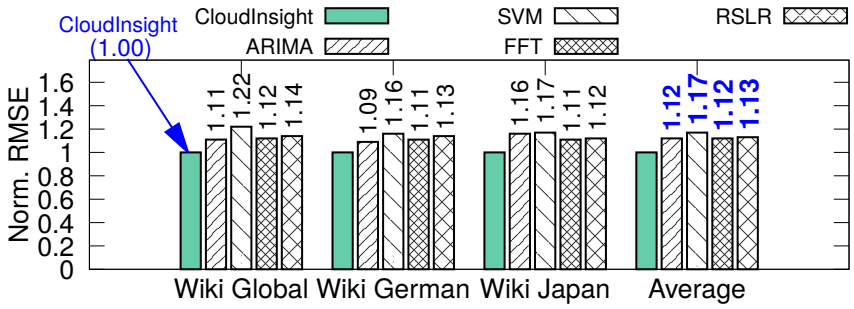


(a) Normalized RMSE

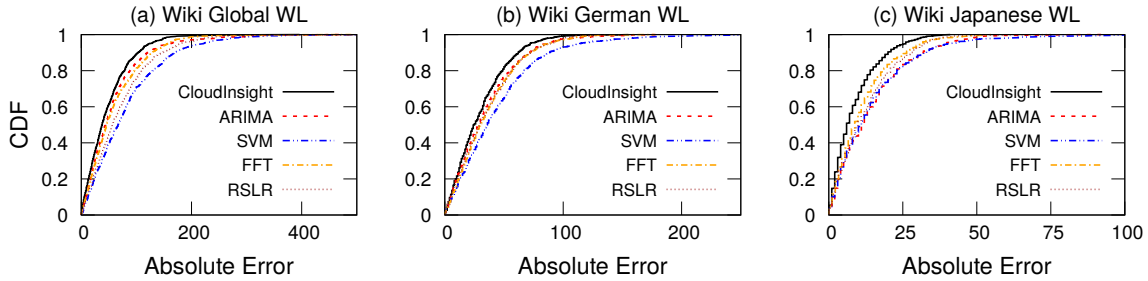


(b) CDF of prediction errors

Figure 4.7: **Prediction accuracy in cluster workloads. Normalized RMSE results in cluster workloads. (1.00 means the result from CloudInsight and higher values indicate worse performance.), (b) CDF of prediction errors in cluster workloads. (Absolute Error** $= |Prediction_t - Actual_t|$**)**



(a) Normalized RMSE



(b) CDF of prediction errors

Figure 4.8: **Normalized RMSE results and CDF of prediction errors in web workloads.**

fit the changes in the workload. Therefore, CloudInsight can show better performance for workload prediction.

Figure 4.7(b) shows the CDF (Cumulative Distribution Function) of prediction errors in cluster workloads. The x-axis represents absolute prediction error, calculated by $|Prediction_t - Actual_t|$, while the y-axis gives the cumulated probability of the errors. As Figure 4.7(b) shows, the curves for CloudInsight are skewed to the left than the baselines, meaning the majority of CloudInsight's prediction errors are smaller than the baselines. Also, the results from the baselines have longer tails, indicating they yield more extreme prediction errors.

Web workload: the RMSE results for web workloads are shown in Figure 4.8-(a) and CloudInsight outperforms the baselines again. On average, CloudInsight has 12% – 17% of fewer errors than the baselines. Because web workloads usually have strong seasonality and trends, all baselines perform better than when predicting cluster workloads. Especially, both FFT and LRSR have a significant improvement in their accuracy. However, although web workloads have relatively stable seasonality and trends, the seasonality and trends can still change over time, albeit less abruptly.

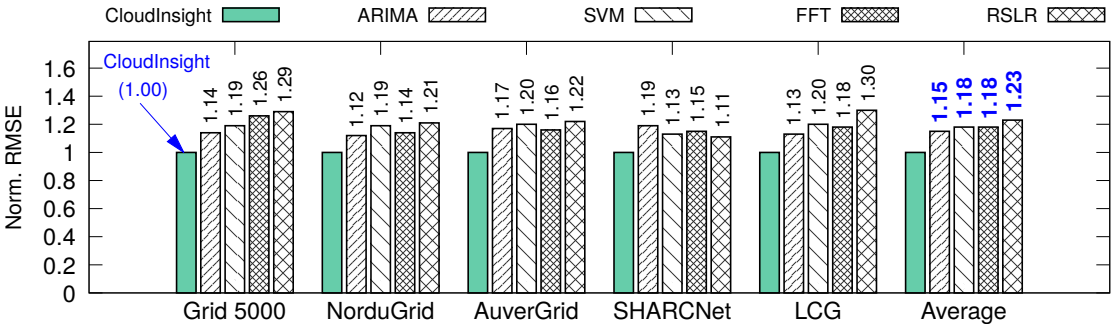
Also, as shown in Wiki Japanese workload, web workload could have a sudden spike of user accesses. CloudInsight can identify the seasonality and trends as well as detect the changes (or spikes) in them. Thus, it can provide better prediction results.

Figure 4.9(b) shows the CDF distribution of the prediction errors in the HPC workloads. The results are similar with the two previous cluster and web workload types. The curves for CloudInsight are more skewed to the left, indicating that CloudInsight has less errors than other baselines.

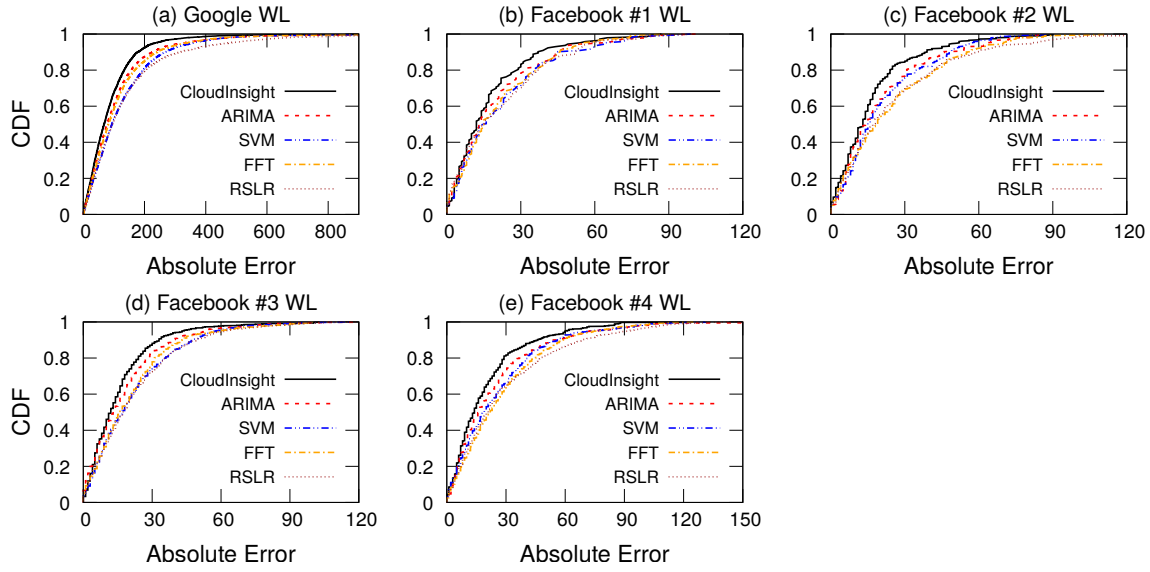
Figure 4.8-(b) illustrates the CDF of prediction errors in web workloads. Similarly, the majority of CloudInsight's errors are still smaller than those of the baselines.

HPC workload: Figure 4.9(a) shows the normalized RMSE results in the five HPC workloads; Grid 5000, NorduGrid, AuverGrid, LCG, and SHARCnet. On average, CloudInsight is 19% more accurate than the four baselines with all the five HPC workloads. Our HPC workloads exhibit a broad range of characteristics. The Grid 5000, AuverGrid, and SHARCNet workloads are bursty and random. They also lack seasonality and trends. The NorduGrid and LCG workloads have relatively clearer seasonality (among HPC workloads), although it is much more bursty and noisier than web workloads. The measurement results with various HPC workloads indicate that CloudInsight can correctly assign weights to the local predictors based on current workloads behaviors so that it can predict with the best predictors for the future workload.

Summary: we evaluated the prediction accuracy and the (CDF) error distributions of three approaches in three different real-world workloads. In all workloads, CloudInsight shows 4% to 20% of better accuracy than ARIMA and SVM. The CDF distribution of prediction errors clearly show the errors from CloudInsight are more skewed to the left, meaning that it generates smaller errors and closer predictions to the real workload changes.



(a) Normalized RMSE



(b) CDF of prediction errors

Figure 4.9: **Normalized RMSE results and CDF of prediction errors in HPC workloads.**

4.3.4 Overhead of CloudInsight

Forecasting future workloads within a short and deterministic amount of time is also a very critical property for all predictive resource scaling systems. Here, we evaluate the overhead of CloudInsight and the four baselines. For the baselines, we only consider "prediction overhead" for this evaluation as there is no additional overhead due to the ensemble model reconstruction for them. We define "prediction overhead" as the time that it takes to make predictions at given time point. For CloudInsight, we measure both "prediction overhead" and "modeling overhead." We define the modeling overhead as the time that CloudInsight takes to create a new ensemble prediction model.

Table 4.2: **Prediction overhead of five approaches.**

	Cluster Workload	Web Workload	HPC Workload	**Average**
CloudInsight	29 ms	37 ms	36 ms	**34** ms
ARIMA	25 ms	24 ms	29 ms	**26** ms
SVM	0.35 ms	0.4 ms	0.4 ms	**0.38** ms
FFT	4.2 ms	5.9 ms	8.8 ms	**6.3** ms
RSLR	22 ms	18 ms	22 ms	**21** ms

Prediction Overhead: Table 4.2 gives the average prediction overhead for all the approaches. On average, CloudInsight takes 34ms to make a prediction, while other methods show lower prediction overhead. i.e., ARIMA takes 26ms, RSLR takes 21ms, FFT takes, 6.3ms and SVM takes 0.38ms. Although SVM has the lowest overhead among the approaches, it has less accuracy than CloudInsight and the others for the majority of our workloads. Even though CloudInsight leverages eight local predictors, it takes only 12ms more time as compared to ARIMA. The reason is that these eight predictors compute the prediction in parallel. The prediction overhead of CloudInsight is determined by the highest prediction time of its eight local predictors. It is worth noting that although CloudInsight has the longest prediction time among five predictors, the absolute prediction time (34ms) is still *negligible* compared to workload prediction intervals and resource reconfiguration intervals. Because the overhead from cloud infrastructure is usually higher than 30 seconds (e.g., VM startup time [130]), autoscaling resource managers often reconfigure their resources at an

interval higher than 30 seconds. As CloudInsight's prediction time is much smaller than the prediction interval, it imposes very limited impact to an autoscaling resource manager.

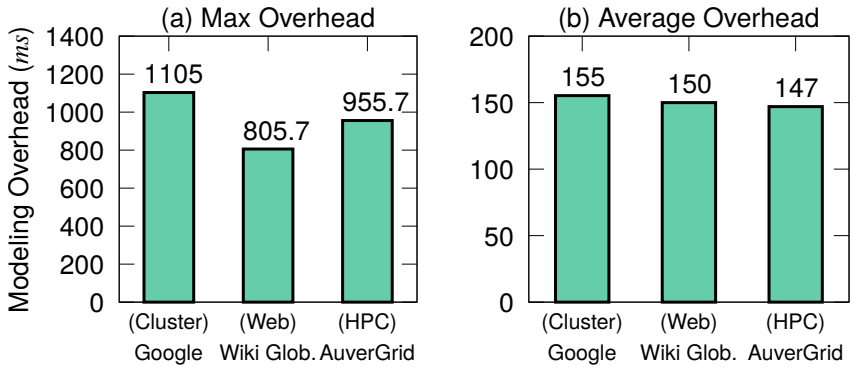


Figure 4.10: **Modeling overhead of CloudInsight.**

Ensemble Modeling Overhead: the ensemble modeling overhead of CloudInsight is defined as the time to create a new ensemble model. The results of this overhead are shown in Figure 4.10. Limited by space, we only show the results from the largest workload for each type of workloads: Google, Wiki Global Main, and AuverGrid. This overhead for the other workloads is usually smaller than these three workloads. In the worst cases, it takes 0.8-1.1 seconds to create a new ensemble model. The average modeling time is less than $155ms$. This overhead is still negligible in practice because CloudInsight can create a new ensemble model and make predictions within the autoscaling resource reconfiguration intervals as stated previously.

Summary: we evaluated the overhead of CloudInsight with diverse workloads. CloudInsight has a higher overhead for both prediction and modeling as compared to the baselines. However, CloudInsight can provide prediction within 100 ms and create a new ensemble model within about 1.1 seconds. This overhead is tolerable by any predictive scaling since it has longer prediction interval, and this overhead can be compensated by the accuracy that CloudInsight shows.

4.3.5 Case Study: Predictive Resource Management

To evaluate the benefits brought by CloudInsight to cloud resource management, we conducted additional experiments where CloudInsight is applied to an elastic resource manager. We also applied the baselines to the same resource manager as a comparison.

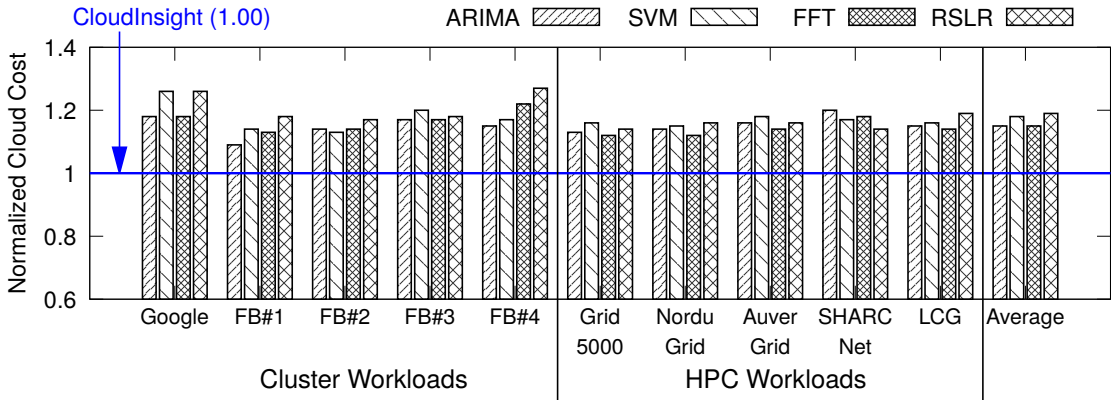


Figure 4.11: **Cost usage of the resource management with five approaches.**

All these experiments are conducted on a trace-based simulator that closely resembles the behaviors of real cloud applications on public clouds. Our simulation model is composed of three sub-components; workload generator, resource manager, and cloud infrastructure (public cloud). The workload generator creates user requests (job arrivals) according to the workload traces (e.g., cluster and HPC workloads) and uses the same configuration with Table 4.1. Each job is also associated with a randomly generated deadline. The resource manager is responsible for predictive scaling-out and reactive scaling-in to handle the workloads. The workload predictors (e.g., CloudInsight, the four baselines) are part of this resource manager. For the scaling-out operation, the workload predictor forecasts the future job arrival rate, and the manager adds more resources if the demand is higher than the currently available VM resources. For the scaling-in operation, the manager checks the status of VMs in every minute and terminates the VM if the VM has no further jobs to process. For the public cloud setting, we use an on-demand homogeneous VM type for the simplicity of this evaluation and use minute-based pricing model for the cost calculation.

We used three metrics to evaluate the effectiveness of the resource manager; 1) cloud usage cost, 2) job deadline miss rate, and 3) under-/over-provisioning time. A higher cloud usage cost means that the VMs are more likely to be over-provisioned (lower cost efficiency). A higher job deadline miss rate suggests that the VMs are more like to be under-provisioned (more SLA violations). Figure 4.11 and 4.12 give the costs and job deadline miss rates of all approaches under cluster and HPC workloads. The results show that the resource manager using CloudInsight has the lowest cost and lowest job deadline miss rate. On average, CloudInsight shows 14% – 22%

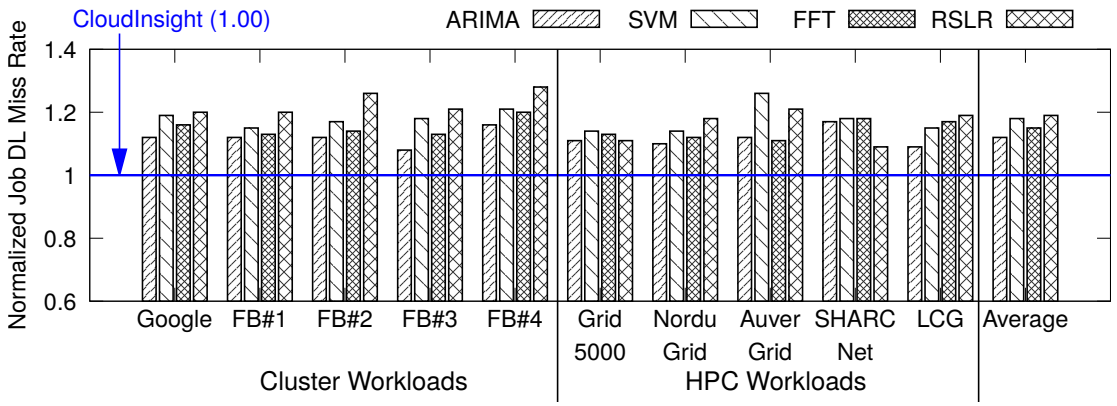


Figure 4.12: **Job deadline miss rate of the resource management with five approaches.**

better cost efficiency and 12% – 21% less job deadline miss rate as compared to the baselines. Table 4.3 shows the normalized under-/over-provisioning time of baselines. These accumulate both under-/over-provisioning periods and are normalized over the results from CloudInsight. As the results show, CloudInsight has 15% – 19% less under-/over-provisioning times compared to the baselines. These results indicate that as CloudInsight increases the accuracy of future job arrival rate predictions, it can significantly reduce the under-/over-provisioning of the cloud resources. The low under- and over-provisioning periods also suggest that the resource manager with CloudInsight always provisions the proper and accurate amount of VM resources according to the actual workload changes while retaining high SLA satisfaction rate and cost efficiency.

4.4 Sensitivity Analysis of CloudInsight with More Predictors

In the previous evaluations, we only use eight local predictors to measure the performance of CloudInsight. However, the group of local predictors may affect the accuracy of CloudInsight. As CloudInsight has no limitation in leveraging the number of predictors, we perform a sensitivity analysis of CloudInsight with more number of local predictors to confirm how this change affects its performance.

To evaluate the impact of the composition of local predictors, we measured the accuracy and overhead of CloudInsight by adding 13 more local predictors (in total 21 local predictors) that are mentioned in Section 3.2. As shown in Figure 3.3 and

Table 4.3: Normalized sum of under-/over-provisioning time.

	ARIMA	**SVM**	**FFT**	**RSLR**
Google	18%	27%	18%	32%
Facebook #1	9%	14%	13%	18%
Facebook #2	14%	13%	14%	17%
Facebook #3	17%	20%	17%	18%
Facebook #4	15%	17%	22%	27%
Grid 5000	13%	16%	12%	14%
NorduGrid	14%	15%	12%	16%
AuverGrid	16%	18%	14%	16%
SHARCNet	20%	17%	18%	14%
LCG	15%	16%	14%	19%
Average	**15%**	**18%**	**15%**	**19%**

Table 3.2 in Section 3.4, these 13 predictors have lower average accuracy than the eight predictors used by CloudInsight in the previous evaluations. In practice, it may be difficult for a user to know which predictors are better than the others for his/her workload. Therefore, a user may be inclined to include all known predictors, many of which could have low accuracy. By including more predictors, this evaluation aims at determining whether CloudInsight can still correctly pick the best predictors from a large pool of mixed predictors. The normalized accuracy of using 21 predictors is reported in Figure 4.13. CloudInsight with 21 predictors has similar or even better accuracy than using eight predictors. Further analysis of the results shows that the *model builder* of CloudInsight can correctly evaluate the accuracy of local predictors and correctly assign higher weights for good predictors and lower weights for poor predictors. Furthermore, for certain parts of a workload, a predictor from the additional 13 predictors may perform better than any of the eight averagely-best predictors. The better accuracy in web and HPC workloads indicates that CloudInsight can correctly catch the accuracy benefits from the additional 13 predictors for certain parts of the workloads and utilize their better accuracy to improve overall prediction results. These results also show that CloudInsight is a generic framework that can be used with a myriad of types of predictors.

While using more predictors does not hurt (sometimes even improves) prediction accuracy, using more predictors increases the prediction overhead of CloudInsight. With 21 predictors, CloudInsight was 3.85x slower than only using eight predictors. We observe that most overhead comes from re-creating the ensemble model: more

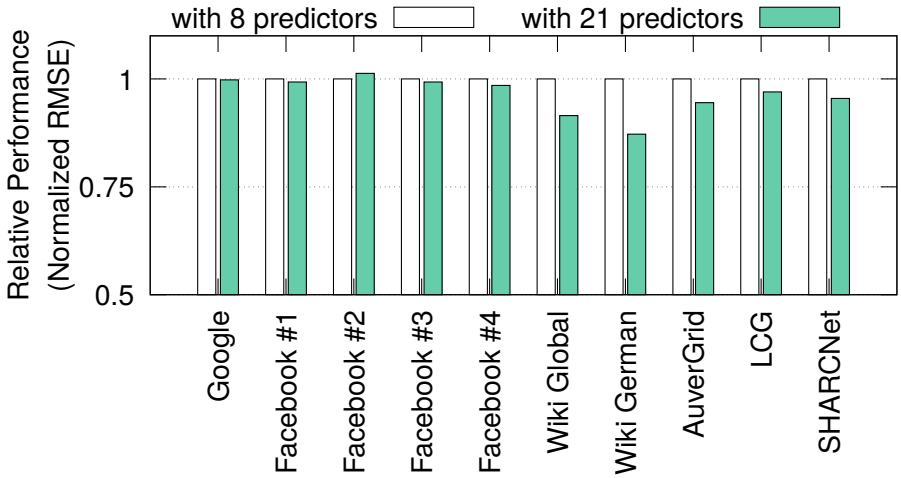


Figure 4.13: **Result of sensitivity analysis of CloudInsight with the more number of local predictors. Note that lower values mean better accuracy and vice versa.**

local predictors exponentially increase the overhead for model re-creation. It is worth noting that even though using 21 predictors is slower, the absolute time for prediction and model re-creation is still much smaller than the prediction and model re-creation intervals (or resource reconfiguration interval). Therefore, the overhead of 21 predictors has limited impact in practical usage. However, if a user has a much larger set of potential local predictors, he/she may want to limit the number of predictors employed to reduce overhead. In the future, we will investigate how to determine the optimal types/numbers of local predictors for CloudInsight.

4.5 Chapter Summary

This chapter presents CloudInsight, which is an online workload prediction framework to address dynamic and highly variable cloud workloads. CloudInsight employs a number of local predictors and creates an ensemble prediction model with them by dynamically determining the proper weights (contributions) of each local predictor. To determine the weights, we formulate this problem as a multi-class regression problem with a SVM classifier.

We have performed a comprehensive study to measure the performance and overhead of this framework with a broad range of real-world cloud workloads (e.g., cluster, web, and HPC workloads). Our evaluation results show that CloudInsight has 13% – 27% of better accuracy than state-of-the-art *one-size-fits-all* style predictors

and it also has low overhead for predicting future workload changes ($< 100ms$) and (re)creating a new ensemble model ($< 1.1sec.$).

In summary, the mechanism and evaluation results of CloudInsight show that our approach is capable of addressing real-world cloud workloads that have dynamic and high variable nature. This chapter will help other cloud researchers and practitioners design a new predictive method for managing and scaling cloud resources autonomously.

Chapter 5

Orchestra: Guaranteeing Performance SLA for Cloud Applications by Avoiding Resource Storms

Chapter 3 and 4 solve the workload uncertainty in cloud computing by CloudInsight, which is a multi-predictor-based online prediction framework. In this chapter, we are focused on addressing the performance uncertainty with user-space application control framework. We begin this chapter by investigating the resource storm problem that causes the performance uncertainty in enterprise cloud environments. Then, we design an accurate performance model for real-world applications and develop an application control framework for modern micro-service architecture that ensures application performance under the performance uncertainty.

