Capacity Scaling for Elastic Compute Clouds

Ahmed Aleyeldin (Ali-Eldin) Hassan

LICENTIATE THESIS, JUNE 2013
DEPARTMENT OF COMPUTING SCIENCE
UMEÅ UNIVERSITY
SWEDEN

Department of Computing Science
Umeå University
SE-901 87 Umeå, Sweden

ahmeda@cs.umu.se

ISBN 978-91-7459-688-5
ISSN 0348-0542
UMINF 13.14

Printed by Print & Media, Umeå University, 2013

Abstract

Cloud computing is a computing model that allows better management, higher utilization and reduced operating costs for datacenters while providing on demand resource provisioning for different customers. Data centers are often enormous in size and complexity. In order to fully realize the cloud computing model, efficient cloud management software systems that can deal with the datacenter size and complexity need to be designed and built.

This thesis studies automated cloud elasticity management, one of the main and crucial datacenter management capabilities. Elasticity can be defined as the ability of cloud infrastructures to rapidly change the amount of resources allocated to an application in the cloud according to its demand. This work introduces algorithms, techniques and tools that a cloud provider can use to automate dynamic resource provisioning allowing the provider to better manage the datacenter resources. We design two automated elasticity algorithms for cloud infrastructures that predict the future load for an application running on the cloud. It is assumed that a request is either serviced or dropped after one time unit, that all requests are homogeneous and that it takes one time unit to add or remove resources. We discuss the different design approaches for elasticity controllers and evaluate our algorithms using real workload traces. We compare the performance of our algorithms with a state-of-the-art controller. We extend on the design of the best performing controller out of our two controllers and drop the assumptions made during the first design. The controller is evaluated with a set of different real workloads.

All controllers are designed using certain assumptions on the underlying system model and operating conditions. This limits a controller's performance if the model or operating conditions change. With this as a starting point, we design a workload analysis and classification tool that assigns a workload to its most suitable elasticity controller out of a set of implemented controllers. The tool has two main components, an analyzer and a classifier. The analyzer analyzes a workload and feeds the analysis results to the classifier. The classifier assigns a workload to the most suitable elasticity controller based on the workload characteristics and a set of predefined business level objectives. The tool is evaluated with a set of collected real workloads and a set of generated synthetic workloads. Our evaluation results shows that the tool can help a cloud provider to improve the QoS provided to the customers.

Preface

This thesis consists of a brief introduction to the field, a short discussion of the main problems studied, and the following papers.

Paper I Ahmed Ali-Eldin, Johan Tordsson and Erik Elmroth.
An adaptive hybrid elasticity controller for cloud infrastructures. In *Proceedings of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS 2012)*, pages 204-212. IEEE, 2012.

Paper II Ahmed Ali-Eldin, Maria Kihl, Johan Tordsson and Erik Elmroth.
Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *Proceedings of the 3rd workshop on Scientific Cloud Computing (ScienceCloud 2012)*, pages 31–40. ACM, 2012.

Paper III Ahmed Ali-Eldin, Johan Tordsson, Erik Elmroth and Maria Kihl.
Workload Classification for Efficient Cloud Infrastructure Elasticity Control. *Technical Report, UMINF 13.13*, Department of Computing Science, Umeå University, Sweden, 2013.

Financial support has been provided in part by the European Community's Seventh Framework Programme OPTIMIS project under grant agreement #257115, the Swedish Government's strategic effort eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

Acknowledgements

This work would have been impossible if it was not for a number of people to whom I am greatly indebted. First of all, I would like to thank my advisor Erik Elmroth for his endurance, patience, inspiration and great discussions. Erik has created a very unique positive research environment that is rare to find any where else. I would also like to thank my coadvisor Johan Tordsson for the hard work, the great ideas, the long discussions, and the great feedback. The past a few years have been very unique, I got married, a revolution happened back home and I got my first kid. Erik and Johan have been considerate, helpful and supportive. They are not just advisors, they are mentors, teachers, and above all friends. It has been a privilege working with you. Your positive impact will stay with me for the rest of my life.

I would also like to thank Daniel Espling and Wubin Li for their great help with OPTIMIS, Per-Olov Östberg, Peter Gardfjäll and Lars Larsson for the inspiring discussions and the great feedback, Ewnetu Bayuh my office mate for the great time we spend together, Christina Igasto for helping me settle, Petter Svärd, Francisco Hernández, Mina Sedaghat, Selome Kosten, Gonzalo Rodrigo, Cristian Klein, Luis Tomas, Amardeep Mehta, Lei Xu, Tomas Forsman, Emad Hassan for the great time we spend together.

I would like to thank Maria Kihl at Lund university for the interesting collaboration, positive feedback and inspiring discussions. Four years ago, when I started my postgraduate studies, I met Sameh El-Ansary who hired me as a research assistant. He taught me a lot about research. He was a great mentor and now he is a dear friend !

On a more personal level, I would like to thank my parents for their love and their support. This work would have not been possible if it was not for them explaining to me maths and physics 24 years ago! I was 4 when they started teaching me the multiplication table. By the time I was five I knew a little bit more than my peers! I love you both and I pray that I will always be a source of happiness to you!

I fell in love with a girl one week before I started my PhD studies. We got married 3 months after I started my PhD. Hebatullah, thank you for being there for me always with love, support and care. I would also like to thank the rest of my family for their love and support. Last but not least, I would like to thank Salma my little daughter. She is the most precious thing I have ever had and the main source of joy in life!