5.1 Introduction

In modern data centers, enterprise cloud instances (i.e., VM) are not only serving user-facing (foreground or FG) applications, but also running diverse types of background (BG) services[1] – *backup*, *security compliance*, *virus scan*, *patching*, and *batch* tasks – in order to securely and reliably manage such instances, and improve overall resource utilization/cost efficiency. Since the BG services frequently perform very critical missions for the management purposes, they have to be executed as planned in many

[1]In this chapter, we use the term "application," "task," and "service" *interchangeably*.

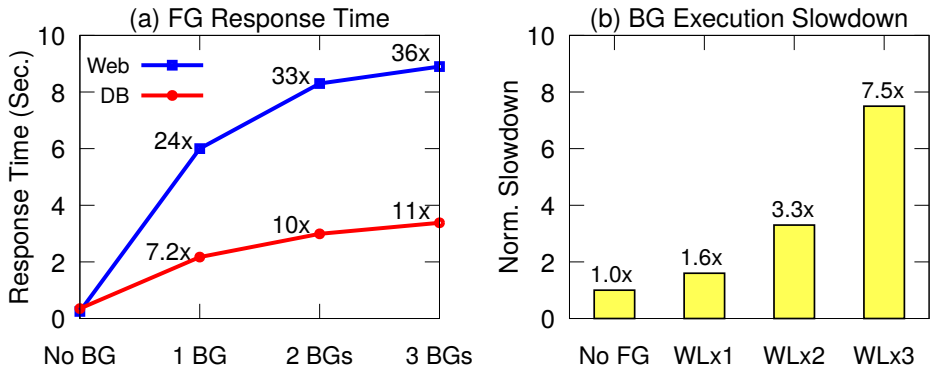


Figure 5.1: **Performance variation of FG and BG applications. (a) the slowdown of FGs (Web and DB) response time (98%tile) when running together with BGs; (b) the slowdown of a 10G data backup duration as the FG workloads increase.**

cases [11]. This requirement incurs *resource storms* that create high peaks of resource usage without knowing when the FGs need more resources. Such resource storms can retard processing time of FG applications and in turn the response time.

Figure 5.1(a) illustrates the performance degradation of two FGs (Olio Web application and MongoDB) when they co-run with BGs. We show that the tail latency (98%tile) of response time could be as slow as 36x. The degraded performance has a significant impact on the QoS of the FGs, resulting in frequent SLA violation and poor user experiences. For instance, Amazon has reported that every 100ms delay loses 1% of the sales profit [124], and video streaming (e.g., YouTube) users start abandoning videos after 2 seconds of buffering time [114].

However, the current cloud instances are not well designed to handle the resource storms. Specifically, stock operating system schedulers such as completely or weighted fair scheduler (CPU, IO) and network queueing (FIFO) mechanisms are designed without considering the resource storms [121, 214], so that SLAs of FGs suffer while parts of shared resources are consumed by BGs [38, 57]. OS modifications – *changing task priority* [159] or *designing a biased OS scheduler* [39] – have been proposed, but such tweaks are not feasible for normal users in the cloud environments due to the technical difficulties. Moreover, intuitive approaches – *terminating* or *suspending* the BGs – to guarantee the FGs' SLA are not sufficient in practice since particular BG tasks (e.g., backup and security checks) could have SLAs[2] to finish the tasks due to the importance of such services [11]. As shown in Figure 5.1(b), the BGs' execution

[2]These SLAs are often very relaxed as compared to the SLAs for the FGs. i.e., once in a day.

time is also highly affected by the amount of FG workloads. Such coarse-grained approaches – *minimizing* the resource allocation – are hard to guarantee the BG's SLAs (or completion of the tasks) and often underutilize the cloud instances by overly controlling the BGs.

The research community has performed significant work, especially when one or more FG applications – normally *latency-sensitive* – are running together with other BGs such as batch jobs [195]. Previous approaches mainly focus on enhancing performance isolation [102, 156, 192, 214], designing intelligent scheduling policies [43, 126, 148, 216] or determining safe co-locations [134, 188, 208]. These techniques often rely on either or both of monitoring the host machine's system/HW-level statistics (e.g., program counter and cache miss rate) and profiling the behaviors of FGs and BGs. While AWS recently started to provide PMU (Performance Monitoring Unit) capability to the dedicated instance users [70], this information is not yet accessible by the users of more general resource provisioning models like on-demand and spot. In general, only cloud providers (e.g., AWS, Azure, or data center operators) are allowed to leverage such information [62, 129], so we do not consider using such information in this chapter. The profiling-based approaches aim to create the performance interference models through off/online measurements. However, the offline models do not provide the flexibility required in highly dynamic cloud environments and workloads [88, 169, 187, 200]. The offline models need profiling of target applications with different constraints – virtual resources (vCPU, memory, network, disk) and even HW architectures –, and different combinations of placement with other types of applications. It is apparent to imagine how much profiling effort needs with all possible combinations. As a result, the online models are more desirable in practice, yet the required computation and monitoring power are very challenging.

To solve this problem, we have created Orchestra, a framework for controlling the FG applications and BG services in the user space, aiming at meeting both SLAs. Orchestra relies on an online approach with very lightweight monitoring at runtime. With the monitoring, Orchestra estimates the response time of FGs using a multivariate polynomial model [72] with a wide range of resource options and predicts a BG's execution time from a multivariate linear regression [72] powered by its resource usage and application-assisted hints. It then optimizes the allocations of diverse resources on cloud instances to both FG and BGs for guaranteeing their SLAs. The resource control by Orchestra leverages the knobs provided by modern OS's improvement such as cgroups [136]. Orchestra is complementary to widely used approaches

for cloud application management. Orchestra components of performance monitoring and resource controlling offer finer-grained mechanisms than off-the-shelf monitoring/management tools like cloud autoscaling and CloudWatch[3] and help cloud users automatically determine *when* to scale.

We have implemented and evaluated Orchestra with real workloads on the production clouds. Our main workloads are a web service and a NoSQL database (MongoDB [145]) for FG applications, and backup (AWS Sync [10]) and virus/malware scanner (ClamAV [37]) for BG services. Our evaluation shows that Orchestra can comply with various SLA targets for FG applications with 70% performance improvement of the BG services. Moreover, Orchestra has a very high overall correctness (less than 5% error), 16.5% of MAPE (Mean Absolute Percentage Error) for the FGs' response time estimation, and over 90% accuracy for the BGs' performance prediction.

5.1.1 Chapter Organization

The rest of this chapter is organized as follows: Section 5.2 contains background of this chapter. Section 5.3 describes the Orchestra framework including the overall architecture, performance models, and its control mechanism. Section 5.4 explains how Orchestra is implemented. Section 5.5 provides the performance evaluation of Orchestra with real-world workloads. Section 5.6 summarizes this chapter.

5.2 Background

5.2.1 Enterprise Cloud Instances

Enterprise or managed cloud instances require high standards for the infrastructure service management such as backup, monitoring, compliance, and patching. The services are typically built by a diverse set of providers. Service vendors often have agent processes running in the background, and they run commands from a central manager or report data to the central location. There has been significant research focus on the resource contention with BG services, especially batch jobs. Although the research community has been mainly focusing on the cloud resource contention problem among VMs as to resource over-provisioning, in practice it is unlikely as cloud providers have reported that the average utilization of cloud instances is about

[3]https://aws.amazon.com/cloudwatch/

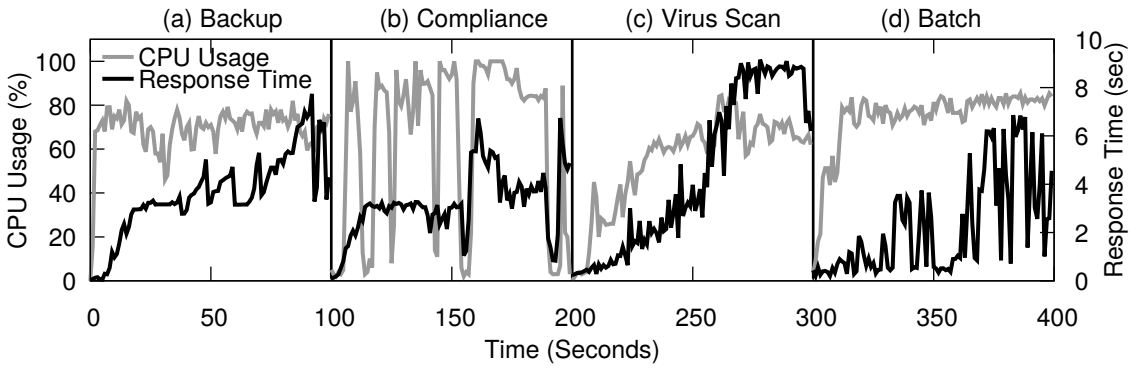


Figure 5.2: **Resource and performance metrics of a managed cloud instance**

20% – 30% [29, 44, 105, 109, 169]. Therefore, over-provisioning resources do not solve the resource contention for cloud applications.

However, each managed cloud instance often runs 5 – 10 BG service agents and their operations are not coordinated as they are from different vendors. Thus, it is highly likely that agents run simultaneously and saturate resources abruptly. This is called *resource storm*. In each cloud instance, resource storms generated from BG management services have a significant impact on the performance of FG applications. Figure 5.2 demonstrates resource and performance metrics of a managed cloud instance that runs a web server (FG) with multiple management services (BGs) such as backup, compliance scan, virus scan, and batch jobs. The response time increases significantly as the BGs preempt shared resources. It is easy to imagine that simultaneous executions could worsen the FG's performance.

5.2.2 Can Autoscaling be a Solution for Resource Storms?

Autoscaling [6, 68, 140] is a widely used approach for the application management on the cloud that fully leverages the benefit – *elasticity* [75, 88] – of cloud infrastructure. Intuitively, the autoscaling might be considered as an approach to this problem by adding more resources to cloud applications when the resource storms happen. (Note that autoscaling is different from the resource over-provisioning in Section 5.2.1.) The autoscaling relies on VMs' resource utilization, and its scaling mechanism could be automatically triggered with high peaks of resource utilization by the resource storms. However, the autoscaling is not sufficient for the following reasons.

First, the resource storms can occur at any time on any cloud instances (VMs).

The VMs, newly added by the autoscaling, are often exposed to the resource storms as well. The BG tasks – *security*, *compliance*, and *patch* – are inevitable for even for the new VMs in practice. A worst-case scenario is that cascading resource storms can incur poor QoS for the entire service from a cloud application. Therefore, it is more appropriate to design an avoidance mechanism for resource storms inside VMs. Furthermore, it is often very challenging for the autoscaling to immediately respond to the resource storms. The autoscaling relies on tools like CloudWatch that monitors the resource utilization on the VMs and catches an unusual resource spike. Unfortunately, a previous work [182] reported that CloudWatch has 1 to 5 minutes of delay to obtain proper monitoring results. Also, a VM provisioning has a delay of 3 to 5 minutes [130]. Enterprise cloud instances generally contain various SW stacks (e.g., web server, DBMS, Memcached) and would take a longer provisioning time than that of vanilla VMs. In the measurement for Figure 2, we observed that the resource storms often have a short-duration and abruptly happen, indicating that the autoscaling is not sufficient to promptly address the resource storms.

5.2.3 User Space Resource Control

The resource controls in user space for each resource type has been developed separately thus far. A CPU control, *nice* directly maps to a kernel system call to manually adjust the task's priority level of a process, and yet another CPU resource control, *cpulimit*[4] repeatedly curbs the CPU usage of a process by stopping the process at different intervals to keep it under the defined ceiling. A disk resource control, *ionice* gets and sets program IO scheduling class and priority to adjust the disk usage of processes. There are network traffic shaping tools such as *trickle* [54], *force_bind*[5], and *damper*[6] that throttle the traffic bandwidth of processes.

With the emergence of cloud computing, there have been demands for the user space resource control mechanisms because otherwise users have to create custom kernels. Especially, the control groups (cgroups) have gained large attention recently under container virtualization dominance. The Linux's built-in cgroups is a mechanism which tells OS schedulers to limit the amount of resources (e.g., CPU time, system memory, disk IO, network bandwidth, or combinations of these resources) available to processes from user space, and this allows users to specify how the kernel

[4]http://cpulimit.sourceforge.net/
[5]http://kernel.embedromix.ro/us/
[6]https://github.com/vmxdev/damper

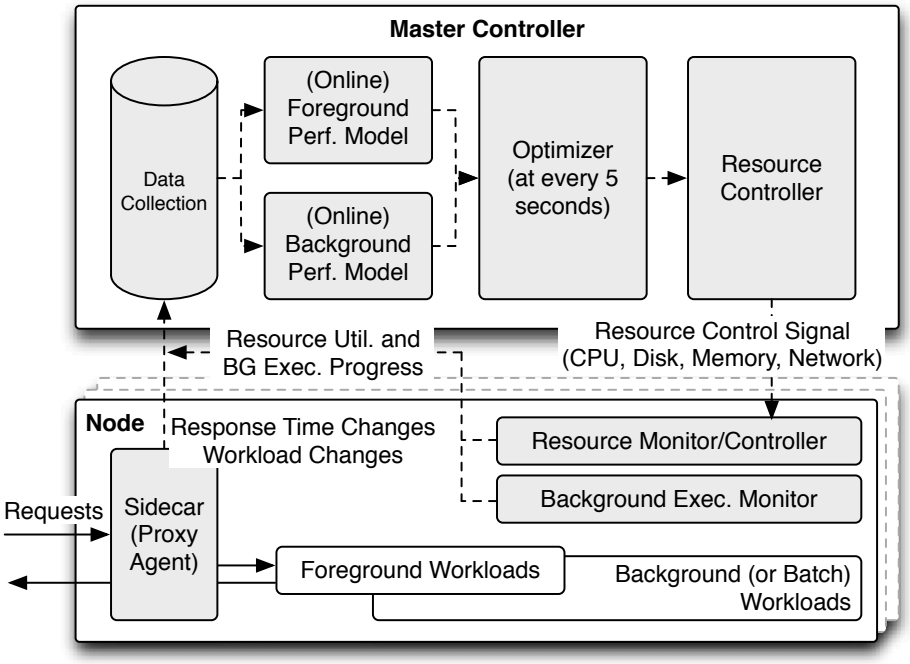


Figure 5.3: **Overall architecture of Orchestra.**

should allocate specific resources to a group of processes. In particular, reconfiguring resources dynamically on a running system is amenable to user space resource control with fine-grained control over allocating, prioritizing, denying, managing, and monitoring system resources. The advantage of cgroups over prior implementations is that the limits are applied to a set of processes, rather than to just one. Orchestra adopts the user space resource control mechanisms.

5.3 Orchestra Framework

5.3.1 Orchestra Overview

Overall Architecture: We design Orchestra with a two-layer, distributed architecture, of *managed nodes* and a *master controller*. Figure 5.3 illustrates the design of Orchestra.

A **managed node** (VM instance) has two components in Orchestra– *sidecar* and *node agent*. The sidecar – *a traffic forwarder used as an online performance monitor* – is designed to watch performance variation of the FGs, i.e., a response time of web requests and DB transactions. It measures the FG's processing time by capturing the

ingress and egress time of user requests. Moreover, the sidecar can monitor a diverse set of FG workloads as long as they use general purpose protocols (e.g., HTTP, TCP) to communicate with end-users. The measured FG's performance is reported to the data collector in the master controller.

The node agents are used for 1) monitoring the resource usage of the target applications, 2) monitoring the progress of BG's execution, and 3) reconfiguring resources allocation to both FG and BGs. The monitoring of the resource usage focuses on collecting the general system statistics, i.e., vCPU, memory, disk and network IO. The BG's execution progress is relying on probing the application-assisted hints such as retrieving logfiles. All the collected statistics – resource utilization and application progress – are reported to the data collector in the master controller. The resource reconfiguration is to manage subsystems of control knobs (e.g., cgroups) with the decision made by the master controller.

The **master controller** plays the most important role in Orchestra by determining the adjusted resource allocations to both FG and BGs with the goal of satisfying FG's SLA requirement and maximize BG's executions. To this end, with various statistics – response time, resource utilization, and application progress – from the node agents, the master controller creates a *response time estimator* (Section 5.3.2) and *performance model* (Section 5.3.3) for both applications on the fly. With these models, the master controller optimizes the resource allocations to achieve the management goal. The detailed mechanisms will be explained in Section 5.3.4.

Orchestra Workflow: Figure 5.4 shows the workflow of Orchestra. Orchestra starts with monitoring diverse statistics of the FG and BG applications' performance and the instance's resource utilization. And it creates feature vectors with the statistics, and these feature vectors are used to train and create two predictive models that forecast the FG's response time and BG's execution duration. If a response time of the FG application is close to or violates a SLA target (defined by Orchestra operator), Orchestra adjusts the resource allocations to both applications through an optimization with two predictive models. Once the proper resource allocation is determined, the decision is sent to a node agent that will change resource usage to both applications. If the performance of FG becomes stable (meeting the SLA targets), then Orchestra stops controlling the resource allocation and comes back to the steady monitoring state that collects the essential statistics to tune two predictive models.

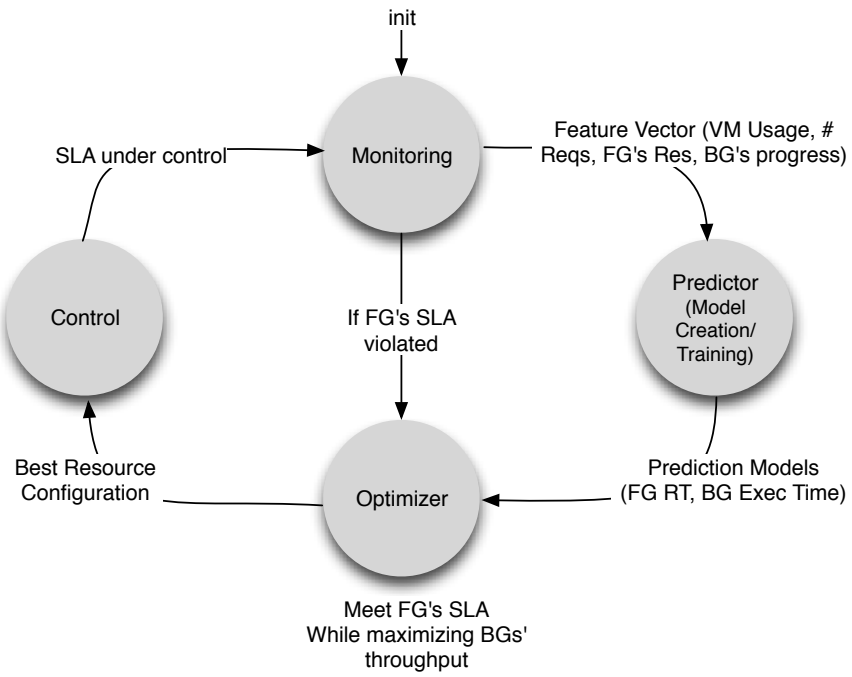


Figure 5.4: **Orchestra state diagram.**

5.3.2 Response Time Estimator for Foreground Applications

A key component of the master controller is the RT (Response Time) estimator that predicts (the near future) web response time or DB transaction time with a broad range of resource utilization. Since Orchestra's decision on the resource control relies upon this estimation model and the resource control should be made at runtime, high accuracy and low overhead are an essential prerequisite.

Feature Selection: We observe the behaviors of two FGs (Web and MongoDB) by running benchmark tools (CloudSuite [59] and TPC-C [163]) without BGs' execution. We then calculate the Pearson Correlation Coefficient between the FG's RT and the following features including the number of requests and various system resources – CPU, memory, disk and network IOs.

Figure 5.5 reports the measured correlation. Three factors show the highest correlation with the RT of web application (FG) – CPU, MEM (Memory) and NRX (Network RX Bytes). The coefficients are between 0.6 and 0.75. Note that we do not measure disk IO for the web application since it has very negligible disk operations. In the MongoDB benchmark, the all features show relatively weaker correlations. Four factors – REQN (the request numbers/sec.), CPU, NRX, and NTX (Network

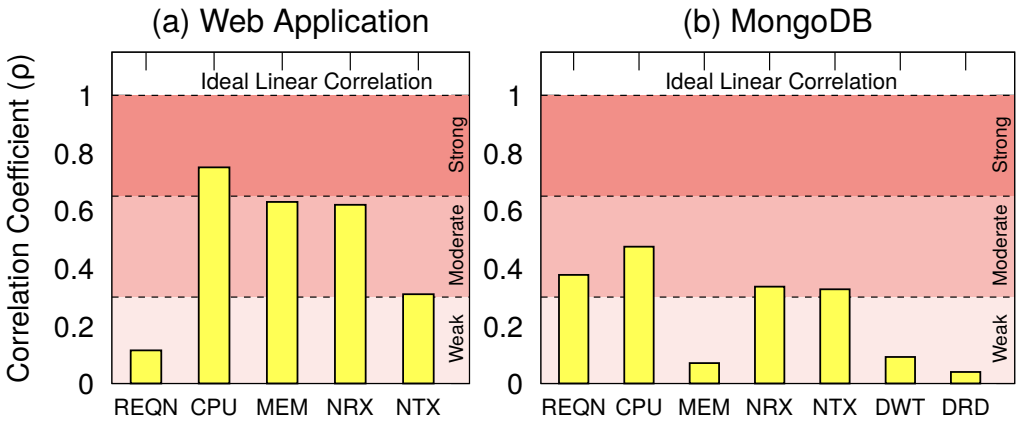


Figure 5.5: **Correlation coefficient of factors that could affect FG's response time. (NRX: Network RX Bytes/sec. NTX: Network TX Bytes/sec., DWT: Disk Write Bytes/sec., DRD: Disk Read Byte/sec.)**

Table 5.1: **Correlation between the selected factors. (1: weak, 2: moderate, 3: strong correlation)**

(a) Web Application

	CPU	**MEM**	**NRX**	**NTX**
CPU	-	2	3	3
MEM	2	-	2	1
NRX	3	2	-	3
NTX	3	1	3	-

(b) MongoDB

	CPU	**MEM**	**NRX**	**NTX**
CPU	-	1	3	3
MEM	1	-	1	1
NRX	3	1	-	3
NTX	3	1	3	-

TX Bytes) – show a moderate correlation with the MongoDB's RT. The coefficients are slightly over 0.3. While the MongoDB has weaker factors, we decide to consider all these (correlated) factors to model the RT estimator because the RT estimator aims to handle both or potentially more types of FGs. The selected factors are CPU, MEM, NRX, and NTX. We exclude REQN from this feature selection since NRX is a more comprehensive metric that covers REQN. For the other FGs, users may add other factors if necessary.

Moreover, we measure the correlation coefficient among these four factors since there could be a certain possibility that one factor can be correlated with other factors. i.e., a correlation of CPU and Network IO. We report the correlations among the factors represented by 1 to 3 of scale (1: weak, 2: moderate, and 3: strong correlation) and the results are reported in Table 5.1. CPU and Network IO are strongly correlated each other for both FGs. MEM is moderately (Web) or lightly (MongoDB) correlated with other two factors. MEM also shows low variance over

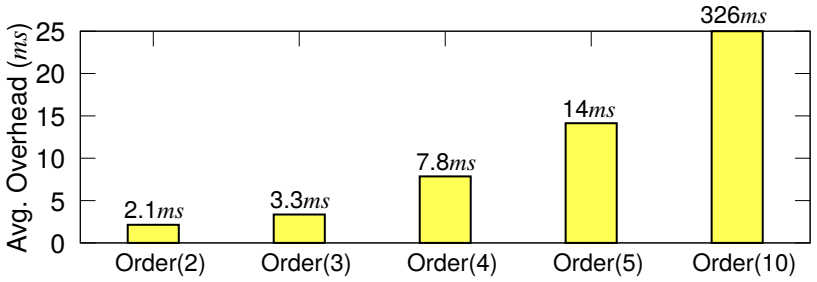


Figure 5.6: **Computational overhead of MVPR model with different polynomial orders.**

the RT's fluctuation. i.e., (μ of 13, σ of 11) for Web, (μ of 23, σ of 2) for MongoDB.

Model Selection: We chose MVPR (Multivariate Polynomial Regression) [72] approach to model the RT estimator because MVPR considers both 1) multiple factors' contribution to the estimation target and 2) the correlation among the selected factors. The MVPR model is expressed as below:

$$f(x_1, x_2, ..., x_p) = \sum_{i=0}^{N} \beta_i \phi_i \tag{5.1}$$

where p indicates the number of independent variables, β_i is coefficient, $\phi_1 = 1$, $\phi_N = x_1^n \cdot x_2^n \cdot x_3^n \cdots x_p^n$, and n is the order of the MVPR.

When applying a polynomial model to a runtime system, the computational overhead is highly concerned. The overhead depends on both the number of the independent variables (p) and the order (n) of the model. With this concern, we use a harmonic mean[7] of NTX and NRX, both representing network-IO statistics and it helps to reduce the number of equation terms. i.e., with a quadratic model, 3 independent variables require 27 terms and 4 variables generate 64 terms. Regarding the order of the model, we empirically test the overhead with a training dataset (1 hour of Web RT data) and various polynomial orders from two to ten. The overhead exponentially increases as the order of the polynomial model increases as shown in Figure 5.6. However, the overhead is not too high in the model with the orders less than 5. For example, the quartic model (order of 4) has the average computational overhead of 7.8 ms and, its highest overhead of prediction is just as high as 87 ms. We limit the order of the model with this observation. Moreover, the overhead of MVPR can be determined by the size of the training dataset. The RT estimation

[7] $2/(1/NTX + 1/NRX)$.

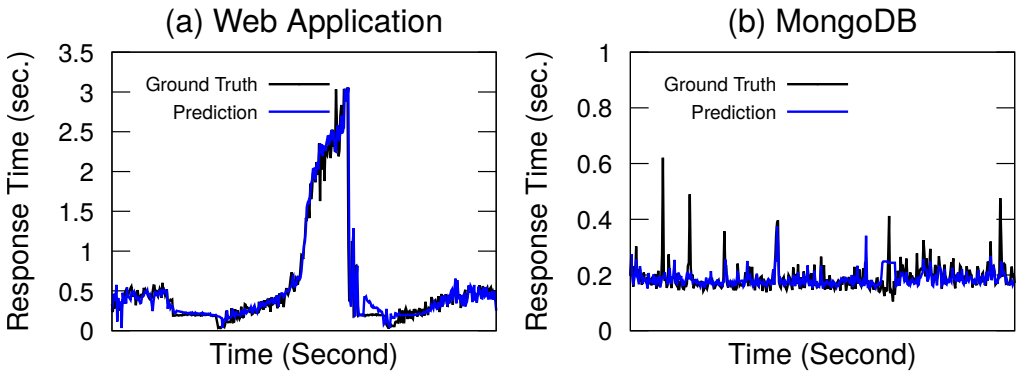


Figure 5.7: **Prediction results from MVPR model (order of 2) for RTs of Web and MongoDB.**

model with MVPR incorporates with a sliding window technique [89] to limit the size of the training dataset as well as leverage the most recent observations.

Model Accuracy: We measure the performance of the RT estimation model with Web and MongoDB. Figure 5.7 illustrates the RT (ground truth) from both FGs and prediction from the model with an order of 2. While this evaluation is performed with less challenging condition (without BGs), it is obvious that the model accurately estimates the RT of the FGs. The prediction shape from the model successfully catches the trend of the RT variation for the web application. For the MongoDB, the model has more errors than the case of the web application, but it also shows robust predictions except for some outliers. We perform more comprehensive evaluations for the accuracy of the RT estimation model with the BGs' executions in the evaluation section.

5.3.3 Performance Models for Background Services

Orchestra requires a performance model that predicts BGs' execution time. The model is essential for monitoring and controlling BGs services because Orchestra needs to assure BGs' SLA satisfaction and/or minimizing their execution time. So this model performs a critical role in optimizing the resource allocation with an accurate prediction of difference resource usages. To create such a model, we consider ClamAV[8] and AWS Sync[9] as the examples of BGs.

[8]ClamAV is an open-source anti-virus engine used to defend user instance (e.g., VM) from computer viruses, Trojan, and other malicious threats [37].

[9]AWS Sync is a backup application for Amazon EC2 instances similar with rsync [10]. Sync recursively copies new or updated files from a source directory on an EC2 instance to S3 [9] storage.

Table 5.2: **Statistics of measurement results for understanding BG applications' characteristics on two EC2 instances.**

	(a) ClamAV		(b) Sync	
Dataset	**35G**	**106G**	**35G**	**106G**
CPU[11]	73.8%	86.5%	78.6%	95.2%
Memory	14.2% (0.53G)	19.5% (0 73G)	3.3% (0.12G)	3.9% (0.15G)
Disk Read	8.3 MB/s	40 MB/s	51 MB/s	133 MB/s
Disk Write	79 KB/s	301 KB/s	2.2 KB/s	159 KB/s
Network TX	-		55 MB/s	141 MB/s
Network RX	-		1.4 MB/s	3.7 MB/s

Feature Selection: We perform a profiling study on two different Amazon EC2 instances[10] – m3.medium and c4.large – of Ubuntu 16.04 LTS with 35G (10K files) and 106G (50K files) dataset. In this measurement, we use the default configuration of Ubuntu OS and run these two BG services individually without any FGs' execution. The statistics and results from this profiling are shown in Table 5.2. ClamAV is observed as a CPU and Disk-IO (Read) bound application and moderately consumes memory resources. Sync mostly consumes CPU, Disk-IO (Read) and Network (TX) resources on the instances. Since both CPU and Disk-IO (Read) are common resource factors that can potentially affect the performance of the BGs, we decide these two resources as main features for the performance models of these two BGs.

Also, we consider leveraging application-assisted hints from these two applications. Intuitively, the BGs' performance could be highly related to the size and number of files they manage. Fortunately, the BGs, like many other applications, support a capability to write log files that saved how many files they scanned or backed up. To test such hints' applicability, we measure the correlation between the size and numbers of files saved in the logs and the execution progress of two BGs. Figure 5.8 represents the progress of file size and numbers scanned or backed up by ClamAV and Sync according to the BGs' execution. Compared to the ideal progress (black line in Figure 5.8), while the progress of the processed numbers and size of files are slightly different from the progress of ideal case, it is obvious that the processed files (numbers and size) are correlated with the ideal progress of BGs. On average, the

[10] m3.medium instance has 1 vCPU, 3.75G RAM, and SSD drive [3]. c4.large instance has 2 vCPUs, 3.75G RAM, and SSD Drive [3].

[11] 100% of CPU means the full usage of 1 vCPU.

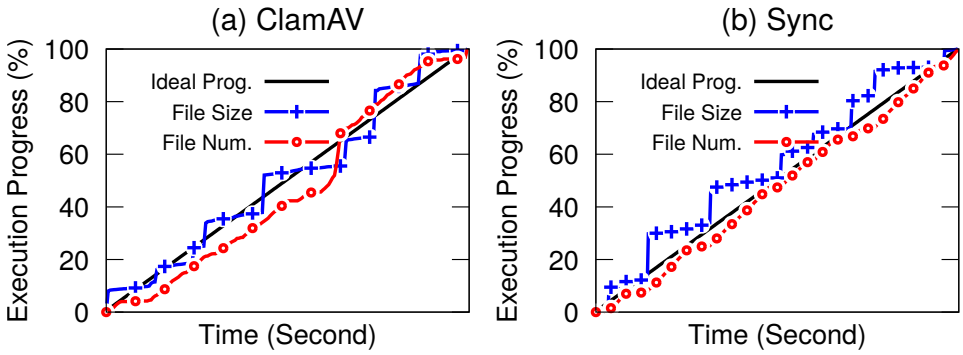


Figure 5.8: **Changes of file size and numbers managed by BG services. (Comparison with the ideal progress.)**

progress by the number and size of processed files has 4.85 and 4.93 of MAE[12] (Mean Absolute Error) and a harmonic mean[13] of two factors has just 2.9 of MAE over the ideal progress. Thus, we consider such hint as a feature for the performance model of BGs and use the harmonic mean of them.

Model Selection: We design the performance models with a multivariate linear regression [72] that models the linear relationship between independent variables (the features) and the corresponding variable y (BG's execution time). The model is formulated as below:

$$y = \sum_{i=1}^{n} \alpha_i x_i + \beta \tag{5.2}$$

where x is the independent variables ($x \in [CPU_{bg}, DRD_{bg}, HINT_{bg}]$) and β is a constant. The corresponding variable y means the (predicted) execution time of the BGs. In this chapter, we consider three features to design the performance models, and users can add/remove more features according to the performance characteristics of other BGs.

5.3.4 Orchestra Resource Optimizer and Controller

Now we explain how Orchestra determines the resource allocation to both applications with two predictive models described in the previous sections. The primary objective of this decision is to satisfy the FG's SLA, so the RT estimation model (Equation (5.1) in Section 5.3.2) should have the following condition. Suppose SLA_{fg} indicates a SLA

[12] $|Progress_{ideal} - Progress_{log}|$.
[13] $2/(1/file\_size + 1/file\_numbers)$.

target for a FG:

$$f(CPU_{fg}, MEM_{fg}, NET_{fg}) \leq SLA_{fg} \tag{5.3}$$

To simplify this equation, we can consider MEM_{fg} as a *constant* because the memory resource has a weak correlation with other factors (shown in Table 5.1) as well as it has no significant variance with the fluctuations of FG's performance. We replace MEM_{fg} with the average memory utilization of the FG. We can also estimate NET_{fg} from EMA (Exponential Moving Average) [216]. This estimation may result in slightly inaccurate prediction for the RT estimation, but it greatly reduces the computation overhead for the RT estimation. i.e., $O(n^2)$ to $O(n)$. Now we transform the RT estimation model from multivariate to univariate model, depending on CPU_{fg}. We can obtain the minimum value of CPU_{fg} that satisfies the SLA_{fg} from the below equation:

$$\hat{CPU}_{fg} = \underset{CPU_{fg}}{\arg\min}\, f(CPU_{fg}) \leq SLA_{fg} \tag{5.4}$$

where $0 < CPU_{fg} < CPU_{max}$. CPU_{max} is the maximum amount of CPU resources in the VM. If CPU_{fg} from Equation (5.4) is greater than CPU_{max}, this means that the FG is impossible to meet SLA requirement with 100% CPU utilization on the instance. Thus, in this case, Orchestra provisions more resources to the FG by collaborating with cluster or application management techniques (e.g., autoscaling) to ensure the SLA satisfaction. With CPU_{fg}, Orchestra can determine the CPU allocation for the BGs by:

$$CPU_{bg} = CPU_{max} - (CPU_{fg} + \varepsilon) \tag{5.5}$$

where ε is the CPU utilization for a third application or the reserved amount of CPU for unknown processes.

Next, Orchestra performs an optimization to minimize Equation (5.2), which is the performance model of the BGs.

$$\begin{aligned} \text{minimize:} \quad & \sum_{i=1}^{3} \alpha_i x_i + \beta, \text{ where} \\ & x_i \in \{CPU_{bg}, DRD_{bg}, HINT_{bg}\} \end{aligned} \tag{5.6}$$

$$\text{subject to:} \quad CPU_{bg} = CPU_{max} - (CPU_{fg} + \varepsilon) \tag{5.7}$$

$$0 \leq DRD_{bg} \leq DRD_{max} \tag{5.8}$$

$$HINT_{bg} = 1, (100\% \text{ prog. of BG}) \tag{5.9}$$

The solution of this optimization determines the desired utilization of Disk IO for the BGs. From Equation (5.3) to (5.9), Orchestra determines all resource allocations to both FG and BGs. The set of resource allocation to both the FG and the BGs are sent to a node agent on the VM instance, which reconfigures resource allocation with cgroups.

5.4 Implementation

The implementation of Orchestra follows the architectures of recent cluster management frameworks such as Kubernetes [116] and Docker Swarm [52] as shown in Figure 5.3 (in Section 5.3). The main components of these architectures include a master and nodes to manage and orchestrate virtual machines, and the sidecar component is used in the micro-service architecture such as Netflix OSS and Istio as a packet forwarder in both/either ingress and/or egress[14]. In the master controller of Orchestra, two predictive models – the RT estimator (FG) and performance model (BG) – are implemented with various statistics and machine learning libraries. i.e., scikit-learn, statsmodels, Pandas and others.

The implementation of node agent focuses on the resource monitoring and control. Sysstat[15] is used to periodically monitor the changes of resource utilization on the VM instances. To control multiple resources, Orchestra consults with two subsystems of cgroups [136] – *cpu* and *blk_io* – to control the CPU and disk IO, and utilizes *tc*[16] for network IO. Whenever the new resource allocations are decided, Orchestra reconfigures a different set of tunable values to *cpu.shares* and *cfs_period_us* (for CPU control) and *read_iops* in *blk_io* (for disk IO control). While Orchestra can determine the proper reconfiguration for network IO and is fully implemented to utilize *tc*, according to our experience, reconfiguring *tc* is often unnecessary because applying the updated value of *cfs_period_us* automatically adjusts network IO as well. (Moreover, in Section 5.3.2, we discussed that both CPU and network IO are highly correlated.)

The sidecar – a performance monitor for the FG – is based on Nginx's reverse proxy [152] and load-balancing [151] functionality. Currently, the sidecar supports multiple protocols for the FG workloads – HTTP and data stream (TCP and UDP) requests – and it forwards the requests to the corresponding FG applications. With

[14]https://istio.io/
[15]http://sebastien.godard.pagesperso-orange.fr/
[16]http://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html

the Nginx's recent improvement, the sidecar can capture ingress and egress time of each request, and the statistics of the FG's RT are reported to the master controller in a real-time manner.

5.5 Performance Evaluation

We evaluate Orchestra on the real cloud environment. We first demonstrate the performance of Orchestra for controlling both FG and BGs to satisfy the FG's SLA goals under the resource storms (Section 5.5.2). Section 5.5.3 analyzes the overall accuracy of Orchestra as well as the accuracy of the RT estimator (FG) and performance models (BG). We then report the overhead of Orchestra (Section 5.5.4).

5.5.1 Evaluation Setup

Evaluation Infrastructure: We use general purpose m4[17] instances of Amazon EC2 clouds since Orchestra aims to provide fine-grained control mechanisms of various VM resources to general cloud users (of course, all controls are performed in the user space). As several works reported [56, 120, 160], Amazon EC2 has *performance variance* due to the resource contentions and HW heterogeneity on the base infrastructure. We use EC2 spot instances in this evaluation (Spot may have even higher level of the performance variance.) – multiple runs and averaging them offset the *variance*.

[17] m4 instances are the latest generation of VM instances and have balanced resource combinations, i.e., the ratio between CPU and memory is 1:4 [3].

FG Workloads: We consider Web application and MongoDB as representatives of FGs and use two different benchmarks for each FG to generate real workloads; CloudSuite 3.0 [59] - Web Serving benchmark and TPC-C [163] for MongoDB. For the Web application, we generate web serving workloads from 50 to 250 concurrent users to create a sufficient level of workload fluctuation. We set up different VMs for the web server (m4.large[18]) and back-ends – Memcached and DBMS – (m4.xlarge[19]) and focus on the resource controls for the front-end (web server) VM. For MongoDB, we install the latest version of MongoDB on m4.large instance and continuously change the number of concurrent users from 2 to 20 to generate the realistic workloads.

BG Workloads: A 5GB of a dataset is used for BG workloads. i.e., ClamAV (virus scan) and AWS Sync (backup). This dataset has approximately 25K of files with various sizes (μ of 1024K, σ of 1495.6). We use a different dataset from the dataset we used in Section 5.3.3 for a fair comparison.

Performance Metric: In Section 5.5.2, meeting a broad range of SLAs is the primary metric. We focus on tail latency – *95%tile* – for this measurement and define that a SLA requirement is satisfied if a 95%tile of a FG's response time is equal to or less than the SLA target. In Section 5.5.3, we use two well-known metrics of measuring the model accuracy; MAPE (Mean Absolute Percentage Error) and RMSE (Root Mean Square Error). In general, lower values of both metrics mean better accuracy. The overhead of Orchestra (Section 5.5.4) measures CPU usage of the master controller and the managed nodes (including the overhead from the sidecar and node agent) when they monitor, predict, and control target applications and resources.

5.5.2 Overall Performance with Foreground Workloads

This section focuses on the overall performance of Orchestra; meeting the SLA goals while minimizing the BGs' execution time. Before we start this evaluation, we need to recognize the range (the upper and lower bound) of SLA targets that Orchestra should meet. We quantify RTs (Table 5.3) of four different scenarios when a FG runs with BGs (resource storms) and without BGs. We pick SLAs for each case within the range of between "RT without a BG" and "RT with BG(s)". Three SLAs – 25%, 37.5%, and 50% $\in$ [RT without a BG, RT with BG(s)] – are chosen for evaluations

[18]m4.large has 2 vCPUs, 8GB RAM, and SSD storage.
[19]m4.xlarge has 4 vCPUs, 16GB RAM, and SSD storage.

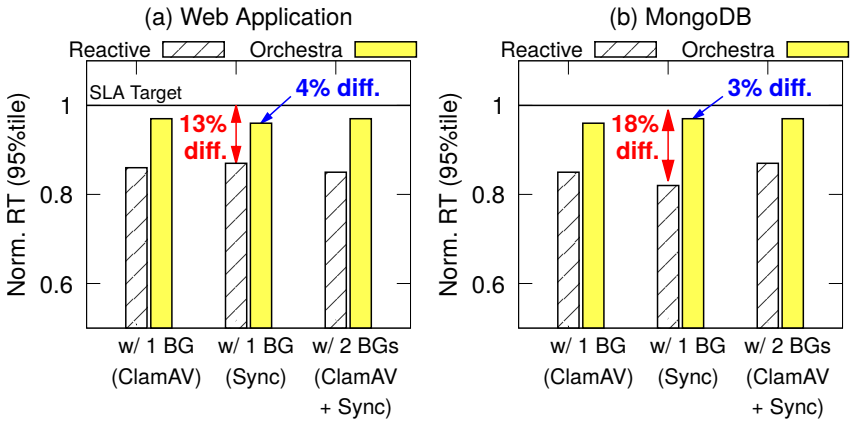


Figure 5.9: **Normalized RT (95%tile) of Web and MongoDB over a set of SLA targets (The best result should be 1.0.)**

with resource storms. We define these three SLA requirements as a "tight SLA (with 25% padding)", "moderate SLA (with 37.5% padding)", and "loosed SLA (with 50% padding)."

Table 5.3: **95%tile response time (RT) of Web and MongoDB with and without resource storms from BGs.**

	Number of BGs	Web	MongoDB
FG Only	0	0.92s	0.86s
FG + ClamAV	1	5.88s	2.71s
FG + Sync		8.65s	3.57s
FG + ClamAV + Sync	2	11.78s	4.31s

We use a reactive approach from other works [208, 214] as the baseline. This reactive system relies on a dynamic adjustment of maximum resource capping. The reactive one sets a low value of the maximum cap (e.g., 5% of CPU utilization) for BGs' resource consumption when the FG's RT violates the SLA goals, and it allocates more resources to the BGs by releasing this cap when the FG shows a stable performance. Also, we add a warning system to this baseline, and the warning system sets a threshold (e.g., 10% gap from the SLAs). When the FG's 95%tile RT exceeds the threshold, the warning system alerts to the master controller, and the reactive framework reduces the resource usage of the BGs with a pre-defined step function. If the FG's RT violates the SLAs, the reactive one uses the resource capping explained above to quickly restore the FG's performance with the SLA.

Figure 5.9 shows the RT (95%tile) of the FG applications managed by Orchestra

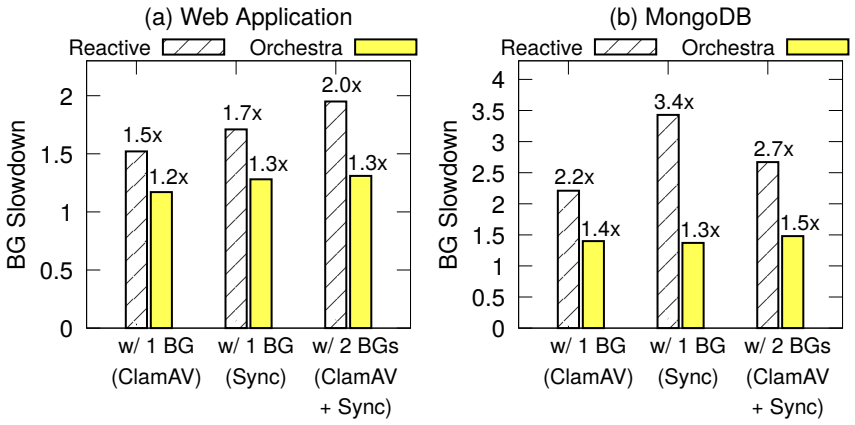


Figure 5.10: **Slowdown of BG's execution under two different control frameworks.**

and reactive approach when the FGs are running with the BGs. All results are normalized over the SLA targets. If a result is equal to or less than 1.0, then the RT meets the SLA goals and vice versa. As shown in Figure 5.9, both frameworks can successfully control the FGs' RT with the SLAs. But Orchestra's results are much closer to the SLAs, and it only has 2 – 5% difference with the SLAs. However, the reactive system has about 15% differences with the SLA targets. These results indicate that the FGs controlled by Orchestra consumes significantly less amount of resources to meet the SLA targets as compared to the FGs with the reactive system. Again, the goal of Orchestra is *not* maximizing a FG's performance, rather meeting the FG's SLA as well as augmenting the performance of a BG(s). These results have a huge impact on the BGs' performance since Orchestra can allow the BGs to utilize more resources to boost their execution (more importantly, without SLA violations). Figure 5.10 reports the difference of the BG(s)'s performance and obviously shows the benefits from Orchestra. While Orchestra only has 1.25x (with Web) and 1.39x (with MongoDB) slowdown[20] of the BGs' execution, the reactive approach requires more sacrifice to the BGs, i.e., 1.73x (with Web) and 2.77x (with MongoDB) slowdown of the BGs. These results are because Orchestra's predictive ability and optimization mechanism could successfully determine the proper level of resource allocation to multiple applications so that Orchestra allows the BGs to consume as high resource as if the FG meets the SLAs.

Figure 5.11 shows resource utilization on the VMs with both approaches. We only show CPU utilization since it is the most representative resource of the VMs.

[20]This is normalized over the BGs' execution without FG workloads.

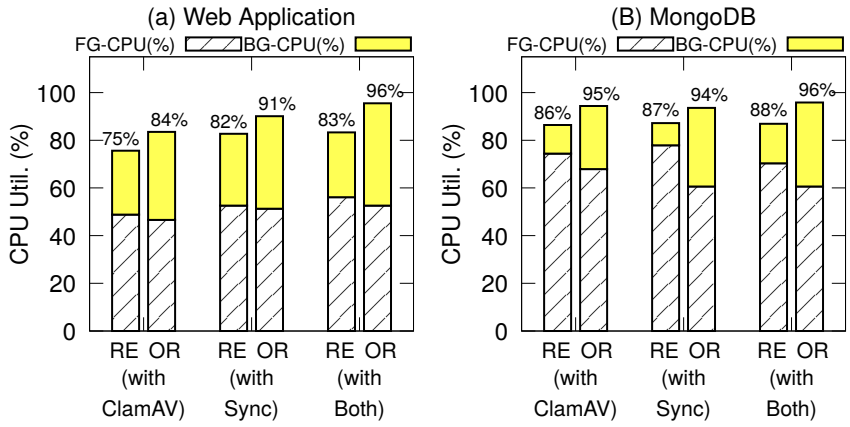


Figure 5.11: **Resource utilization (CPU) of VM with two control frameworks. (RE: reactive framework, OR: Orchestra)**

For the Web application case (Figure 5.11(a)), Orchestra utilizes over 90% of CPUs, which is leveraging 10% more resources than the reactive approach. Interestingly, Orchestra uses the similar amount of CPU (compared to the reactive one, only 2% difference) for the Web (FG) and increases the overall utilization by allocating more resources to the BGs. In the evaluation with MongoDB (Figure 5.11(b)), Orchestra improves CPU utilization over 95% and provides balanced resource allocations to the both, implying Orchestra is not only able to meet the FG's SLAs, but also boost the BG's performance, resulting in high resource utilization on the VMs. However, the reactive one allocates more than 75% of CPU to the FG, indicating it overly assigns the resources that lead to retard the BGs' execution.

Figure 5.12 and 5.13 illustrate Orchestra's behavior for managing FG's RT and controlling VM resources. The top graph shows the changes in the FG's RT and the SLA target. The middle graph reports the CPU control for the FG, and the bottom graph shows the CPU control for the BG. For the use case of Web (Figure 5.12), while Orchestra controls the CPU resources based on its estimation mechanism, the most critical SLA violation happens in the first 100 seconds (the top left graph). This violation is because the RT estimation model for FG does not have enough training dataset to build a robust model. After this initial training period, Orchestra performs the successful control in the CPU resources. The middle and bottom graphs in Figure 5.13 show that Orchestra increases CPU resources for the FG and, at the same time, decreases the resource for the BG when the FG's RT is close to or violates the SLA target. i.e., at 220, 250, 400, and 580 second. Similar behaviors are shown in the MongoDB case (Figure 5.13). The first 350 second period (top right graph) can be

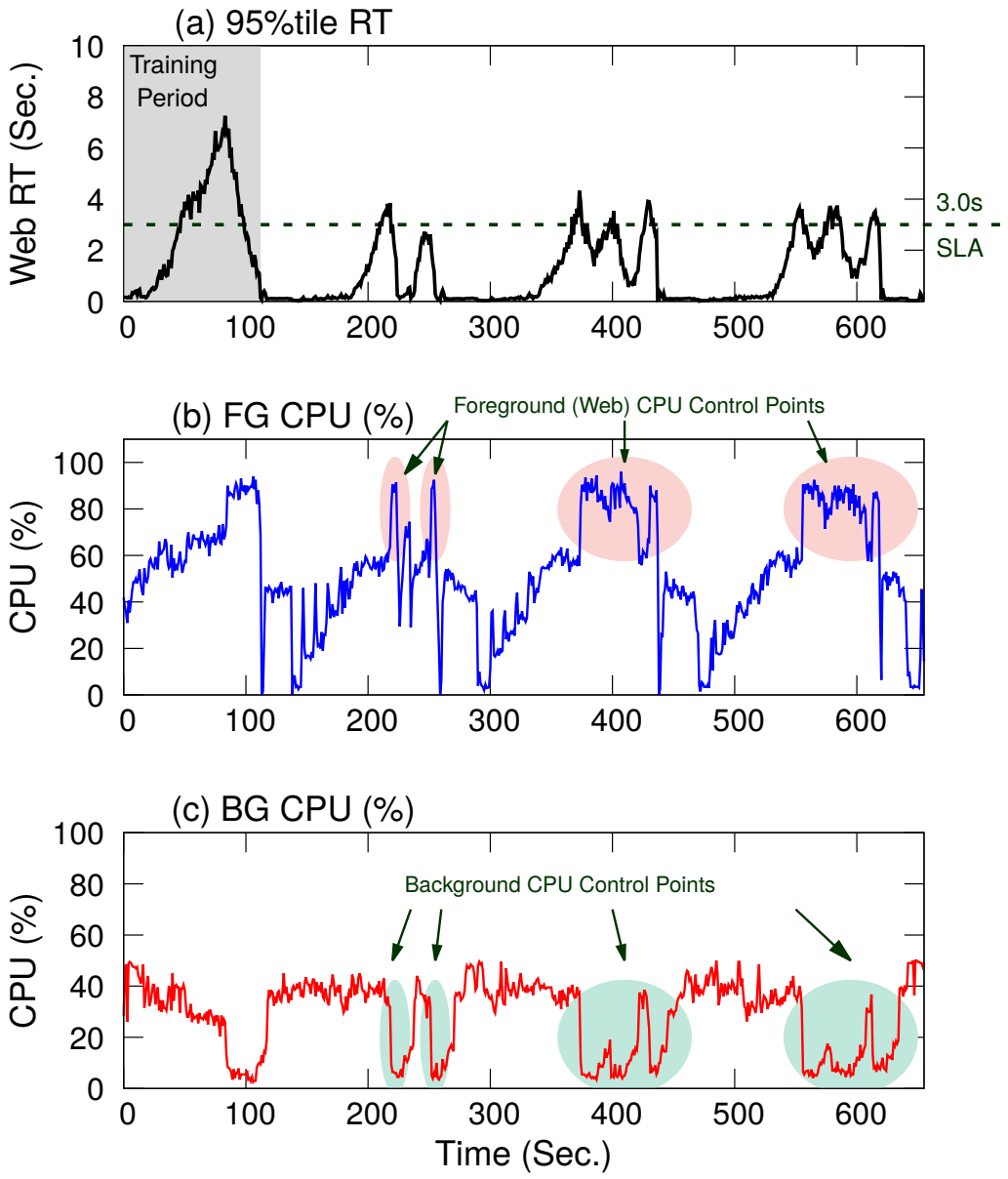


Figure 5.12: **Change of CPU resources for both a FG and BGs from Orchestra and the changes of 95%tile RT of the FG. Web (FG) and Sync (BG) with a SLA of 3.0 sec.**

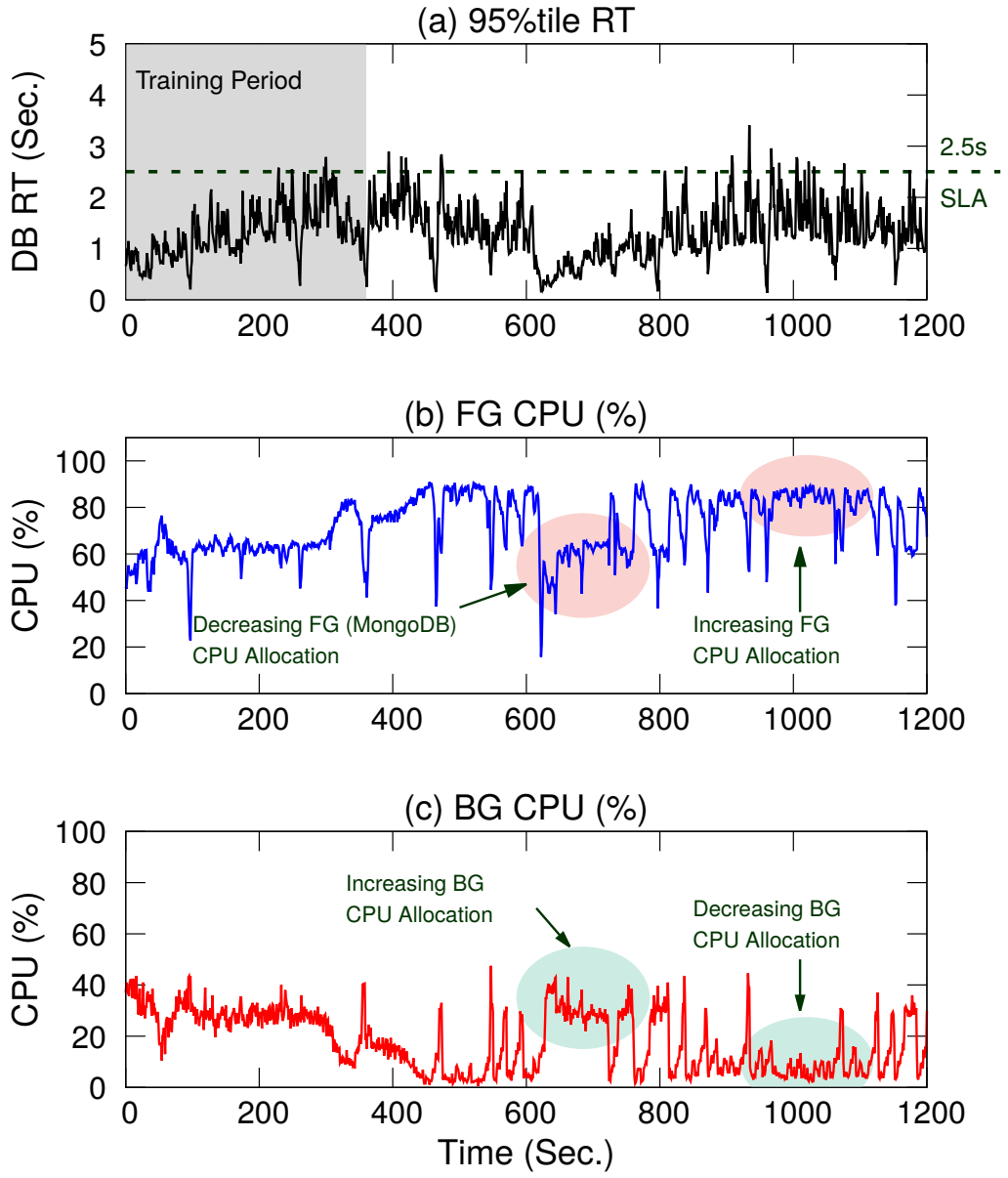


Figure 5.13: **Change of CPU resources for both a FG and BGs from Orchestra and the changes of 95%tile RT of the FG. MongoDB (FG) and 2 BGs (Sync and ClamAV) with a SLA of 2.5 sec.**

considered as the training period. After this training period, Orchestra shows stable adjustments in CPU resources for both applications regarding the SLA satisfaction. For instance, during the period between 600- and 800-second, Orchestra reduces the CPU allocation to the FG (middle graph) and allocates more to the BGs (bottom graph) as the MongoDB's RT is a lot faster than the SLA goal. Another example can be shown in the period between 900- and 1100-second. Orchestra increases the CPU allocation to the FG while the CPU allocation for the BG decreases since the MongoDB's RT is close or slight violates the SLA goal. Then, the FG's RT becomes stable.

5.5.3 Orchestra Accuracy

Performance evaluations of Orchestra's accuracy include the overall model accuracy of Orchestra and the accuracy of two predictive models. i.e., the RT estimator (FG) and the BG performance model.

Overall Orchestra Accuracy: We first collect various system/application statistics (including RT, utilization, and throughput from FGs, and BG's execution) with a set of different resource configuration between a FG and a BG. We manually assign fixed resource ratios to the FG and BG(s) from 1:1 to 1:5. We then generate the identical FG workloads to Orchestra with SLA targets obtained from the static resource allocation. We compare the results/statistics from Orchestra with the ones from the static resource allocation and calculate the accuracy (normalized over the static allocation) of both cases.

Table 5.4: **Orchestra's accuracy (MAPE) over the static resource allocation. The results show FG statistics.**

FG	BG	RT Statistics (%tile)				Throughput (Requests/sec.)
		Average	90%	95%	97%	
Web Application	ClamAV	0.050	0.061	0.038	0.011	0.007
	Sync	0.132	0.008	0.029	0.028	0.020
MongoDB	ClamAV	0.010	0.015	0.006	0.013	0.036
	Sync	0.003	0.012	0.014	0.003	0.015

First, we show the accuracy of Orchestra regarding FG behaviors. The results are shown in Table 5.4 and Figure 5.14. The changes in FG's RT with Orchestra are very close to the RT variations in static resource allocations. On average, Orchestra can

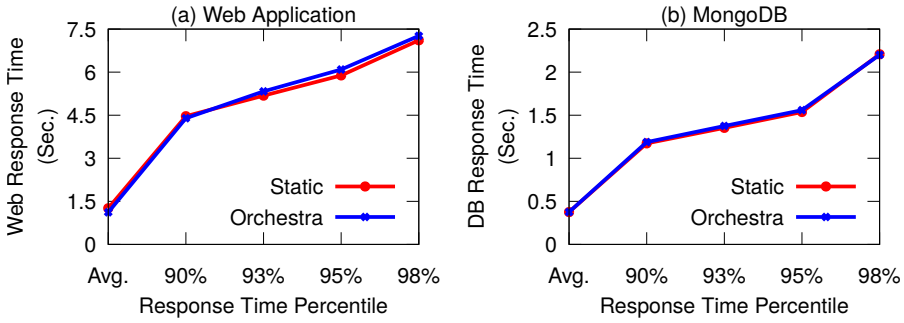


Figure 5.14: **Accuracy for the FG's RT; comparison with the static resource allocation.**

control a FG with 2.7% errors in RT and 1.95% error for its throughput.

Then, we observe the difference in the resource utilization of the VM instance between Orchestra and the static resource allocations. Figure 5.15 illustrates the comparison results. Figure 5.15(b) and (d) have a "NET" (network IO) filed since the Sync (backup BG) is a network-bound application. For all comparison cases, Orchestra shows only 3% errors in CPU, 7% errors in disk IO (read), and 2% errors in disk IO (write), indicating that Orchestra is very accurate for FG applications. However, for the BGs, Orchestra shows slightly different statistics. Especially when Orchestra manages the Web application as a FG, it allows BGs to consume 17% to 20% more resources. This difference results in distinct execution time of BGs. With more resources, Orchestra could reduce the execution time of BGs (15% – 17% reduction for both BGs). The differences in BG's utilization are due to the following reasons. Orchestra mechanism to manage BGs naturally allocates more resources to the BGs if the FG's RT meets a SLA and the VM has residue resources possibly being consumed by the BGs. Furthermore, this is also related to the FG's workloads. The workloads generated by CloudSuite have higher variation than DB workloads from TPC-C. This high variance is not only relevant to the changes in the number of requests but also associated with the diverse types of the requests, i.e., from simple web renderings to complex social activities in web sites. Given the high fluctuating workloads, it is evident that Orchestra tries to minimize the execution duration of BGs and this is a clear indication that Orchestra works correctly under such workloads. Similarly, the higher amount of network TX in Figure 5.15(b) is because of the same reason that the BG (Sync) sends the larger amount of backup data per second to S3 with more allocated resources.

Accuracy of RT Estimator (FG): Next, we measure the accuracy for the RT

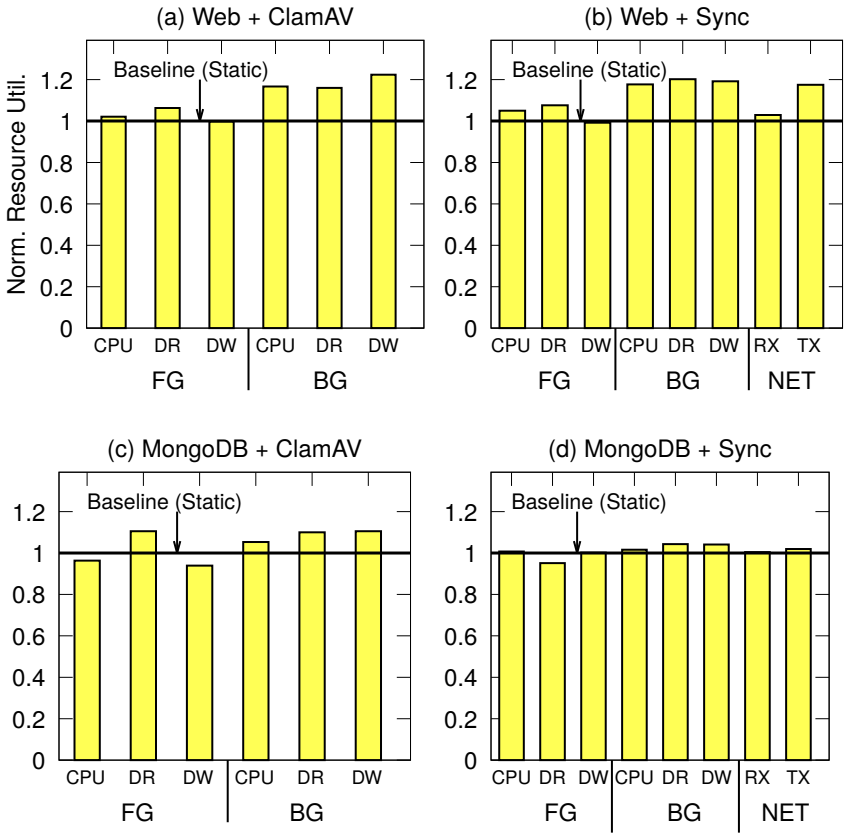


Figure 5.15: **Accuracy comparison of VM utilization between Orchestra and the static resource allocation. (NET: Network IO, DR: Disk Read bytes/sec., DW: Disk Write byte/sec., TX: Network TX bytes/sec., RX: Network RX bytes/sec.)**

estimator (described in Section 5.3.2) in Orchestra. We calculate the accuracy of the RT estimator by comparing its prediction results with the actual RT (the ground truth) from the FGs. The results are that Orchestra shows 0.17 of MAPE and 0.45 of RMSE.

To validate these results are sufficient, we compare the performance of the RT estimator with a SVM (Support Vector Machine)-based predictor. SVM [72] is a well-known classifier widely used in data mining and machine learning area, but it shows very robust performance for regression (prediction) problems, i.e., application performance modeling [36]. To optimize the performance of the baseline, we employ RBF (Radial Basis Function) as the kernel of the SVM and test a broad set of (soft-margin and kernel) parameters with a grid approach [20]. Figure 5.16 illustrates the comparison results of both models. The RT estimator of Orchestra outperforms the

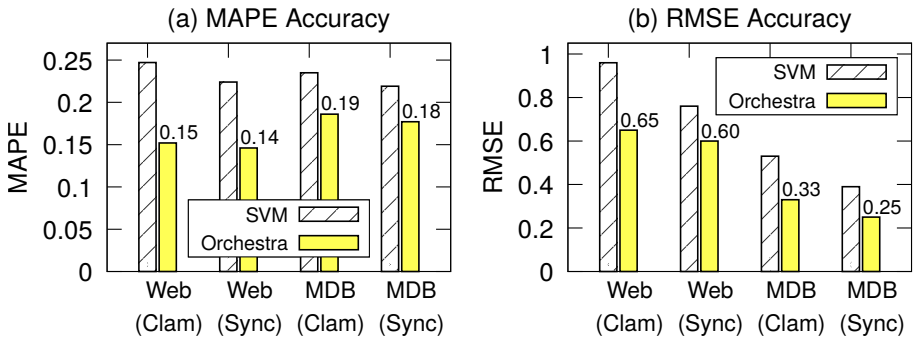


Figure 5.16: **Accuracy comparison of RT estimation of Orchestra with SVM. Lower values mean better accuracy.**

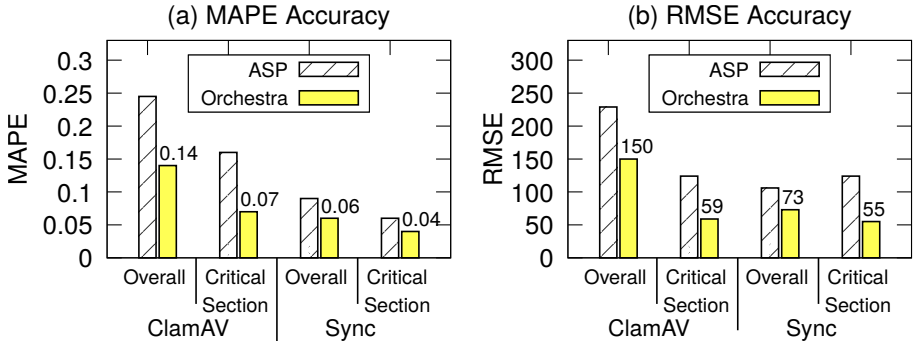


Figure 5.17: **Accuracy comparison of the BG performance model in Orchestra with the application-specific predictor.**

SVM and has approximately 40% lower prediction errors than the results from the SVM model. The SVM model just shows 0.23 of MAPE and 0.64 of RMSE.

Accuracy of BG Performance Model: We also measure the accuracy of the BG performance model (described in Section 5.3.3) in Orchestra. The model predicts the BGs' execution time whenever Orchestra changes the resource allocation to the FG and BGs. If the workloads have high fluctuations, Orchestra predicts the BGs' execution time more often. With considering a life-cycle of a BG's execution, the prediction model can show higher accuracy when the BG is approaching to the end of the execution. However, it is important for Orchestra to accurately predict the BGs' execution time (or finishing time), particularly when it reconfigures resource allocations to the BG at the middle of its life-cycle. We thus measure accuracy at two different points, i.e., 1) the overall accuracy – averaging all predictions during BG's execution and 2) the performance in the critical section. We define "30% – 70%" of a BG's execution as the critical section. For this evaluation, we employ an application-

specific predictor as a baseline. This predictor only relies on application-assisted hints (execution progress) and forecasts the BG's execution time by calculating the progress ratio.

Figure 5.17 shows the accuracy results. While both predictors show good accuracy, Orchestra outperforms the baseline in all cases. On average, Orchestra has 0.11 of MAPE and 112 of RMSE, indicating that it has 67% and 50% less error than the baseline. More interestingly, Orchestra produces more accurate predictions in the critical section, and it only makes 0.06 of MAPE and 59 of RMSE, showing 2x better accuracy than the overall results.

5.5.4 Orchestra Overheads

Orchestra's overhead is also a significant characteristic in the evaluation. Since Orchestra dynamically controls VM resources at runtime, it is also desired that the framework should not interfere the performance of FG and BGs. (Orchestra should not generate resource storms.) The overheads are measured from two different components in Orchestra, i.e., the managed node and the master controller.

Figure 5.18 reports the overhead (CPU usage) in Orchestra. The result measured in the managed node includes the overhead in performance monitoring from the sidecar, and resource usage monitoring and reconfiguring cgroups from the node agent. The master controller's results focus on the computational overheads for the management including prediction and optimization cost. The CPU consumption increases as the frequency of management operation in Orchestra increases. Orchestra consumes 1% – 5% of CPU resources on the managed node and 2% – 7% on the master controller. In the previous evaluation with FG workloads (Section 5.5.2), while Orchestra performs such monitor and control operations in every 5 seconds, it shows a desirable performance in managing both FG and BG's performance, indicating that 1% – 2% of CPU resources is sufficient to Orchestra.

5.6 Chapter Summary

Resource storms in enterprise cloud environments become a significant challenge for managing the performance of cloud applications. To improve this situation, we presented Orchestra, *cloud-specific framework for controlling both FG and BGs in the user space* to guarantee the FG's performance while minimizing the performance penalty

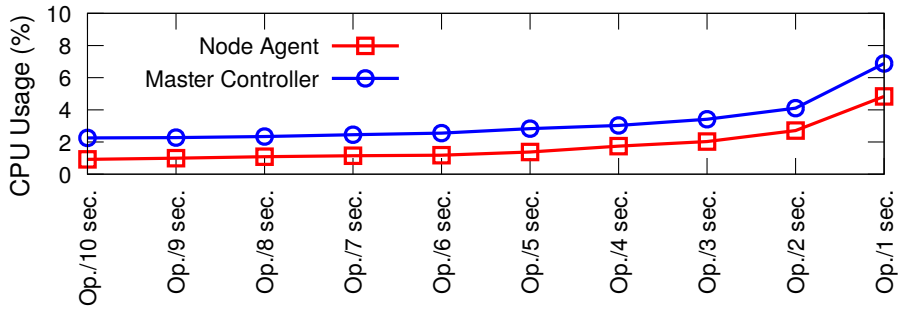


Figure 5.18: **Orchestra Overhead.**

of BGs. Orchestra measures a FG's performance (RT) in a real-time manner, and it creates a lightweight RT estimation and performance models for both applications on the fly. With the resource statistics and such predictive models, Orchestra optimizes the resource allocation to multiple cloud applications with SLA targets.

We have implemented and evaluated Orchestra with real workloads on Amazon EC2. Our primary workloads are a web service and a NoSQL database (MongoDB) for FGs and AWS Sync (backup) and ClamAV (virus and malware scanner) for BGs. We also presented the performance of Orchestra with a number of SLA constraints. Orchestra guarantees the FG's SLA satisfaction at all times with 70% performance improvement of BGs. Moreover, Orchestra shows a very high overall correctness (less than 5% error), 16.5% of MAPE for the FGs' response time estimation, and over 90% accuracy for the BGs' performance prediction.

Chapter 6

PICS: A Public IaaS Cloud Simulator

In chapter 3 and 4, we introduced a solution to address the workload uncertainty by designing a novel ensemble approach that predicts future workload changes in diverse cloud applications. Chapter 5 presented an application control framework that addresses the performance uncertainty caused by a modern cloud deployment model in enterprise clouds. This chapter discusses a solution for the last challenge in this research, which is the difficulty in the large-scale evaluation of cloud applications and infrastructure. We begin this chapter by discussing the concerns and questions from the cloud application providers/developers for this challenge, then describe the limitations of existing approaches proposed by research community and industry. After which, we introduce new cloud simulator that enables the users (application providers and developers) to evaluate diverse aspects/performance metrics of the cloud applications and infrastructure. Lastly, we validate the correctness of our simulator by comparing with the measurement results from deploying cloud applications on a production cloud (AWS).

6.1 Introduction

As we stated in Chapter 1, when cloud application developers and deployers evaluate the performance of large-scale cloud applications and public cloud infrastructure, the key limitation they are facing is that *there does not appear to be a viable alternative for evaluating the cloud other than to actually use the cloud*. This approach is problematic for a number of reasons. First, the time-consuming and sometimes tedious

learning of idiosyncratic cloud APIs can distract from the real issue, which centers around specific application logic and requirements. Second, the evaluation tends to be specific to one cloud and not readily generalizable to other clouds. Third, to evaluate at scale via this approach, the cloud-application typically requires significant changes to its architecture. Fourth, the evaluation is geared toward the present time, whereas longer-term issues/concerns are often more important than short-term issues of today's cloud – there is little opportunity to ask *what-if* questions of performance, reliability or cost.

There are a number of cloud simulators that exist (e.g. CloudSim [26], iCanCloud [157], GreenCloud [111]). They have the potential to aid in this evaluation. However, in general, these simulators are designed to answer questions related to datacenter management (e.g., how many concurrent users can I support if I deploy a private cloud on my existing hardware?) Furthermore, typical tools [5, 144, 172, 177] provided by commercial cloud vendors only address a small part of the concerns, which is an overall cost in the aggregate based on resource utilization (e.g., how much does it cost to run 100 small VMs for one month and to store 10 TB in long-term cloud storage for 1 year?) These existing cloud simulators and vendor tools do not broader, end-to-end concerns such as:

1. What is the average/worst response time for a particular application and a particular arrival pattern, when servicing via a specific VM type and a specific set of autoscaling rules?

2. Which public IaaS cloud provides the best cost efficiency for a particular application, given the different VM configurations, storage services and pricing models?

3. Which resource management and job scheduling policy maximize the cost efficiency and minimize the response time for a particular application?

4. Above all, if a simulator can provide answers for above questions, another question the cloud users could have is how reliable are the simulation results? or how accurately can the simulator resemble actual clouds' behavior?

To enable potential public IaaS cloud users to address these and other challenging concerns without *actually deploying the cloud-application*, we have created the PICS[1],

[1]Source code of PICS is publicly available at http://www.cs.virginia.edu/~ik2sb/PICS

a trace-based public IaaS cloud simulator. PICS provides following capabilities to address the potential cloud user's concerns:

- Assessing a wide range of properties of cloud services and the cloud-applications, including the cloud cost, job response time, and VM utilization.
- Allowing the simulation users to specify different workload types, including varying job arrival patterns and SLA requirements (e.g. deadline).
- Simulation of a broad range of resource management policies: i.e., horizontal/vertical[2] autoscaling, job scheduling and job failure policies.
- Enabling the users to evaluate the performance of different types of public IaaS cloud configurations such as a variety of resource types (VM and storage), unique billing models, and performance uncertainty [160, 178, 179].

We validated the correctness of PICS by focusing on following capabilities: cloud cost, the number of created VMs, VM utilization, horizontal and vertical scaling of cloud resources, and job deadline satisfaction rate. We compare the simulation results of PICS with the actual measurements from real-world cloud-applications on Amazon Web Services (AWS). The results show that PICS provides very accurate simulation results (less than 5% of average errors) in every validation cases. Furthermore, we conduct a sensitivity test of PICS with $\pm 10\%$ and $\pm 20\%$ of imprecise simulation parameter by considering of the performance uncertainty of IaaS clouds. The results show that PICS with imprecise simulation parameters still provides very reliable simulation results.

6.1.1 Chapter Organization

The rest of this chapter is organized as follows: Section 6.2 describes the design and implementation of PICS. Section 6.3 contains validation of PICS. Section 6.4 is discussion focusing on sensitivity of simulation parameters and Section 6.5 provides the summary of this chapter.

[2]Vertical scaling means "scale-up" or "scale-down" operations for the cloud resources. i.e., migrating the user-applications to higher (scale-up) or lower (scale-down) performance instances. [15]

6.2 Simulator Design

6.2.1 Simulator Design Overview

Goal: The goal of PICS is to correctly simulate the behaviors of public clouds from the cloud users' perspectives as if they deploy a particular cloud-application on public IaaS cloud. From the potential cloud users' perspective, the cloud cost, the job response time and the resource usages (VM utilization and size) are the most important criteria to evaluate cloud service for their cloud-applications. Key challenges to design PICS are:

- How to correctly model the behavior of public clouds. More specifically, how to handle a variety of resources (e.g. VM, storage, and network).
- How to properly model the behavior of the cloud-application. More specifically, how to handle varying workload patterns and performance uncertainty [160, 178, 179].
- How to correctly model the behavior of cloud users' resource management policy.

For the first challenge, we designed a convenient configuration interface for the simulation users to define diverse types of cloud resources as an input of the simulator. For the second challenge, we collected data from real public clouds, profile performance uncertainty, and leverage these results to design the simulator. For the last challenge, we provided abundant configuration options to let user define various custom resource management policies.

Input: PICS requires five types of inputs: VM configurations, storage and network configurations, workload information, job scheduling policies, and cloud resource management policies.

- *The VM configuration* includes detailed specifications of VMs, such as cost (including public cloud's billing model), machine performance (CPU, RAM), network performance, and the range of startup delays [130] of cloud resources. This configuration is designed to simulate various VM types of current public clouds because public clouds have a diversity of VM types based on performance, cost, and the VM's purpose.
- *The storage and network configuration* has detailed information on storage and network service on public IaaS clouds. We model storage services to reflect

current public clouds' actual characteristics based on Amazon S3 [9] and Azure Storage [143]. To model network service, we collect data from actual network-I/O test by using various types of VM on real cloud services. We then reflect the data to simulator configurations.

- *The workload information* contains detailed configurations on job generation such as job arrival time, job execution time, job deadline, size of network I/O, etc. This input reflects end users requests to a particular cloud-application.

- *The job scheduling policy* defines various cloud user job scheduling policy for end users' requests. PICS includes three types of job scheduling policies, i.e., EDF (Earliest Deadline First), Round-Robin, and Greedy scheduling mechanisms. In the future, PICS will support more complicated job scheduling policies and APIs. Furthermore, the simulation users can configure recovery policies for job failures, which enable the users to conduct realistic tests for public cloud services.

- *The cloud resource management policy* contains detailed specifications for the cloud resource management. This input supports simulation configurations for maximum number of concurrent VMs, and horizontal and vertical scaling policies. Moreover, the simulator users can configure various ways to monitor and analyze an end user's job request patterns such as linear and time-series methods. The simulator users are able to leverage this mechanism to design and test their own resource management mechanisms.

Output: PICS provides three types of output: cost, resource usage, and job processing results.

- *Cost results* provide overall cost for total cloud usage, cost trace at fine grained time interval, and usage cost per cloud resource type. Overall cost means how much cost the simulation users are expected to spend on servicing a particular cloud-application under a particular job arrival pattern. Cost trace provides fluctuation and accumulation of usage cost at fine grained time interval. For example, expected usage cost for time at t is \$100 and \$200 for time at $t+1$. The usage cost per cloud resource type provides a detailed cost based on resource types such as how much cost they spent on each type of resources (e.g. VM, storage, network).

- *Resource usage results* provide detailed information on resource usage such as how many VMs created, how much storage space spent, and how many network data sent and received. Moreover, these results offer fine-grained traces for both horizontal and vertical scaling. These traces help users determine the number and types of VMs running at time t and $t+1$. The users also check when (time) and how (scaling-up/down) the vertical scaling decisions are made.

- *Job processing* results provide specific information on job processing, such as the job arrival/start/finish time, as well as whether the job deadlines are satisfied (if specified). These results are basic metrics to evaluate the user job scheduling and resource management policies. Moreover, these results include the analysis of job arrival patterns (min, max, and average of job arrival time, as well as predictions for the next job arrivals) using linear and time-series methods.

6.2.2 Simulator Internals

PICS is composed of three hierarchical layers: simulation configuration layer, simulation entity layer, and simulation core layer as shown in Fig. 6.1. The simulation configuration layer is responsible for accepting the user inputs which are passed on to the simulation entity layer. The simulation entity layer contains the simulation logic and is driven by the events generated from the simulation core layer. The simulation core layer is also responsible for producing simulation reports. Because the simulation configuration layer has already been covered by the previous section, we focus on the two remain layers here.

Simulation Core Layer: The simulation core layer consists of Simulation Wall Clock, Simulation Event Processor and Simulation Report Generator.

1. **Simulation Wall Clock** is working as a heart for PICS by managing simulation clock. (Basic time unit is a second.) To manage the simulation clock, Simulation Wall Clock collaborates with Simulation Event Processor. When the simulation clock is updated, this component sends a clock update event to Simulation Event Processor to notify an update of clock.

2. **Simulation Event Processor** handles every event generated in a simulation. After receiving the clock update event from Simulation Wall Clock, this component passes on this event to simulation entities, which advance their simulation

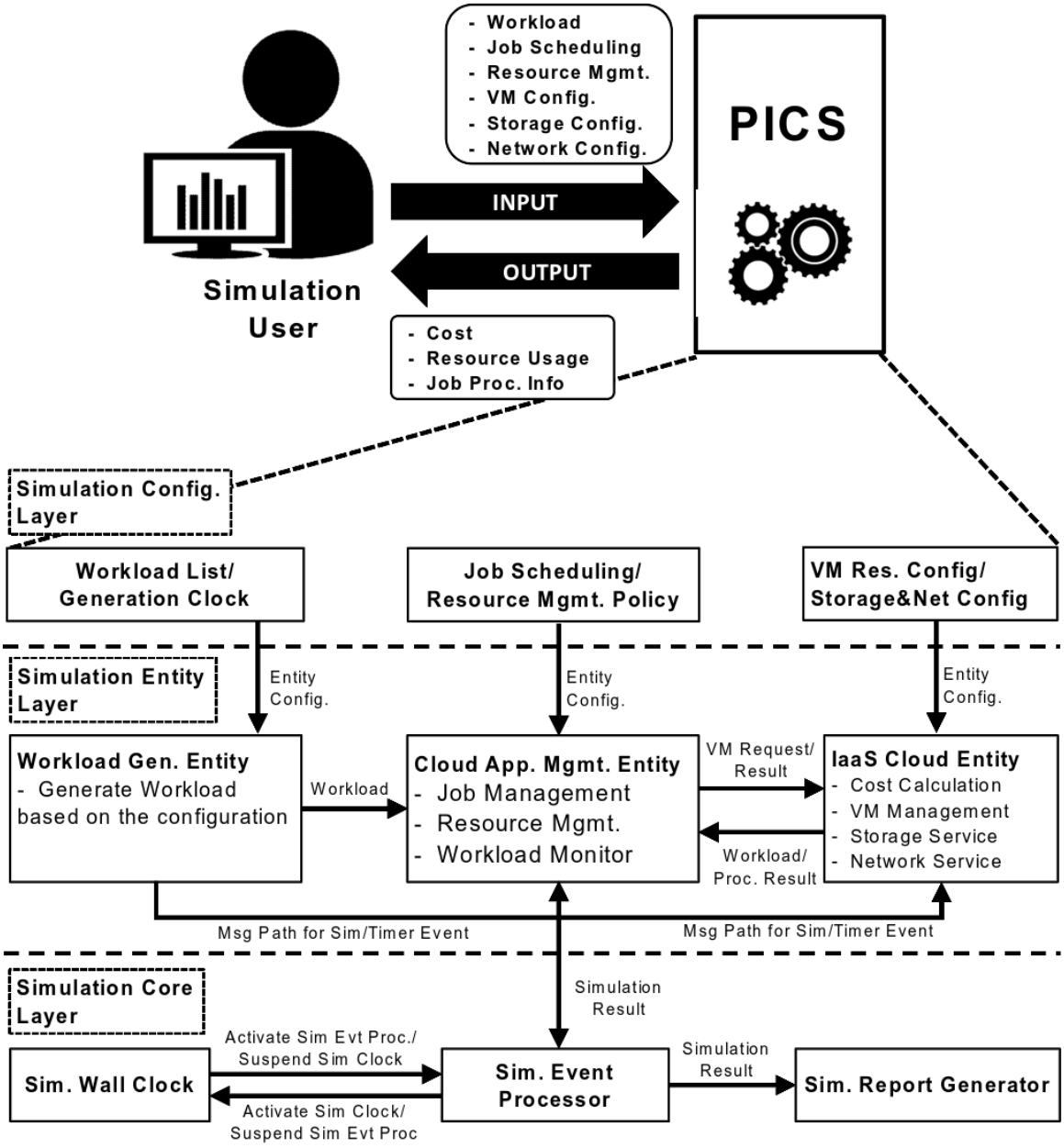


Figure 6.1: **Design overview of PICS.**

of the behaviors of public cloud services and cloud-application to the new clock cycle. This component also manages simulation timer events. Simulation entities use these events to register timers within Simulation Event Processor. This component is responsible for notifying the corresponding simulation entity when a timer expires. Moreover, this component handles all simulation command events, all invoked events between simulation entities, in order to validate the authentication of the command events. Because this component can monitor all events in the simulation, this plays a main role to generate real-time traces for the simulation.

3. **Simulation Report Generator** is used to generate output reports for the simulation results such as simulation trace and overall reports. Reporting simulation results at fine-grained time interval is an important capability for simulators. Simulation Report Generator is responsible for generating real-time traces called simulation trace reports. These reports contain simulation results at user-defined time interval for cost, resource scaling, and job processing information.

Simulation Entity Layer: Simulation entity layer are composed of three entities, which are Workload Generator Entity, Cloud Application Management Entity, and IaaS Cloud Entity. These entities reflect three main components of cloud-applications and public clouds.

1. **Workload Generator Entity** generates jobs and sends them to the Cloud Application Management Entity to process, based on the workload file from the simulation user. The workload file includes job arrival time, job execution time, job deadlines, I/O data specification, and data transfer specification to support various types of possible end user's job requests.

2. **Cloud Application Management Entity** is designed to resemble the cloud-application's behaviors. There are sub-components in this entity: job management module, resource management module and workload monitoring module.

    - *The job management module* is designed to simulate job management policies of a cloud-application. This module conducts three operations for the simulation; job scheduling, job response time estimation, and job failure management. Job scheduling is used to perform job scheduling policies of

the cloud users and assign an arrived job from Workload Generator Entity to a selected VM from IaaS Cloud Entity. Job response time estimation predicts the job response time, which is defined as the clock time when a particular job finishes. The prediction of job response time is based on job execution time and the current load of available cloud resources, and is used for the job scheduling. Job failure management is used for job failure simulation in the case of application failure or cloud infrastructure problem (e.g. VM down). Job failure management supports four types of recovery policies for job failure simulations.

- *The resource management module* is designed to simulate the resource management policies of a cloud-application. It handles three types of cloud resources of public IaaS clouds, i.e., VM, storage and network. For the VM management, this module enables the simulation users to examine their VM selection mechanisms (e.g. cost, performance, cost/performance-balanced) and VM scaling mechanisms (e.g. horizontal, vertical scaling). For storage and network resources, this module can simulate File Read/Write operations to cloud storage and data transmissions by collaborating with IaaS Cloud Entity.
- *The workload monitoring module* is designed to analyze workload arrival patterns from Workload Generator Entity. The simulation users can leverage this module to improve their policies for job scheduling and resource management for variable workload patterns.

3. **IaaS Cloud Entity** is used to simulate the public cloud's behavior. It has sub-modules to simulate public clouds, which include cost calculation, VM management, storage service and network service module.

    - *The cost calculation module* calculates all cloud cost used by Cloud Application Management Entity. It generates the cost traces based on user-defined time interval and creates the final results when the simulation is completed.
    - *The VM repository module* manages the configurations of all VM types defined by the users and resembles the on-demand VM service of IaaS clouds. This module stores VM information to correctly measure VM usage cost and simulate job execution on VMs. Moreover, this module

generates startup-delay for new VM creations based on user input, and simulates all operations of the VMs. This module also handles workloads on VM such as job execution and failure generation.

- *Storage service and network service module* simulates file I/O and data transmission operations based on their configurations and the workload information. It generates the overall and real-time traces of the usage of storage and network services for the workloads.

6.3 Simulator Validation

6.3.1 Experiment Setup

In order to validate the simulation results of PICS, we compared PICS results with a real cloud application on AWS. We design and implement a cloud-application that executes user-applications with three different types of MapReduce [41] jobs and two job arrival patterns. The workflow of the cloud-application goes through the following five key steps: 1) Job execution time prediction via recent execution history, 2) EDF job scheduling, 3) Cost-based VM selection, 4) Deadline-based horizontal and vertical autoscaling, and 5) Job Execution.

The cloud-application starts with receiving jobs from the end users. It conducts **job execution time prediction via recent execution history** for incoming jobs. The cloud-application schedules them by the **EDF job scheduling** and sends them to the work queue in the VMs of choice. For VM selection, this cloud-application uses **cost-based VM selection** that selects the cheapest VM type that meets the deadline for a job. For the VM scaling, the cloud-application makes the scaling decision based on the **deadline-based horizontal and vertical autoscaling**. The cloud-application first determines the fastest possible response time for the new job based on the load of the job queues of active VMs. If the fastest possible response time still misses the job deadline, horizontal or vertical scaling is engaged. In the case of horizontal scaling, a new VM is created for the new job. In the case of vertical scaling, an active VM is "scaled-up" to a higher performance VM for the new job. Note that vertical scaling happens when there is a limit on the number of active VMs which prevents the creation of new VMs. Additionally, after each job finishes, the cloud-application checks active VMs. If the active VMs provide more than enough computation power to satisfy deadline requirements, then a "scale-down" happens

to improve the cost efficiency. Once the current VM type is determined, the job is assigned to that VM for **job execution**.

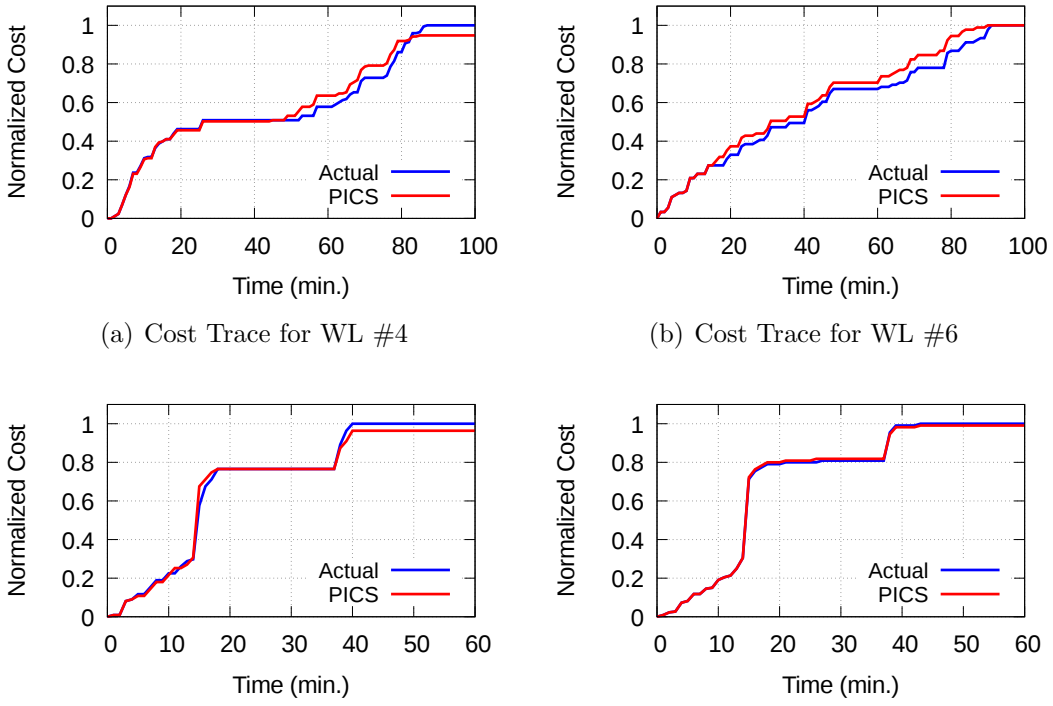
For the validation of PICS, we use three types of MapReduce jobs (Word count, PI calculation, and Terasort) [13]. Word count is an I/O- and memory-bound job, and it uses S3 storage to download input dataset and upload the final result. PI calculation is a fully-parallelized CPU-bound job. Terasort is a standard Hadoop benchmark application. These jobs were randomly generated based on two arrival patterns (Poisson and Bursty). Poisson arrival pattern has an average job arrival interval of 30 seconds with a standard deviation of 29.63. Bursty pattern has an average job arrival interval of 15 seconds with a standard deviation of 11.50.

We deployed the cloud-application on AWS. The reasons that we use AWS for the PICS validation are as follows: 1) AWS is widely used public IaaS cloud; 2) according to recent works [120, 160, 178, 179], AWS EC2 performance fluctuates a lot and AWS has less predictability (higher variance) than other public IaaS clouds, therefore AWS is better than other IaaS clouds when evaluating the sensitivity of PICS to the performance uncertainty of public clouds. Our cloud-application uses four types of on-demand EC2 VM instances, which are *m3.medium*, *m3.large*, *m3.xlarge*, and *m3.2xlarge*. These four types of EC2 on-demand instances are general purpose VMs and commonly used by the cloud users.

Based on the above experimental configurations, we created 16 validation workloads as shown in Table 6.1. These workloads are categorized based on job arrival patterns (Poisson and bursty), job types, single or multiple VM types, and scaling policies. WL #1 – #6, #13, and #14 are for the tests under Poisson job arrival pattern, and others are for bursty arrival pattern. WL #1, #2, #7, #8 and #13 process word count jobs, WL #3, #4, #9, #10 and #14 handles PI calculation jobs, and WL #5, #6, #11 and #12 deal with Terasort jobs. WL #1, #3, #5, #7, #9, and #11 only use a single VM type (e.g. *m3.medium*, *m3.large*, or *m3.xlarge*) in order to validate a case when a cloud-application uses a single type of VM. The others use all four types of general purpose EC2 instances. This is to test more complicated use cases for the VM resource management. For scaling validation, WL #1 – #12 are for horizontal scaling and WL #13 – #16 are for vertical scaling use cases. We submitted these 16 workloads to PICS and the cloud-application running on AWS, and measured the cloud cost, the total number of created VMs, VM utilization, and job deadline satisfaction rate. These metrics are expressed as equation – (6.1), (6.2), and (6.3). We then measured the simulation error by equation – (6.4).

Table 6.1: **Validation Workloads for PICS. Poisson pattern has 30s of average job arrival rate and 29.63 of standard deviation. Bursty pattern has 15s of average job arrival rate and 11.50 of standard deviation. (WC: Word Count, PI: PI Calculation, TS: TeraSort)**

Worloads	Scaling	Job Arrival Patterns	Job Type	# of Concurr. VMs	VM Types	# of Jobs	Avg. Job DL	Std. Dev
WL #1	Horizontal Scaling	Poisson	WC	Unlimited	m3.medium	200	272s	129.55
WL #2					All Types			
WL #3			PI		m3.large		538s	266.10
WL #4					All Types			
WL #5			TS		m3.xlarge		1065s	531.09
WL #6					All Types			
WL #7		Bursty	WC		m3.medium		500s	127.60
WL #8					All Types			
WL #9			PI		m3.large		515s	263.50
WL #10					All Types			
WL #11			TS		m3.xlarge		1102s	559.75
WL #12					All Types			
WL #13	Vertical Scaling	Poisson	WC	3	All Types	500	510s	265.03
WL #14			PI	7			1029s	556.50
WL #15		Bursty	WC	5			523s	279.38
WL #16			PI	10			1045s	563.15



(a) Cost Trace for WL #4

(b) Cost Trace for WL #6

(c) Cost Trace for WL #8

(d) Cost Trace for WL #10

Figure 6.2: **Cost trace for horizontal scaling cases.**

$$Cost = \sum_{i=1}^{n} cost_{VM_i} \tag{6.1}$$

$$VM\ Utilization = \frac{\sum_{i=1}^{n} Time_{JobExec.,VM_i}}{\sum_{i=1}^{n} Time_{TotalRun.,VM_i}} \tag{6.2}$$

$$Job\ Deadline\ Satisfaction\ Rate = \frac{N_{DeadlineSatisfiedJobs}}{N_{AllJobs}} \tag{6.3}$$

$$Simulation\ Error = \left| \frac{Actual - Simulation}{Actual} \right| \times 100\% \tag{6.4}$$

6.3.2 Horizonal Scaling Cases

We use 12 workloads (WL #1 – #12) for the validation of horizontal scaling cases. The "**Cost**" column of Table 6.2 shows PICS simulation error of the overall cloud cost over the actual measurements on AWS. The average error of the cloud cost from PICS is only 2.6% compared to the actual results. The highest error is only 6.1% (WL #12). A more important metric is the cost trace because it shows how accurately PICS

Table 6.2: **Simulation errors in horizontal scaling cases.**

Workloads	Cost	# of VMs	VM Util.	Job DL.
WL #1	3.1%	3.4%	1.1%	0.6%
WL #2	2.4%	1.8%	1.4%	1.1%
WL #3	1.8%	2.3%	0.7%	0.6%
WL #4	5.2%	4.7%	2.2%	3.2%
WL #5	2.2%	7.1%	1.6%	2.1%
WL #6	0.8%	2.4%	2.4%	2.6%
WL #7	2.6%	3.6%	0.5%	2.5%
WL #8	3.6%	0.0%	1.0%	1.1%
WL #9	1.5%	1.5%	0.7%	0.6%
WL #10	0.9%	1.5%	0.9%	3.9%
WL #11	1.0%	0%	2.2%	0.5%
WL #12	6.1%	1.4%	1.3%	3.8%
Average Error	**2.6%**	**2.4%**	**1.3%**	**1.9%**

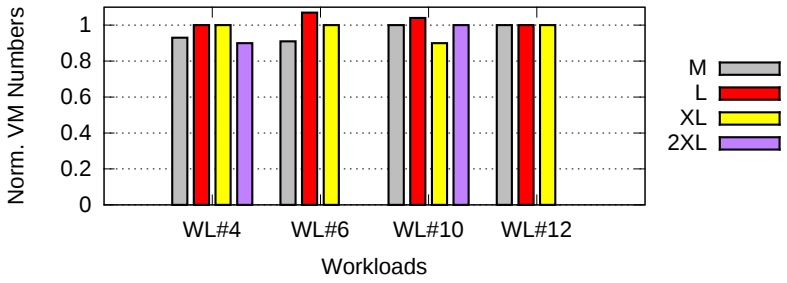


Figure 6.3: **The VMs numbers per VM type, normalized over the actual measurement (M: m3.medium, L: m3.large, X: m3.xlarge, XL: m3.xlarge)**

simulates the behavior of the cloud-application and public IaaS cloud at fine-grained intervals. To show these traces, we select four complicated workloads, which are WL #4, #6, #8, #10. Figure 6.2 shows these four cost traces. The other workloads have similar results. As shown in Figure 6.2, PICS is able to accurately calculate the cloud cost at each time interval, demonstrating that PICS correctly resembles the behavior of the cloud-application and public IaaS cloud for each step of execution.

The simulation error of created VM numbers over the actual measurements are shown in the "**# of VMs**" column of Table 6.2. PICS can highly accurately calculate the number of created VMs with an average error of 2.4%. The highest error is only 7.1% (WL #5). For the workloads that use multiple types of VMs, a precise simulation of the number of VMs for each VM type is also critical to the cloud users

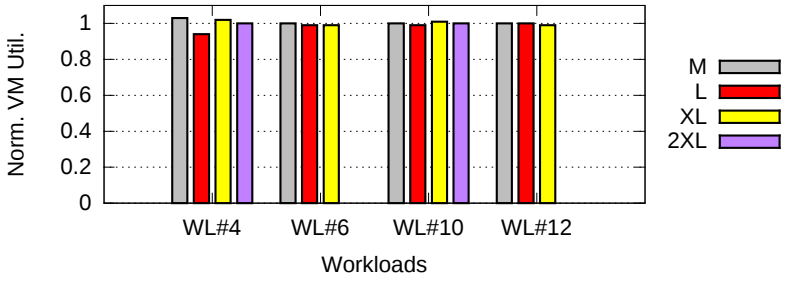
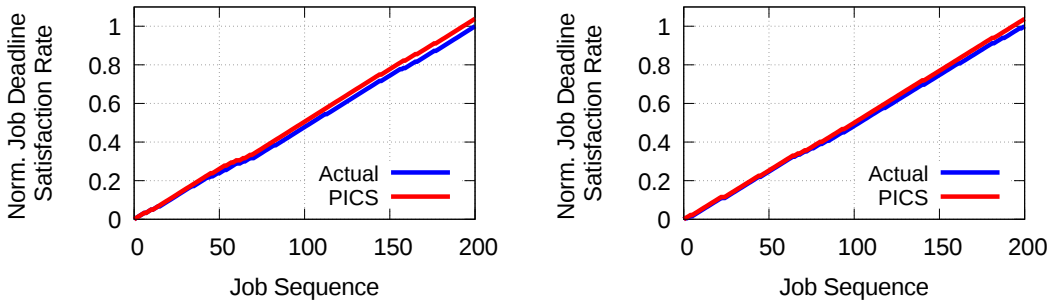


Figure 6.4: **The VM utilizations per VM type, normalized over the actual measurement (M: m3.medium, L: m3.large, X: m3.xlarge, XL: m3.xlarge)**



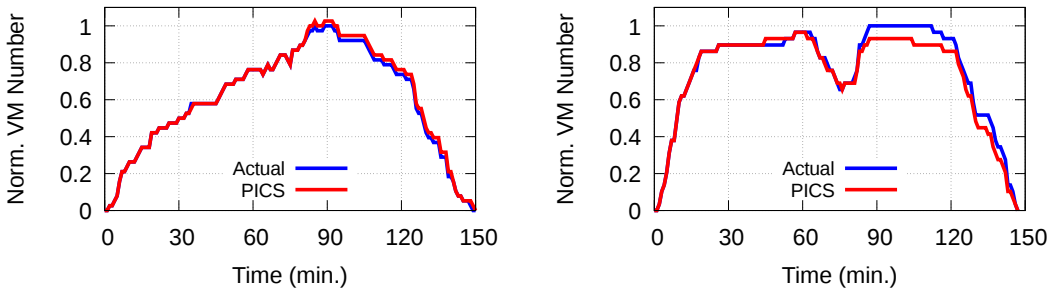
(a) WL #10 (b) WL #12

Figure 6.5: **Job deadline satisfaction traces for horizontal scaling cases.**

to determine if PICS can accurately resemble the cloud-application's resource management policies (VM selection and scaling) and public cloud's behaviors. Figure 6.3 shows the result of the number of created VMs for each VM type for four workloads (WL #4, #6, #10, #12). The other workloads have similar results. As Figure 6.3 shows, PICS can also accurately simulate the VM numbers for each type of VMs. For the horizontal scaling cases, the cloud-application created an average of 42.5 VMs and the range of the created number of actual VMs is from 15 (WL #5) to 70 (WL #12).

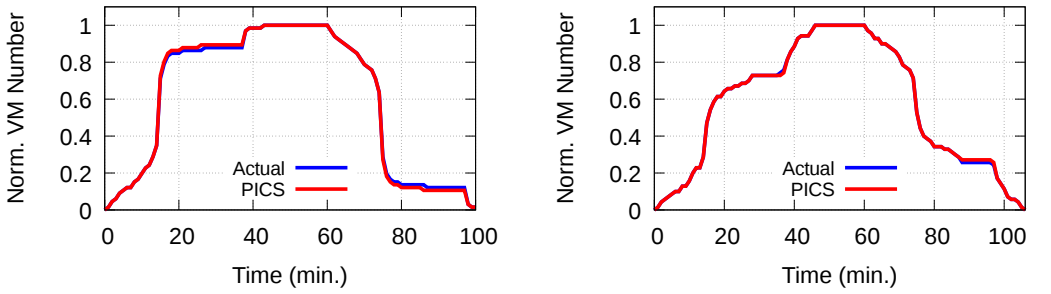
Similarly, the overall simulation error of VM utilization are shown in the "**VM Util.**" column of Table 6.2, while the detailed per-VM type utilization results are shown in Figure 6.4. Average error of overall VM utilization results between the actual measurements and the simulations is only 1.3%. For the detailed results, PICS is able to accurately simulate the utilization for each VM type with 0.5 – 2.4% of simulation error for most cases. The worst case is *m3.large* instance of WL #4 and the simulation error is only 5.8%.

In addition, we conducted a validation focusing on job deadline satisfaction rate.



(a) VM scaling trace for WL #4

(b) VM scaling trace for WL #8



(c) VM scaling trace for WL #10

(d) VM scaling trace for WL #12

Figure 6.6: **Horizontal VM scaling traces.**

Deadline satisfaction rate is an important metric for a cloud-application to ensure the cloud-application's job scheduling and resource management policy meets certain temporal requirements, such as those found in SLA. We measured the job deadline satisfaction rates of PICS, and compared them with the results of the actual cloud-application on AWS. Note that the deadlines in our workload are generated randomly. The overall results are shown in the "**Job DL.**" column of Table 6.2. The average error of PICS is only 1.9%. The worst case error is about 4% (WL #10, #12). More importantly, we measured the traces for job deadline satisfaction because we want to determine if PICS can precisely simulate whether a particular job satisfies its deadline or not. Figure 6.5 shows the job traces for two worst workloads (WL #10, #12). Figure 6.5 shows that, even for the worst cases, PICS accurately simulates the behavior of AWS at every phase of the execution.

We also measured the VM scaling of the actual cloud-application on AWS and PICS, over the whole course of the execution. Due to space limitation, Figure 6.6 shows the scaling traces for only four workloads (WL #4, #8, #10, #12). The other workloads have similar behaviors. As shown in Figure 6.6, traces from PICS and the actual cloud-application closely match each other, which means PICS accurately

simulates horizontal scaling at every phase of the execution. For the VM scaling traces, the average number of running VMs in parallel (by the cloud-application) is 39 and the cloud-application runs from 12 (WL #5) to 70 VMs (WL #12) in parallel.

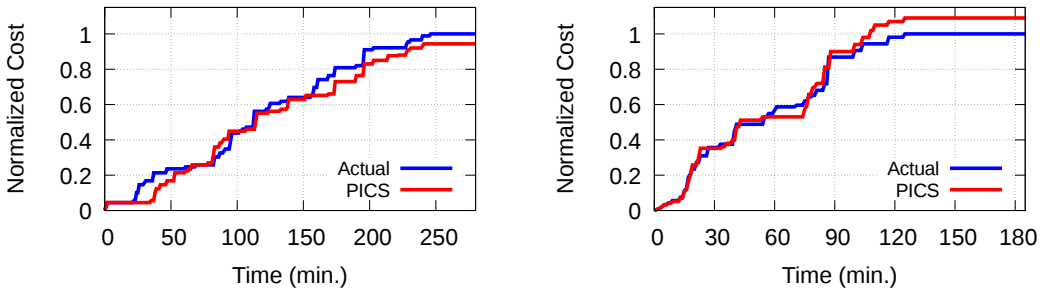
6.3.3 Vertical Scaling Cases

Due to the importance of vertical scaling in the near future, PICS also supports vertical scaling simulation. We validate the results involving vertical scaling using four workloads (WL #13 – #16). Note that for these experiments, both horizontal and vertical scaling are enabled, similar to real-life usages of cloud services. Table 6.3 shows the overall simulation errors for vertical scaling cases. The results show that PICS can accurately simulate the overall results for vertical scaling workloads: for the cloud cost, the average error is only 5.5% ("**Cost**" column); for the number of created VMs, the average error is only 3.6% ("**# of VMs**" column); for the VM utilization, the average error is only 2.9% ("**VM Util.**" column); for deadline satisfaction, the average error is only 2.6% ("**Job DL.**" column).

Table 6.3: **Simulation errors in vertical scaling cases.**

Workloads	Cost	# of VMs	VM Util.	Job DL.
WL #13	6.1%	7.1%	4.3%	0.8%
WL #14	3.1%	1.9%	2.4%	4.6%
WL #15	3.2%	3.4%	1.7%	1.9%
WL #16	9.7%	1.9%	3.3%	3.2%
Average Error	**5.5%**	**3.6%**	**2.9%**	**2.6%**

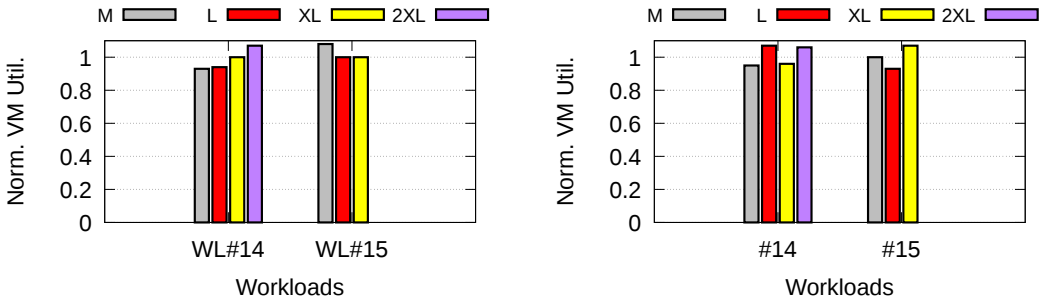
To demonstrate the accuracy of PICS simulation at fine-grained time intervals and resource types, we present the detailed results for these measurements, which are the cost traces, the number of created VMs per VM type, the VM utilizations for each VM type, and the deadline satisfaction traces. Figure 6.7 shows the cost traces for WL #13 and #16, which are two worst cases for the vertical scaling validations. Even for these two worse cases, the cost trace results from PICS closely resemble with the actual measurements. Figure 6.8(a) shows the normalized results for the number of created VM for other two workloads (WL #14, #15). The results show that PICS has only 8.3% simulation error in the worst case (*m3.medium* instance of WL #15). Figure 6.8(b) represents the utilization for each VM type. In the worst cases (*m3.large* of WL #14), PICS has 7.3% simulation error. Figure 6.9 gives the



(a) VM Cost Trace for WL #13

(b) VM Cost Trace for WL #16

Figure 6.7: **Cost traces for vertical scaling cases.**



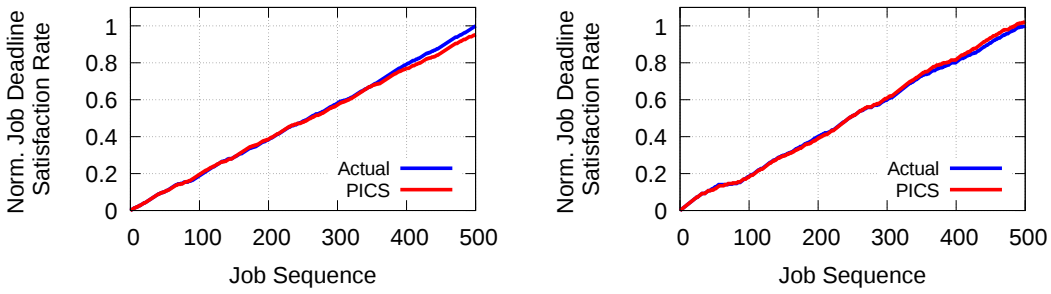
(a) Vertical Scaling VM Numbers

(b) Vertical Scaling VM Utilization

Figure 6.8: **The VM numbers (a) and utilizations (b) per VM type, normalized over the actual measurement (M: m3.medium, L: m3.large, X: m3.xlarge, XL: m3.xlarge)**

job deadline satisfaction rate traces for the two worse cases workloads (WL #14 and #16). Both the PICS and real measurement trace curves in Figure 6.9 closely match each other. Based on the results in Figure 6.7 to 6.9, we can conclude that PICS can also accurately simulate the detailed results of the cloud cost, the VM creation, the VM utilization and the deadline satisfaction rate for each VM type and at fine-grained time intervals.

For the last validation of the vertical scaling cases, we measured the number of vertical scaling decisions for the four workloads. These results are used to show how PICS accurately simulates the vertical scaling operations. The results are shown in Figure 6.10. For the total number of vertical scaling decisions, PICS has only 3.5% average error. Average error for scaling-up is 6.7% and average error for scaling down decision is 6.3%. PICS also has less than 10% of simulation errors in every scaling decision for all four workloads. These results imply that PICS can precisely simulate the vertical scaling operations of the cloud-application on real public cloud. For the



(a) WL #14 (b) WL #16

Figure 6.9: **Job deadline satisfaction traces for vertical scaling cases**

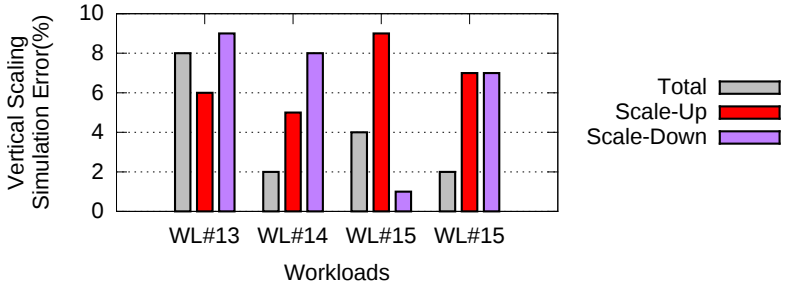
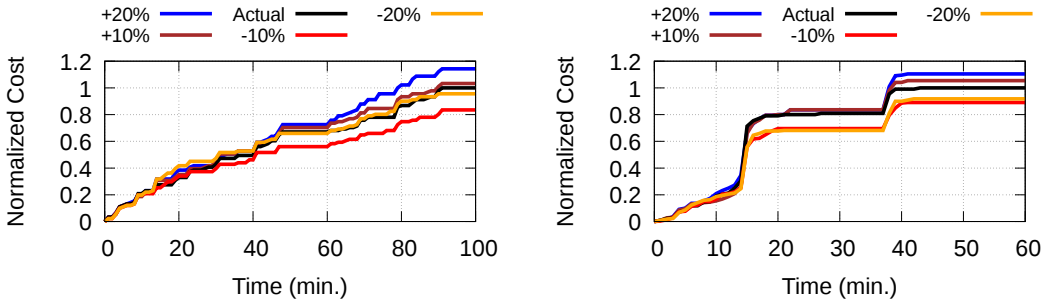


Figure 6.10: **Simulation errors for the numbers of vertical scaling decisions.**

best of our knowledge, PICS is the first cloud simulator that supports the vertical resource scaling simulation.

6.4 Discussion

The previous section demonstrates that PICS accurately simulates the behavior of cloud-application and public IaaS. However, the accuracy of PICS depends on the accuracy of user-provided parameters and configurations. Although users can provide accurate values for most parameters and configurations with ease, one parameter – the job execution time – may be difficult to acquire precisely. The difficulty comes from the performance uncertainty of real public clouds [160, 178, 179]. In our experiments, we used the average execution time from 800 samples (i.e., 800 executions on AWS) for each job type and VM type. However, in real practice, it may not be feasible for the users to acquire such high number of samples. That is, the users may provide inaccurate job execution time to the simulator. In this section, we analyze the impact of imprecise job execution time on simulation accuracy. More specifically, we seek the answers to the following questions:



(a) WL #6 with imprecise parameter

(b) WL #10 with imprecise parameter

Figure 6.11: **Cost traces of simulations with ±10% and ±20% of imprecise job execution time parameter.**

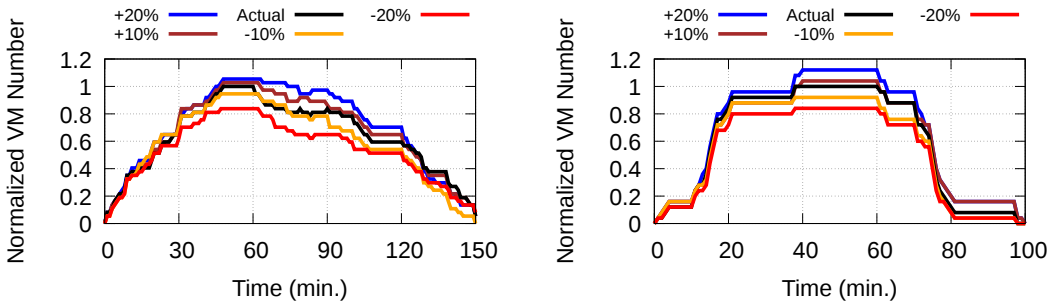
1. How much error should we expect from the use-provided parameter of job execution time?
2. What is the accuracy of PICS if the parameter of job execution time has certain errors?

Table 6.4: **Simulation Errors when the job execution time parameter has ±10% and ±20% errors.**

Error in Parameters	Cost	# of VMs	VM Utilization	Job Deadline
-20%	16.2%	13.5%	0.9%	5.96%
-10%	7.5%	6.3%	0.9%	4.25%
+10%	4.6%	4.7%	0.2%	3.31%
+20%	13.8%	11.7%	1.9%	2.01%

To answer the second question, we simulated the 16 workloads again using job execution times with ±10% and ±20% errors. These errors represent the aforementioned expected and maximum errors from user inputs. Table 6.4 shows the average errors for the 16 workloads with imprecise job execution time parameter in PICS. Fig 6.11 and Fig 6.12 show the cost and horizontal scaling traces of selected two workloads with imprecise parameters. (The other workloads have similar results.) These table and figures show that the errors of PICS are *considerably smaller than the errors in the job execution time parameter, and PICS retains high accuracy even when user provides imprecise job execution time parameter.*

We observed that PICS has lower errors than the parameter of job execution time for two reasons: *The first reason is that the running times of low-load VMs are less susceptible to input errors.* Because of we used large workloads with varied job arrival



(a) WL #6 with imprecise parameter

(b) WL #11 with imprecise parameter

Figure 6.12: **Horizontal VM Scaling Traces of simulations with ±10% and ±20% of imprecise job execution time parameter.**

times, many VMs have only a small number of jobs to execute. These VMs are usually created during periods with low job arriving rates. The running times of these low-load VMs have large fluctuations due to real cloud's unstable performance. Because of this fluctuation, the running time of many low-load VMs is close to the simulation based on the imprecise job execution time. Moreover, the execution time of a low-load VM is also considerably affected by the VM start-up time, which further reduces the impact of the imprecise parameter. *The second reason is the horizontal "scale-in" policy of our resource manager.* Our resource manager keeps VMs alive for some time in the anticipation of new jobs. Thus, the total running time of a VM is longer than the total execution time of its jobs, which reduces the impact of imprecise parameter of the job execution time.

In summary, due to the performance uncertainty in real clouds, user-provided job execution time usually has less than 10% errors, with a maximum error of 22%. With these potential input errors, the PICS simulator can provide reliable results to help users to assess their cloud application and services.

6.5 Chapter Summary

In order to answer for potential cloud users' questions about *evaluating the public clouds without actually deploying the cloud-application*, we have created PICS, a public IaaS cloud simulator. PICS provides the capabilities for accurately evaluating the cloud cost, resource usages (VM scaling, utilization, and the number of VMs), and job deadline satisfaction rate. In this chapter, we describe the configurations and architecture of PICS, and validate the accuracy of PICS by comparing it with actual

measurements from a real cloud-application on real public IaaS cloud. The validation results show that PICS very accurately simulates the behaviors of the cloud-application and public IaaS clouds (with less than 5% of average errors). Moreover, we show the sensitivity of PICS with an imprecise simulation parameter (job execution time with ±10% and ±20% errors). The results show PICS still provides very reliable simulation results with the imprecise parameter. These results demonstrated that PICS is both versatile and reliable for cloud user to *evaluate the public clouds without actually deploying the cloud-application*.

In the near future, we will have more comprehensive validations of PICS with three different directions. The first direction is to show the correctness of PICS with cloud-applications on other public cloud services such as Microsoft Azure [144] and Google Compute Engine [67]. Validating PICS on the different cloud providers is particularly important to demonstrate the simulation correctness and generalizability of PICS, because these cloud providers have different performance of VMs and cloud services (e.g. storage and network) as well as different cost models for cloud usages (e.g. minute-based billing model). The second direction is validating PICS with other cloud-applications and resource management policies. We will use n-tier web and scientific/big-data analytics applications with various management configurations because these are the common cloud-application deployment models for industry and research community. Furthermore, we will validate PICS based on the other metrics such as storage and network I/O usage.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

Because public IaaS cloud provides many attractive capabilities, it has become a standard computing infrastructure that executes diverse applications including web search, social network, multimedia streaming, business, and scientific applications and services. Maximizing SLA satisfaction (or application performance target) with minimizing cloud cost is a primary objective of managing cloud resources. Various mechanisms, such as reactive autoscaling and predictive scaling, have been proposed from industry and research community to achieve this goal with elastic resource management. However, these approaches are still insufficient to determine *when or what to scale* cloud resources and often has resource under-/over-provisioning that results in poor SLA satisfaction and cost efficiency.

The workload uncertainty is because of dynamic nature of cloud workload patterns. Based on the analysis of real-world workload traces, it turns out that cloud workloads are actively fluctuating over time and are highly likely to be composed of interleaving short-lived/bursty patterns that have different characteristics. The performance uncertainty can be caused by several reasons. The most common causalities are multi-tenancy, hardware heterogeneity in data centers, and co-running applications/containers on cloud resources. Among these reasons, the performance uncertainty/interference from co-running applications/containers is becoming the very common phenomenon in enterprise clouds due to increasing popularity of container technologies. Unfortunately, addressing the co-running applications/containers is inadequately researched in the research community and industry. In addition to these two uncertainties, the last challenge most cloud application service providers face

when they develop cloud applications and a resource management system is difficulty in a large-scale evaluation of such management mechanisms, applications, and infrastructures. This challenge is because there does not appear to be a viable alternative for evaluating the cloud other than to actually deploy the applications and use the public clouds.

This dissertation solves these three problems and introduces new resource management mechanisms that ensure the predictable end-to-end performance of cloud applications. Both Chapter 3 and 4 focus on addressing the workload uncertainty, Chapter 5 provides a solution for the performance uncertainty, and Chapter 6 presents a new evaluation platform that supports the large-scale evaluation for cloud applications and resource management mechanisms. More specifically,

- Chapter 3 performed a holistic evaluation of existing workload predictors and predictive scaling styles. In this chapter, we collected well-known existing workload predictors from an extensive literature survey and quantified the performance of all 21 predictors based on the model accuracy and overhead. Moreover, we evaluated the benefits/performance from different predictive scaling styles (e.g., predictive scaling-out, predictive scaling-in, or both) and determines the best scaling styles for cloud resource management. This chapter was presented in 2016 IEEE International Conference on Cloud Computing [108].

- Chapter 4 presented CloudInsight– an online workload prediction framework that addresses dynamic and highly variable cloud workloads. CloudInsight relies on multiple workload predictors of choice, and it periodically creates an ensemble workload prediction model of them with dynamic predictor selection and weight decision, calculated at runtime based on the predictor's accuracy for the current workload at previous time intervals. To select predictors with proper weight, the chapter also described a novel decision mechanism based on a SVM multiclass regression model. Part of this chapter is accepted for publication in 2018 IEEE International Conference on Cloud Computing [106].

- Chapter 5 confirmed and quantified the impact of one of the major causalities – *resource storm* – in performance uncertainty. and presented Orchestra framework, a cloud-specific framework for controlling multiple cloud application in user-space to guarantee the performance SLA requirements. Orchestra employs real-time, lightweight monitoring agents that collect diverse aspects of application performance and resource consumption, then creates performance models

for target applications. Orchestra also contains optimizer for VM resource allocations (e.g., CPU, memory, IO, and network) and controls the resources to multiple target applications by leveraging the knobs provided by modern OS such as control groups. Part of this chapter is accepted for publication in 2018 IEEE International Symposium on Parallel and Distributed Computing [110].

- Chapter 6 discussed PICS, which a new cloud simulator to support various and trustworthy evaluation of cloud applications, resource management, and cloud infrastructure. PICS supports diverse resource management mechanisms (e.g., horizontal and vertical scaling) for diverse cloud resources like VM, storage, network, and others. We thoroughly validated PICS' correctness by comparing the simulation results with the measurement dataset from actual cloud applications (MapReduce application and its management system) and infrastructure (AWS). Furthermore, we conduct a sensitivity test of PICS with imprecise simulation parameter by considering of the performance uncertainty of IaaS clouds. The results show that PICS with imprecise simulation parameters still provides very reliable simulation results. This chapter was presented in 2015 IEEE International Conference on Cloud Computing [104].

7.2 Future Work

7.2.1 Support for Cloud Function and Serverless Architecture

Regarding the resource management dimensions, this work focuses on the IaaS cloud resources – VMs and containers – the computing platform for current cloud applications. Today, cloud functions and serverless architectures – AWS Lambda [8], Azure Functions [142], Google Cloud Functions [66], Open Lambda [74], PyWren [98, 99] – have great attention from the research and development community as the next evolution of cloud computing. In particular, cloud functions and serverless architectures have:

- No requirement for computing infrastructure that reduces cloud engineer's workload and burden to manage cloud instances like VMs.
- Fewer worries for over-provisioning due to finer-grained resource and application provision.

- Less execution cost (cloud cost) because users pay the cost while only functions execute.

The future direction of this research is to improve the mechanisms presented in this dissertation to support the new evolution of cloud computing and address new research problems. The emerging research problems in this new domain are function/event scheduling, launching overhead, resource scaling, high-latency compatibility with legacy application models. Some of these problems can be directly addressed by this dissertation work without any modifications, but there are lots of research opportunity based on this dissertation work. The focus will be enhancing usability in cloud function/serverless architecture such as programming models and APIs, debugging and testing, and deployment models.

7.2.2 Assuring Application Performance for Large-scale Data Science Pipelines

The next direction of future work is to create a new cloud-scale infrastructure that assures predictable application performance for large-scale data science pipelines. This dissertation work is highly relevant to this future research direction. Emerging infrastructure management systems for data sciences pipelines, such as Mesos [82], Kubernetes [116], Docker Swarm [52], and others, highly require capabilities of accurate workload prediction (CloudInsight in Chapter 4) and the strong performance isolation/guarantee (Orchestra in Chapter 5).

The first step to support the large-scale data science pipeline is to apply both CloudInsight and Orchestra to those frameworks and develop them as open-source projects. Once the techniques are applied to those frameworks, there will be more challenging research problems due to the maturity of container technologies in data science and scientific computation. Furthermore, it is easily witnessed that hundreds of Docker containers are running together on the same host (in either physical or virtual machine). Therefore, the critical challenges will be how to understand the performance implications of many contributors (containers and application processes) to a composed service/application scenario and how to design a mechanism that provides strong performance guarantee of all the containers.

7.2.3 Distributed Resource Management for Cloud IoT and Edge Computing

For the last decade, cloud computing has been a dominant infrastructure and its centralized management allows enormous resources to provide a collective power to address many applications scenarios such as analyzing large-scale (or "cloud-scale") data and handling a traffic surge to user-facing services. However, due to the large deployment of IoT, sensor and smart devices, cloud computing may not be the best option for applications with such devices. These devices generate tremendous data every second, and it is often infeasible to transfer the data to cloud data center in a real-time manner due to the bandwidth limitation, battery life, and privacy concerns. Edge computing [27, 53] helps to bridge the gap between the devices and cloud computing by placing a small or medium-size cluster with computing and storage nodes at the Internet's edge near the sensors. Edge computing is an emerging research area with many opportunities especially related to resource management including micro datacenter, middleware (for edge resource management), and serverless/cloud function. This future work will focus on making the edge computing more intelligent and autonomous, and provide better collaboration techniques between the edge and traditional cloud infrastructure.

Bibliography

[1] A. Ali-Eldin, A. Rezaie, A. Mehta, S. Razroev, S. S. de Luna, O. Seleznjev, J. Tordsson, and E. Elmroth. "How will your workload look like in 6 years? Analyzing Wikimedia's workload". In: *IEEE International Conference on Cloud Engineering (IC2E)*. Boston, MA, USA, 2014.

[2] A. Ali-Eldin, O. Seleznjev, S. S. de Luna, J. Tordsson, and E. Elmroth. "Measuring Cloud Workload Burstiness". In: *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. London, UK, 2014.

[3] Amazon Web Services. *Amazon EC2 Instance Types*. `https://aws.amazon.com/ec2/instance-types/`. [ONLINE]. 2018.

[4] Amazon Web Services. *Amazon Elastic Container Service*. `https://aws.amazon.com/ecs/`. [ONLINE].

[5] Amazon Web Services. *Amazon Web Services*. `http://aws.amazon.com`.

[6] Amazon Web Services. *AWS Auto Scaling*. `https://aws.amazon.com/autoscaling`. [ONLINE]. 2018.

[7] Amazon Web Services. *AWS EMR – Elastic MapReduce*. `https://aws.amazon.com/emr/`. [ONLINE].

[8] Amazon Web Services. *AWS Lambda – Serverless Compute - Amazon Web Services*. `https://aws.amazon.com/lambda/`. [ONLINE].

[9] Amazon Web Services. *AWS S3 - Amazon Simple Storage Service*. `https://aws.amazon.com/s3/`. [ONLINE]. 2018.

[10] Amazon Web Services. *AWS sync*. `http://docs.aws.amazon.com/cli/latest/reference/s3/sync.html`. [ONLINE]. 2018.

[11] G. Amvrosiadis, A. D. Brown, and A. Goel. "Opportunistic Storage Maintenance". In: *ACM Symposium on Operating Systems Principles (SOSP)*. Monterey, CA, USA, 2015.

[12] G. Ananthanarayanan, C. Douglas, R. Ramakrishnan, S. Rao, and I. Stoica. "True Elasticity in Multi-Tenant Data-Intensive Compute Clusters". In: *ACM Symposium on Cloud Computing (SoCC)*. San Jose, CA, USA, 2012.

[13] Apache. *Apache Hadoop*. `http://hadoop.apache.org`. [ONLINE].

[14] Apache CloudStack. *Open Source Cloud Computing*. `https://cloudstack.apache.org/`. [ONLINE]. 2018.

[15] R. Appuswam, C. Gkantsidis, D. Narayanan, O. Hodson, and A. Rowstron. "Scale-up vs Scale-out for Hadoop: Time to Rethink?" In: *ACM Symposium on Cloud Computing (SoCC)*. Santa Clara, CA, USA, 2013.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. "A View of Cloud Computing". In: *Communications of the ACM* 53.4 (2010).

[17] E.-J. van Baaren. "WikiBench: A Distributed, Wikipedia based Web Application Benchmark". In: *Master Thesis, VU University Amsterdam* (2009).

[18] A. A. Bankole and S. A. Ajila. "Cloud Client Prediction Models for Cloud Resource Provisioning in a Multitier Web Application Environment". In: *IEEE International Symposium on Service Oriented System Engineering (SOSE)*. San Francisco, CA, USA, 2013.

[19] A. Barker, B. Varghese, J. S. Ward, and I. Sommerville. "Academic Cloud Computing Research: Five Pitfalls and Five Opportunities". In: *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. 2014.

[20] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl. "Algorithms for Hyper-Parameter Optimization". In: *Neural Information and Processing Systems (NIPS)*. Granada, Spain, 2011.

[21] N. Bobroff, A. Kochut, and K. Beaty. "Dynamic Placement of Virtual Machines for Managing SLA Violations". In: *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. Munich, Germany, 2007.

[22] P. Bodik, R. Griffith, C. A. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson. "Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters". In: *USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. Santa Diego, CA, USA, 2009.

[23] box.com. *Secure File Sharing, Storage, and Collaboration — Box*. `https://www.box.com`. [ONLINE]. 2018.

[24] F. Brosig, F. Gorsler, N. Huber, and S. Kounev. “Evaluating Approaches for Performance Prediction in Virtualized Environments”. In: *IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Sstems (MASCOTS)*. 2014.

[25] M. Bux and U. Leser. “DynamicCloudSim: Simulating Heterogeneity in Computational Clouds”. In: *Future Generation Computer Systems* 46 (2015).

[26] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. “CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithm”. In: *Software: Practice and Experience* 41.1 (2011), pp. 23–50.

[27] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. D. Rose, and R. Buyya. “The Emergence of Edge Computing”. In: *IEEE Computer* 50.10 (2017), pp. 30–39.

[28] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyy. “Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications’ QoS”. In: *IEEE Transactions on Cloud Computing* 3.4 (2015).

[29] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes. “Long-term SLOs for Reclaimed Cloud Computing Resources”. In: *ACM Symposium on Cloud Computing (SoCC)*. Seattle, WA, USA, 2014.

[30] A. Chandra, W. Gong, and P. Shenoy. “Dynamic Resource Allocation for Shared Data Centers Using Online Measurements”. In: *International Workshop on Quality of Service (IWQoS)*. Berkeley, CA, USA, 2003.

[31] G. Chen, W. He, J. Liu, S. Nath, L. Rigas, L. Xiao, and F. Zhao. “Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services”. In: *USENIX Symp. on Networked Systems Design and Implement. (NSDI)*. San Francisco, CA, USA, 2008.

[32] W. Chen and E. Deelman. “WorkflowSim: A Toolkit for Simulating Scientific Workflows in Distributed Environments”. In: *IEEE International Conference on eScience (eScience)*. Chicago, IL, USA, 2012.

[33] Y. Chen, S. Alspaugh, and R. Katz. “Interactive Analytical Processing in Big Data Systems: A Cross-Industry Study of MapReduce Workloads”. In: *the VLDB Endowment* 5.12 (2012), pp. 1802–1803.

[34] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz. "The Case for Evaluating MapReduce Performance Using Workload Suites". In: *IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Singapore, 2011.

[35] L. Cheng, J. Rao, and F. C. Lau. "vScale: Automatic and Efficient Processor Scaling for SMP Virtual Machines". In: *ACM European Conference on Computer Systems (EuroSys)*. London, UK, 2016.

[36] R. C. Chiang, J. Hwang, H. H. Huang, and T. Wood. "Matrix: Achieving Predictable Virtual Machine Performance in the Clouds". In: *International Conference on Autonomic Computing (ICAC)*. Philadelphia, PA, USA, 2014.

[37] Clam AntiVirus. *ClamAV*. `https://www.clamav.net/`. [ONLINE]. 2018.

[38] Daniel Shelepov and Juan Carlos Saez Alcaide and Stacey Jeffery and Alexandra Fedorova and Nestor Perez and Zhi Feng Huang and Sergey Blagodurov and Viren Kumar. "HASS: A Scheduler for Heterogeneous Multicore Systems". In: *ACM SIGOPS Operating Systems Review* 43.2 (2009), pp. 66–75.

[39] David Koufaty and Dheeraj Reddy and Scott Hahn. "Bias Scheduling in Heterogeneous Multi-core Architectures". In: *ACM European Conference on Computer Systems (Eurosys)*. Paris, France, 2010.

[40] J. Dean and L. A. Barroso. "The Tail at Scale". In: *Communications of the ACM* 56.2 (2013), pp. 74–80.

[41] J. Dean and S. Ghemawat. "MapReduce: Simplified Data Processing on Large Clusters". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. San Francisco, CA, USA, 2004.

[42] Delft University of Technology. *The Grid Workloads Archive*. `http://gwa.ewi.tudelft.nl`. ONLINE. 2018.

[43] C. Delimitrou and C. Kozyrakis. "Paragon: QoS-aware Scheduling for Heterogeneous Datacenters". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Houston, TX, USA, 2013.

[44] C. Delimitrou and C. Kozyrakis. "Quasar: Resource-Efficient and QoS-Aware Cluster Management". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Salt Lake City, UT, USA, 2014.

[45] M. Dhingra, J. Lakshmi, S. K. Nandy, C. Bhattacharyya, and K. Gopinath. "Elastic Resources Framework in IaaS, preserving performance SLAs". In: *IEEE Int'l Conf. on Cloud Computing (CLOUD)*. Santa Clara, CA, USA, 2013.

[46] S. Di, D. Kondo, and W. Cirne. "Host Load Prediction in a Google Compute Cloud with a Bayesian Model". In: *International Conference on High Performance Computing Networking, Storage and Analysis (SC)*. Salt Lake City, UT, USA, 2012.

[47] H. X. Di Niu, B. Li, and S. Zhao. "Quality-Assured Cloud Bandwidth Auto-Scaling for Video-on-Demand Applications". In: *IEEE International Conference on Computer Communications (INFOCOM)*. Orlando, FL, USA, 2012.

[48] D. Didonaand, F. Quagliaand, P. Romanoand, and E. Torre. "Enhancing Performance Prediction Robustness by Combing Analytical Modeling and Machine Learning". In: *ACM/SPEC International Conference on Performance Engineering (ICPE)*. 2015.

[49] P. A. Dinda and D. R. O'Hallaron. "Host Load Prediction using Linear Models". In: *Cluster Computing* 3.4 (2000), pp. 265–280.

[50] B. Ding, L. Kot, A. Demers, and J. Gehrke. "Centiman: Elastic, High Performance Optimistic Concurrency Control by Watermarking". In: *ACM Symposium on Cloud Computing (SoCC)*. Kohala Coast, Hawaii, USA, 2015.

[51] Docker. *Docker – Build, Ship, and Run Any App, Anywhere*. `https://www.docker.com/`. [ONLINE].

[52] Docker. *Swarm: a Docker-native clustering system*. `https://github.com/docker/swarm`. [ONLINE].

[53] A. R. Elias, N. Golubovic, C. Krintz, and R. Wolski. "Where's The Bear?: Automating Wildlife Image Processing Using IoT and Edge Cloud Systems". In: *International Conference on Internet-of-Things Design and Implementation (IoTDI)*. Pittsburgh, PA, USA, 2017.

[54] M. A. Eriksen. "Trickle: A Userland Bandwidth Shaper for UNIX-like Systems". In: *USENIX Annual Technical Conference, FREENIX Track*. 2005.

[55] W. Fang, Z. Lu, J. Wu, and Z. Cao. "RPPS: A Novel Resource Prediction and Provisioning Scheme in Cloud Data Center". In: *IEEE International Conference on Services Computing (SCC)*. Honolulu, HI, USA, 2012.

[56] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift. "More for Your Money: Exploiting Performance Heterogeneity in Public Cloud". In: *ACM Symposium on Cloud Computing (SoCC)*. San Jose, CA, USA, 2012.

[57] A. Fedorova, M. Seltzer, and M. D. Smith. "Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler". In: *International Conference on Parallel Architecture and Compilation Techniques (PACT)*. Brasov, Romania, 2007.

[58] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. "Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications". In: (2014).

[59] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. "Clearing the Clouds – A Study of Emerging Scale-out Workloads on Modern Hardware". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Bordeaux, France, 2012.

[60] H. Fernandez, G. Pierre, and T. Kielmann. "Autoscaling Web Applications in Heterogeneous Cloud Infrastructures". In: *IEEE International Conference on Cloud Engineering (IC2E)*. Boston, MA, USA, 2014.

[61] A. Gandhi, M. Harchol-Balter, R. Raghunathan, and M. A. Kozuch. "AutoScale: Dynamic, Robust Capacity Management for Multi-Tier Data Centers". In: *ACM Transactions on Computer Systems* 30.4 (2012).

[62] A. Gandhi, P. Dube, A. Karve, A. Kochut, and H. Ellanti. "The Unobservability Problem in Clouds". In: *International Conference on Cloud and Autonomic Computing (ICCAC)*. Cambridge, MA, USA, 2015.

[63] S. K. Garg and R. Buyya. "NetworkCloudSim: Modelling Parallel Applications in Cloud Simulations". In: *IEEE International Conference on Utility and Cloud Computing (UCC)*. Melbourne, Australia, 2011.

[64] Z. Gong, X. Gu, and J. Wilkes. "PRESS: PRedictive Elastic ReSource Scaling for cloud systems". In: *International Conference on Network and Service Management (CNSM)*. Niagara Falls, Cadana, 2010.

[65] Google. *App Engine*. `https://cloud.google.com/appengine/`. [ONLINE]. 2018.

[66] Google. *Cloud Functions - Event-driven Serverless Computing*. https://cloud.google.com/functions/. [ONLINE].

[67] Google. *Compute Engine - Google Cloud Platform*. https://cloud.google.com/compute/. [ONLINE].

[68] Google. *Google Cloud Platform – Autoscaling Groups of Instances*. https://cloud.google.com/compute/docs/autoscaler. [ONLINE]. 2018.

[69] Google Cloud Platform. *Google Kubernetes Engine*. https://cloud.google.com/kubernetes-engine/. [ONLINE].

[70] B. Gregg. *The PMCs of EC2: Measuring IPC*. http://www.brendangregg.com/blog/2017-05-04/the-pmcs-of-ec2.html. [ONLINE]. 2018.

[71] J. Gupta, P. Faraboschi, F. Gioachin, L. V. Kale, R. Kaufmann, B.-S. Lee, V. March, D. Milojicic, and C. H. Suen. "Evaluating and Improving the Performance and Scheduling of HPC Applications in Cloud". In: *IEEE Transactions on Cloud Computing* 4.3 (2014), pp. 307–321.

[72] T. Hastie, R. Tibshirani, and J. Friedman. "The Element of Statistical Learning: Data Mining, Inference, and Prediction". In: (2011).

[73] T. Heinze, L. Roediger, A. Meister, Y. Ji, Z. Jerzak, and C. Fetzer. "Online Parameter Optimization for Elastic Data Stream Processing". In: *ACM Symposium on Cloud Computing (SoCC)*. Kohala Coast, Hawaii, USA, 2015.

[74] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. "Serverless Computation with OpenLambda". In: *USENIX Workshop on Hop Topics in Cloud Computing (HotCloud)*. Denvor, CO, USA, 2016.

[75] N. R. Herbst, S. Kounev, and R. Reussner. "Elasticity in Cloud Computing: What It Is, and What It Is Not". In: *International Conference on Autonomic Computing (ICAC)*. San Jose, CA, USA, 2013.

[76] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn. "Self-Adaptive Workload Classification and Forecasting for Proactive Resource Provisioning". In: *ACM/SPEC International Conference on Performance Engineering (ICPE)*. Prague, Czech Republic, 2013.

[77] H. Herodotou and S. Babu. "Profiling, What-if Analysis, and Cost-based Optimization of MapReduce Programs". In: *the VLDB Endowment* 4.11 (2011).

[78] H. Herodotou, F. Dong, and S. Babu. "No One (Cluster) Size Fits All: Automatic Cluster Sizing for Data-intensive Analytics". In: *ACM Symposium on Cloud Computing (SoCC)*. Cascais, Portugal, 2011.

[79] Heroku. *Cloud Application Platform*. `https://www.heroku.com/`. [ONLINE]. 2018.

[80] T. H. Hetherington, M. O'Connor, and T. M. Aamodt. "MemcachedGPU: Scaling-up Scale-out Key-value Stores". In: *ACM Symposium on Cloud Computing (SoCC)*. Kohala Coast, Hawaii, USA, 2015.

[81] Z. Hill and M. Humphrey. "A quantitative analysis of high performance computing with Amazon's EC2 infrastructure: The death of the local cluster?" In: *IEEE/ACM International Conference on Grid Computing (GRID)*. 2009.

[82] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. Katz, S. Shenker, and I. Stoica. "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center". In: *USENIX Symp. on Networked Systems Design and Implement. (NSDI)*. Boston, MA, USA, 2011.

[83] Y. H. Hu, S. Palreddy, and W. J. Tompkins. "A Patient-Adaptable ECG Beat Classifier Using a Mixture of Experts Approach". In: *IEEE Transactions on Biomedical Engineering* 44.9 (1997).

[84] IBM. *IBM Cloud*. `https://www.ibm.com/cloud/`. [ONLINE]. 2018.

[85] A. Iosup, N. Yigitbasi, and D. Epema. "On the Performance Variability of Production Cloud Services". In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Newport Beach, CA, USA, 2011.

[86] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema. "The Grid Workloads Archive". In: *Future Generation Computer Systems* 24.7 (2008), pp. 672–686.

[87] ISI, USC. *The Network Simulator – ns-2*. `http://www.isi.edu/nsnam/ns`. [ONLINE].

[88] S. Islam, S. Venugopal, and A. Liu. "Evaluating the Impact of Fine-scale Burstiness on Cloud Elasticity". In: *ACM Symposium on Cloud Computing (SoCC)*. Hawaii, USA, 2015.

[89] S. Islam, J. Keung, K. Lee, and A. Liu. "Empirical Prediction models for Adaptive Resource Provisioning in the Cloud". In: *Future Generation Computer Systems* 28.1 (2012).

[90] R. A. Jacobs. "Methods for Combining Experts' Probability Assessments". In: *Neural Computation* 7.5 (1995), pp. 867–888.

[91] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. "Adaptive Mixtures of Local Experts". In: *Neural Computation* 3.1 (1991), pp. 79–87.

[92] D. Jacobson, D. Yuan, and N. Joshi. *Scryer: Netflix's Predictive Auto Scaling Engine*. The Netflix Tech Blog. `http://techblog.netflix.com/2013/11/scryer-netflixs-predictive-auto-scaling.html`. Nov. 2013.

[93] S. A. Javadi and A. Gandhi. "DIAL: Reducing Tail Latencies for Cloud Applications via Dynamic Interference-aware Load Balancing". In: *International Conference on Autonomic Computing (ICAC)*. Columbus, OH, USA, 2017.

[94] H. Jayathilaka, C. Krintz, and R. Wolski. "Response Time Service Level Agreements for Cloud-hosted Web Applications". In: *ACM Symposium of Cloud Computing (SoCC)*. 2015.

[95] M. Jeon, Y. He, S. Elnikety, A. L. Cox, and S. Rixner. "Adaptive Parallelism for Web Search". In: *ACM European Conference on Computer Systems (Eurosys)*. Prague, Czech Republic, 2013.

[96] Y. Jiang, C.-S. Perng, T. Li, and R. N. Chang. "ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning". In: *IEEE International Conference on Data Mining (ICDM)*. Vancouver, BC, Canada, 2011.

[97] Y. Jiang, C.-S. Perng, T. Li, and R. N. Chang. "Cloud Analytics for Capacity Planning and Instant VM Provisioning". In: *IEEE Transactions on Network and Service Management* 10.3 (2013), pp. 312–325.

[98] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. "Encoding, Fast and Slow: Low-Latency Video Processing Using Thousands of Tiny Threads". In: *USENIX Symp. on Networked Systems Design and Implement. (NSDI)*. Boston, MA, USA, 2017.

[99] E. Jonas, Q. Pu, S. Venkataraman, I. Stoica, and B. Recht. "Occupy the Cloud: Distributed Computing for the 99%". In: *ACM Symposium on Cloud Computing (SoCC)*. Santa Clara, CA, USA, 2017.

[100] G. Jung, M. A. Hiltunen, K. R. Joshi, R. D. Schlichting, and C. Pu. "Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures". In: *International Conference on Distributed Computing Systems (ICDCS)*. Genova, Italy, 2010.

[101] M. Kambadur, T. Moseley, R. Hank, and M. A. Kim. "Measuring Interference Between Live Datacenter Applications". In: *International Conference on High Performance Computing Networking, Storage and Analysis (SC)*. Salt Lake City, UT, USA, 2012.

[102] H. Kasture and D. Sanchez. "Ubik: Efficient Cache Sharing with Strict QoS for Latency-Critical Workloads". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Salt Lake City, UT, USA, 2014.

[103] A. Khan, X. Yan, S. Tao, and N. Anerousis. "Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach". In: *IEEE Network Operations and Management Symposium (NOMS)*. Maui, Hi, USA, 2012.

[104] I. K. Kim, W. Wang, and M. Humphrey. "PICS: A Public IaaS Cloud Simulator". In: *IEEE International Conference on Cloud Computing (CLOUD)*. New York, NY, USA, 2015.

[105] I. K. Kim, S. Zeng, C. Young, J. Hwang, and M. Humphrey. "A Supervised Learning Model for Identifying Inactive VMs in Private Cloud Data Centers". In: *ACM/IFIP/USENIX International Middleware Conference (Middleware)*. Trento, Italy, 2016.

[106] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey. "CloudInsight: Utilizing a Council of Experts to Predict Future Cloud Application Workloads". In: *IEEE International Conference on Cloud Computing (CLOUD)*. San Francisco, CA, USA, 2018.

[107] I. K. Kim, J. Steele, Y. Qi, and M. Humphrey. "Comprehensive Elastic Resource Management to Ensure Predictable Performance for Scientific Applications on Public IaaS Clouds". In: *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. London, UK, 2014.

[108] I. K. Kim, W. Wang, Y. Qi, and M. Humphrey. "Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling". In: *IEEE*

International Conference on Cloud Computing (CLOUD). San Francisco, CA, USA, 2016.

[109] I. K. Kim, S. Zeng, C. Young, J. Hwang, and M. Humphrey. "iCSI: A Cloud Garbage VM Collector for Addressing Inactive VMs with Machine Learning". In: *IEEE International Middleware Conference on Cloud Engineering*. Vancouver Canada, 2017.

[110] I. K. Kim, J. Hwang, W. Wang, and M. Humphrey. "Orchestra: Guaranteeing Performance SLAs for Cloud Applications by Avoiding Resource Storms". In: *IEEE International Symposium on Parallel and Distributed Computing (ISPDC)*. Geneva, Switzerland, 2018.

[111] D. Kliazovich, P. Bouvry, and S. U. Khan. "GreenCloud: a Packet-level Simulator of Energy-Aware Cloud Computing Data Centers". In: *Journal of Supercomputing* 62.3 (2013), pp. 1263–1283.

[112] S. Koyano, S. Ata, I. Oka, and K. Inoue. "A High-grained Traffic Prediction for Microseconds Power Control in Energy-aware Routers". In: *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*. Chicago, IL, USA, 2012.

[113] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. E. Culler, and R. H. Katz. "NapSAC: Design and Implementation of a Power-Proportional Web Cluster". In: *Computer Communication Review* 41.1 (2011).

[114] S. S. Krishnan and R. K. Sitaraman. "Video Stream Quality Impacts Viewer Behavior: Inferring Causality Using Quasi-Experimental Designs". In: *ACM SIGCOMM Internet Measurement Conference (IMC)*. Boston, MA, USA, 2012.

[115] J. Krzywda, A. Ali-Eldin, T. E. Carlson, P.-O. Ostberg, and E. Elmroth. "Power-performance tradeoffs in data center servers: DVFS, CPU pinning, horizontal, and vertical scaling". In: *Future Generation Computer Systems* 81 (2018), pp. 114–128.

[116] Kubernetes. *Kubernetes – Production-Grade Container Orchestration*. `https://kubernetes.io/`. [ONLINE]. 2018.

[117] S. Kulkarni, N. Bhagat, M. Fu, V. Kedigehalli, C. Kellogg, S. Mittal, J. M. Patel, K. Ramasamy, and S. Taneja. "Twitter Heron: Stream Processing at Scale". In: *ACM SIGMOD International Conference on Management of Data (SIGMOD)* (2014).

[118] S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. "Application Performance Modeling in a Virtualized Environments". In: *International Symposium on High-Performance Computer Architecture (HPCA)*. 2010.

[119] S. Kundu, R. Rangaswami, A. Gulati, M. Zhao, and K. Dutta. "Modeling Virtualized Applications using Machine Learning Techniques". In: *International Conference on Virtual Execution Environments (VEE)*. 2012.

[120] P. Leitner and J. Cito. "Patterns in the Chaos–A Study of Performance Variation and Predictability in Public IaaS Clouds". In: *ACM Transactions on Internet Technology* 16.15 (2016).

[121] J. Leverich and C. Kozyrakis. "Reconciling High Server Utilization and Sub-millisecond Quality-of-Service". In: *ACM European Conference on Computer Systems (Eurosys)*. Amsterdam, Netherlands, 2014.

[122] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble. "Tales of the Tail: Hardware, OS, and Application-level Sources of Tail Latency". In: *ACM Symposium on Cloud Computing (SoCC)*. Seattle, WA, USA, 2014.

[123] S.-P. Li and M.-H. Wong. "Data Allocation in Scalable Distributed Database Systems Based on Time Series Forecasting". In: *IEEE International Congress on Big Data (BigData Congress)*. Santa Clara, CA, USA, 2013.

[124] G. Linden. *Make Data Useful*. `http://www.gduchamp.com/media/StanfordDataMining.2006-11-28.pdf`. [ONLINE].

[125] C. Liu, C. Liu, Y. Shang, S. Chen, B. Cheng, and J. Chen. "An Adaptive Prediction Approach based on Workload Pattern Discrimination in the Cloud". In: *Journal of Network and Computer Applications* 80 (2017).

[126] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. "Heracles: Improving Resource Efficiency at Scale". In: *Int'l Symp. on Computer Architecture (ISCA)*. Portland, OR, 2015.

[127] J. Loff and J. Garcia. "Vadara: Predictive Elasticity for Cloud Applications". In: *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Singapore, 2014.

[128] A. K. Maji, S. Mitra, and S. Bagchi. "ICE: An Integrated Configuration Engine for Interference Mitigation in Cloud Services". In: *International Conference on Autonomic Computing (ICAC)*. Grenoble, France, 2015.

[129] A. K. Maji, S. Mitra, B. Zhou, S. Bagchi, and A. Verma. "Mitigating interference in cloud services by middleware reconfiguration". In: *International Middleware Conference (Middleware)*. Bordeaux, France, 2014.

[130] M. Mao and M. Humphrey. "A Performance Study on the VM Startup Time in the Cloud". In: *IEEE international Conference on Cloud Computing (CLOUD)*. Honolulu, HI, USA, 2012.

[131] M. Mao and M. Humphrey. "Auto-Scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Seattle, WA, USA, 2011.

[132] M. Mao and M. Humphrey. "Scaling and Scheduling to Maximize Application Performance within Budget Constraints in Cloud Workflows". In: *IEEE International Symposium on Parallel and Distributed Processing (IPDPS)*. Cambridge, MA, USA, 2013.

[133] C. C. T. Mark, D. Niyato, and T. Chen-Khong. "Evolutionary Optimal Virtual Machine Placement and Demand Forecaster for Cloud Computing". In: *IEEE International Conference on Advanced Information Networking and Applications (AINA)*. Biopolis, Singapore, 2011.

[134] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible colocations". In: *International Symposium on Microarchitecture (Micro)*. Porto Alegre, Brazil, 2011.

[135] P. Mell and T. Grance. *The NIST Definition of Cloud Computing*. `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf`. [ONLINE]. 2011.

[136] P. Menage. *CGROUPS*. `https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt`. [ONLINE]. 2018.

[137] Mesosphere. *Marathon: A container orchestration platform for Mesos and DC/OS*. `https://github.com/docker/swarm`. [ONLINE].

[138] H. Mi, H. Wang, G. Yin, Y. Zhou, D. Shi, and L. Yuan. "Online Self-reconfiguration with Performance Guarantee for Energy-efficient Large-scale Cloud Comp. Data Centers". In: *IEEE International Conference on Services Computing (SCC)*. Miami, Florida, 2010.

[139] R. Mian, P. Martin, F. Zulkernine, and J. L. Vazquez-Poletti. "Toward Building Performance Models for Data-Intensive Workloads in Public Clouds". In: *ACM/SPEC International Conf. on Performance Engineering (ICPE)*. 2013.

[140] Microsoft. *Azure Autoscale*. `https://azure.microsoft.com/en-us/features/autoscale`. [ONLINE]. 2018.

[141] Microsoft. *Azure Container Service (AKS)*. `https://azure.microsoft.com/en-us/services/container-service/`. [ONLINE].

[142] Microsoft. *Azure FunctionsServerless Architecture*. `https://azure.microsoft.com/en-us/services/functions/`. [ONLINE].

[143] Microsoft. *Azure Storage*. `http://azure.microsoft.com/en-us/services/storage`. [ONLINE].

[144] Microsoft. *Microsoft Azure*. `http://azure.microsoft.com`. [ONLINE].

[145] MongoDB. *MongoDB*. `https://www.mongodb.com/`. [ONLINE]. 2018.

[146] Moor Insights and Strategy. *TCO Analysis Demonstrates How Moving To The Cloud Can Save Your Company Money*. `https://www.forbes.com/sites/moorinsights/2016/04/11/tco-analysis-demonstrates-how-moving-to-the-cloud-can-save-your-company-money/\#1c30177d7c4e`. [ONLINE]. 2016.

[147] S. Muppala, X. Zhou, and L. Zhang. "Regression Based Multi-tier Resource Provisioning for Session Slowdown Guarantees". In: *IEEE International Performance Computing and Communications Conference (IPCCC)*. Albuquerque, NM, USA, 2010.

[148] R. Nathuji, A. Kansal, and A. Ghaffarkhah. "QClouds: Managing Performance Interference Effects for QoS-Aware Clouds". In: *ACM European Conference on Computer Systems (Eurosys)*. Paris, France, 2010.

[149] Netflix. *Fenzo: Extensible Scheduler for Mesos Frameworks*. `https://github.com/Netflix/Fenzo`. [ONLINE].

[150] M. A. S. Netto, C. Cardonha, R. L. F. Cunha, and M. D. Assuncao. "Evaluating Auto-scaling Strategies for Cloud Computing Environments". In: *IEEE International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*. Paris, France, 2014.

[151] NGINX. *Nginx Load Balancing – TCP and UDP Load Balancer*. https://www.nginx.com/resources/admin-guide/tcp-load-balancing/. [ONLINE]. 2018.

[152] NGINX. *Nginx Reverse Proxy*. https://www.nginx.com/resources/admin-guide/reverse-proxy/. [ONLINE]. 2018.

[153] A. Y. Nikravesh, S. A. Ajila, and C.-H. Lung. "Towards an Autonomic Auto-Scaling Prediction System for Cloud Resource Provisioning". In: *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. Firenze, Italy, 2015.

[154] S. Niu, J. Zhai, X. Ma, X. Tang, and W. Chen. "Cost-effective Cloud HPC Resource Provisioning by Building Semi-Elastic Virtual Clusters". In: *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. Denver, CO, USA, 2013.

[155] Q. Noorshamsand, A. Buschand, A. Rentschlerand, D. Bruhnand, S. Kounevand, P. Tumaand, and R. Reussner. "Automated Modeling of I/O Performance and Interference Effects in Virtualized Storage System". In: *International Workshop on Big Data and Cloud Perf. (DCPerf)*. 2014.

[156] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini. "DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments". In: *USENIX Annual Technical Conference (ATC)*. San Jose, CA, USA, 2013.

[157] A. Nunez, J. L. Vazquez-Poletti, A. C. Caminero, G. G. Castane, J. Carretero, and I. M. Llorente. "iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator". In: *Journal of Grid Computing* 10.1 (2012), pp. 185–209.

[158] Openstack. *OpenStack is open source software for creating private and public clouds*. https://www.openstack.org/. [ONLINE]. 2018.

[159] OpenSUSE. *Tuning the Task Scheduler*. https://doc.opensuse.org/documentation/leap/tuning/html/book.sle.tuning/cha.tuning.taskscheduler.html. [ONLINE]. 2017.

[160] Z. Ou, H. Zhuang, J. K. Nurminen, and A. Yla-Jaaski. "Exploiting Hardware Heterogeneity within the Same Instance Type of Amazon EC2". In: *USENIX Workshop on Hop Topics in Cloud Computing (HotCloud)*. Boston, MA, USA, 2012.

[161] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. "Automated Control of Multiple Virtualized Resources". In: *ACM European conference on Computer systems (Eurosys)*. Nuremberg, Germany, 2009.

[162] N. Park, I. Ahmad, and D. J. Lilja. "Romano: Autonomous Storage Management using Performance Prediction in Multi-Tenant Datacenters". In: *ACM Symposium on Cloud Computing (SoCC)*. San Jose, CA, USA, 2012.

[163] A. Pavlo. *Python TPC-C*. `https://github.com/apavlo/py-tpcc`. [ONLINE]. 2018.

[164] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu. "HadoopWatch: A First Step Towards Comprehensive Traffic Forecasting in Cloud Computing". In: *IEEE Conference on Computer Communications (INFOCOM)*. Toronto, ON, Canada, 2014.

[165] E. Pettijohn, Y. Guo, P. Lama, and X. Zhou. "User-Centric Heterogeneity-Aware MapReduce Job Provisioning in the Public Cloud". In: *International Conference on Autonomic Computing (ICAC)*. 2014.

[166] A. Pucher, E. Gul, R. Wolski, and C. Krintz. "Using Trustworthy Simulation to Engineer Cloud Schedulers". In: *IEEE International Conference on Cloud Engineering (IC2E)*. 2015.

[167] Rackspace. *Rackspace: Managed Dedicated and Cloud Computing Services*. `http://www.rackspace.com/`. [ONLINE].

[168] N. Rameshan, L. Navarro, E. Monte, and V. Vlassov. "Stay-Away, protecting sensitive applications from performance interference". In: *ACM/IFIP/USENIX International Middleware Conference (Middleware)*. Bordeaux, France, 2014.

[169] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch. "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis". In: *ACM Symposium on Cloud Computing (SoCC)*. San Jose, CA, USA, 2013.

[170] Z. Ren, X. Xu, J. Wan, W. Shi, and M. Zhou. "Workload Characterization on a Production Hadoop Cluster: A Case Study on Taobao". In: *IEEE International Symposium on Workload Characterization (IISWC)*. Portland, OR, USA, 2013.

[171] RightScale. *2017 State of the Cloud Report*. `https://www.rightscale.com/lp/state-of-the-cloud`. [ONLINE]. 2017.

[172] RightScale. *RightScale*. http://www.rightscale.com. [ONLINE].

[173] R. da Rosa Righi, V. F. Rodrigues, C. A. da Costa, G. Galante, L. C. E. de Bona, and T. Ferreto. "AutoElastic: Automatic resource elasticity for high performance applications in the cloud". In: *IEEE Transactions on Cloud Computing* 4.1 (2016).

[174] N. Roy, A. Dubey, and A. Gokhale. "Efficient Autoscaling in the Cloud using Predictive Models for Workload Forecasting". In: *IEEE International Conference on Cloud Computing (CLOUD)*. Washington DC, USA, 2011.

[175] Salesforce.com. *Salesforce.com: The Customer Success Platform To Grow Your Business*. https://www.salesforce.com. [ONLINE]. 2018.

[176] P. Saripalli, G. Kiran, R. S. R, H. Narware, and N. Bindal. "Load Prediction and Hot Spot Detection Models for Autonomic Cloud Computing". In: *IEEE International Conference on Utility and Cloud Computing (UCC)*. Melbourne, Australia, 2011.

[177] SCALR. *SCALR – The Hybrid Cloud Management Platform*. http://www.scalr.com. [ONLINE].

[178] J. Schad, J. Dittrich, and J.-A. Quiane-Ruiz. "Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance". In: *the VLDB Endowment* 3.1–2 (2010), pp. 460–471.

[179] M. Schwarzkopf, D. G. Murray, and S. Hand. "The Seven Deadly Sins of Cloud Computing Research". In: *USENIX Workshop on Hop Topics in Cloud Computing (HotCloud)*. Boston, MA, USA, 2012.

[180] U. Sharma, P. Shenoy, and S. Sahu. "A Flexible Elastic Control Plane for Private Clouds". In: *ACM Cloud and Autonomic Computing Conference (CAC)*. Miami, FL, USA, 2013.

[181] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. "CloudScale: Elastic Resource Scaling for Multi-Tenant Cloud Systems". In: *ACM Symposium on Cloud Computing (SoCC)*. Cascais, Portugal, 2011.

[182] M. Smit, BradleySimmons, and M. Litoiu. "Distributed, application-level monitoring for heterogeneous clouds using stream processing". In: *Future Generation Computer Systems* 29.8 (2013).

[183] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya. "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers". In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Shenzhen, China, 2015.

[184] T. Spangler. *Netflix Bandwidth Usage Climbs to Nearly 37% of Internet Traffic at Peak Hours*. `http://variety.com/2015/digital/news/netflix-bandwidth-usage-internet-traffic-1201507187/`. [ONLINE]. 2015.

[185] S. Spinner, N. Herbst, S. Kounev, X. Zhu, L. Lu, M. Uysal, and R. Griffith. "Proactive Memory Scaling of Virtualized Applications". In: *IEEE International Conference on Cloud Computing (CLOUD)*. New York, NY, USA, 2015.

[186] L. Suresh, M. Canini, S. Schmid, and A. Feldmann. "C3: Cutting Tail Latency in Cloud Data Stores via Adaptive Replica Selection". In: *USENIX Symp. on Networked Systems Design and Implement. (NSDI)*. Oakland, CA, USA, 2015.

[187] SWIMProjectUCB. *Workloads repository*. `https://github.com/SWIMProjectUCB/SWIM/wiki/Workloads-repository`. ONLINE. 2018.

[188] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa. "ReQoS: Reactive Static/Dynamic Compilation for QoS in Warehouse Scale Computer". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Houston, TX, USA, 2013.

[189] S. K. Tesfatsion, E. Wadbro, and J. Tordsson. "Autonomic Resource Management for Optimized Power and Performance in Multi-tenant Clouds". In: *International Conference on Autonomic Computing (ICAC)*. Wurzburg, Germany, 2016.

[190] J. M. Tirado, D. Higuero, F. Isaila, and J. Carretero. "Predictive Data Grouping and Placement for Cloud-based Elastic Server Infrastructures". In: *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. Newport Beach, CA, USA, 2011.

[191] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D. Ryaboy. "Storm @ Twitter". In: *ACM SIGMOD International Conference on Management of Data (SIGMOD)* (2014).

[192] N. Vasic, D. Novakovic, S. Miucin, D. Kostic, and R. Bianchini. "DejaVu: Accelerating Resource Allocation in Virtualized Environments". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Bordeaux, France, 2012.

[193] Veenhof. *Workload Patterns for Cloud Computing*. http://watdenkt.veenhof.nu/2010/07/13/workload-patterns-for-cloud-computing/. [ONLINE].

[194] A. Verma, L. Cherkasova, and R. H. Campbell. "ARIA: Automatic Resource Inference and Allocation for MapReduce Environments". In: *International Conference on Autonomic Computing (ICAC)*. Karlsruhe, Germany, 2011.

[195] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. "Large-scale Cluster Management at Google with Borg". In: *ACM European conference on Computer systems (Eurosys)*. Bordeaux, France, 2015.

[196] B. J. Watson, M. Marwah, D. Gmach, Y. Chen, M. Arlitt, and Z. Wang. "Probabilistic Performance Modeling of Virtualized Resource Allocation". In: *International Conference on Autonomic Computing (ICAC)*. 2010.

[197] S. Watts. *BMC Blog – SaaS vs PaaS vs IaaS: Whats The Difference and How To Choose*. http://www.bmc.com/blogs/saas-vs-paas-vs-iaas-whats-the-difference-and-how-to-choose/. [ONLINE]. 2018.

[198] B. Wickremasinghe, R. N. Calheiros, and R. Buyya. "CloudAnalyst: A CloudSim-based Visual Modeller for Analysing Cloud Computing Environments and Applications". In: *IEEE International Conference on Advanced Information Networking and Applications (AINA)*. Perth, Western Australia, 2010.

[199] WikiBench. *Wikipedia Access Traces*. http://www.wikibench.eu. 2018.

[200] J. Wilkes. *More Google cluster data*. Google research blog. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html. Nov. 2011.

[201] R. Wolski and J. Brevik. *Eucalyptus IaaS Cloud Workload*. https://www.cs.ucsb.edu/~rich/workload/. ONLINE. 2018.

[202] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif. "Black-box and Gray-box Strategies for Virtual Machine Migration". In: *USENIX Symp. on Networked Systems Design and Implement. (NSDI)*. Cambridge, MA, USA, 2007.

[203] T. Wood, L. Cherkasova, K. M. Ozonat, and P. J. Shenoy. "Profiling and Modeling Resource Usage of Virtualized Applications". In: *The 9th ACM/IFIP/USENIX Middleware Conference (Middleware)*. Leuven, Belgium, 2008.

[204] Q. Wu. *Making Facebook's software infrastructure more energy efficient with Autoscale*. https://code.facebook.com/posts/816473015039157/making-facebook-s-software-infrastructure-more-energy-efficient-with-autoscale/. [ONLINE].

[205] S. Wu, B. Li, X. Wang, and H. Jin. "HybridScaler: Handling Bursting Workload for Multi-tier Web Applications in Cloud". In: *International Symposium on Parallel and Distributed Computing (ISPDC)*. FuZhou, China, 2016.

[206] Z. Xiao, W. Song, and Q. Chen. "Dynamic Resource Allocation Using Virtual Machines for Cloud Computing Environment". In: *IEEE Transactions on Parallel and Distributed Systems* 24.6 (2013).

[207] N. J. Yadwadkar, G. Ananthanarayanan, and R. Katz. "Wrangler: Predictable and Faster Jobs using Fewer Resources". In: *ACM Symposium on Cloud Computing (SoCC)*. Seattle, WA, USA, 2014.

[208] H. Yang, A. Breslow, J. Mars, and L. Tang. "Bubble-Flux: Precise Online QoS Management for Increased Utilization in Warehouse Scale Computers". In: *Int'l Symp. on Computer Architecture (ISCA)*. Tel-Aviv, Israel, 2013.

[209] J. Yang, C. Liu, Y. Shang, B. Cheng, Z. Mao, C. Liu, L. Niu, and J. Chen. "A Cost-aware Auto-scaling Approach Using the Workload Prediction in Service Clouds". In: *Information Systems Frontiers* 16.1 (2014).

[210] J. Yang, C. Liu, Y. Shang, Z. Mao, and J. Chen. "Workload Predicting-Based Automatic Scaling in Service Clouds". In: *IEEE International Conference on Cloud Computing (CLOUD)*. Santa Clara, CA, USA, 2013.

[211] L. Yazdanov and C. Fetzer. "Lightweight Automatic Resource Scaling for Multi-tier Web Applications". In: *IEEE International Conference on Cloud Computing (CLOUD)*. Alaska, USA, 2014.

[212] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. "Spark: Cluster Computing with Working Sets". In: *USENIX Workshop on HotTopics in Cloud Computing (HotCloud)*. 2010.

[213] Q. Zhang, M. F. Zhani, S. Zhang, Q. Zhu, R. Boutaba, and J. L. Hellerstein. "Dynamic Energy-Aware Capacity Provisioning for Cloud Computing Environments". In: *International Conference on Autonomic Computing (ICAC)*. San Jose, CA, USA, 2012.

[214] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. "CPI^2: CPU Performance Isolation for Shared Compute Clusters". In: *ACM European Conference on Computer Systems (Eurosys)*. Prague, Czech Republic, 2013.

[215] Z. Zhang, L. Cherkasova, and B. T. Loo. "Benchmarking Approach for Designing a MapReduce Performance Model". In: *ACM/SPEC International Conference on Performance Engineering (ICPE)*. 2013.

[216] H. Zhu and M. Erez. "Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems". In: *Int'l Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Atlanta, GA, USA, 2016.

[217] E. Zohar, I. Cidon, and O. Mokryn. "The Power of Prediction: Cloud Bandwidth and Cost Reduction". In: *ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. Toronto, ON, Canada, 2011.