Thank you all !

Contents

Chapter 1

Introduction

The idea of having computing power organized as a utility dates back to the 1960s. In a speech in 1961, John McCarthy predicted that "computation may someday be organized as a public utility" just like electricity and water [20]. His idea did not gain popularity until the late 1990s when research on Grid computing started. The term Grid computing was used to describe technologies that enable on demand usage of computing power [20]. Grid computing has been used mainly for scientific applications within the scientific community and did not gain widespread support outside that community.

Driven originally by economic needs, cloud computing can be considered as an evolution of Grid computing that gained popularity during the last a few years. The cloud computing model aims for the efficient use of datacenter resources by increasing resource utilization and reducing the operating costs while providing on demand computing resources. In contrast to Grid computing, cloud computing has mainly been used for commercial systems with some interest from the scientific community [58]. Most grid systems are used mainly for batch jobs which are very common for scientific applications. Clouds on the other hand support the deployment of more application types including webservers, data-processing applications and batch systems.

There is no agreement on how to define cloud computing [9, 13, 20]. The definition used in this thesis is aligned with the NIST definition [31] which describes cloud computing as a resource utilization model that enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. The computing resources can be rapidly provisioned and released with minimal management effort or service provider interaction.

There are many technologies that contributed directly to the possibility of building cloud systems. Advances in virtualization technologies [1, 10], advances in server power management techniques [17] and increased network bandwidth are the main enabling technologies for cloud computing.

Customers lease resources from a cloud service provider to run their applications, services, computations or store their data on the leased resources.

Often, the resources are used to deploy a web-based service accessible by other entities, e.g, Reddit runs a social news and entertainment service on Amazon [37]. For the rest of this work, we interchangeably use the terms 'service' and 'application' to describe the customer's use of cloud resources.

1.1 Cloud computing characteristics

There are five essential characteristics of cloud computing identified by NIST [31]. These five characteristics can be summarized as follow:

1. On demand provisioning of resources requiring no human interaction with the service provider.

2. Broad network access able to handle users of different network clients such as, mobile phones and workstations.

3. Resource pooling between multiple customers having different resource requirements. The cloud is abstract to the costumers generally. Customers have no control or knowledge over the exact location of their resources except at a higher level of abstraction.

4. Rapid elasticity which is the ability to vary allocated capacity depending on the load, sometimes automatically. The resources available for provisioning often appear unlimited for the service user.

5. Transparent resource usage monitoring where running applications are actively monitored. Usage and performance reports should be reported transparently.

We add to the previous list two more characteristics that we believe are essential for the cloud service model.

1. Fault tolerance is important for cloud platforms since cloud platforms are built using commodity hardware [11]. The cloud provider should provide transparent fault tolerance mechanisms that mask failures from the customers.

2. The cloud provider has to provide Quality-of-Service (QoS) guarantees for the customers. The QoS guarantees should be at least similar to the guarantees

Although these two characteristics are not unique to clouds, they are essential for the realization of the cloud computing model. Full realization of the cloud model is still far from reality [36, 18, 54]. Limitations in the current cloud offerings require more research in order to fill the gaps between the cloud model and the reality. This thesis contributes to better management of cloud systems, thus it contributes to filling this gap.

1.2 Cloud Computing models

Cloud systems can be categorized based on their service models or deployment models. Service models describe the type of service offered by the cloud provider. Deployment models describe the way a service running on the cloud is deployed on the actual infrastructure.

1.2.1 Cloud Computing service models

Cloud computing systems are often classified into three main service models, namely:

1. The Infrastructure-as-a-Service (IaaS) model, where the service provider leases raw computing resources. These resources can be either physical (sometimes referred to as Hardware-as-a-Service) or virtual [39]. This model enables the cloud service user to specify the needs of an application in terms of raw computing resources. The cloud user is responsible for deploying and configuring the operating systems, the applications and any required software packages to run on the infrastructure. The user does not control the cloud infrastructure, e.g., the user has no control over where the assigned resources are in a datacenter. Amazon EC2 [39] and RackSpace [45] are two examples of IaaS platforms.

2. The Platform-as-a-Service (PaaS) model, where the service provider offers a platform that supports certain programming languages, libraries, services, and tools. The cloud service user can use the platform to develop and/or deploy applications using the provided tools and/or environments. The user does not manage the operating system, the underlying hardware or the software packages supported by the PaaS provider. Google's App Engine [42] and Windows Azure [43] are two examples of PaaS platforms.

3. The Software-as-a-Service (SaaS) model, where the service provider offers an application running on a cloud infrastructure to the customers. The application is used by the cloud service user as is. Oracle Cloud [44] and Salesforce [47] are two examples of SaaS providers.

These models are not mutually exclusive. They can coexist in the same datacenter. Some IaaS providers host Paas and SaaS Clouds in an IaaS cloud [38].

1.2.2 Cloud Computing deployment models

Cloud deployment models describe how and where services running in a cloud are deployed. It also describes who can access the resources of a cloud. The deployment models identified by NIST are:

1. Private clouds: Owned, managed and used by a single organization. Access to the cloud is restricted to entities within the organization. This

model provides higher security for the organization since all sensitive data is kept internal. The National Security Agency in the USA recently revealed that they are operating a private cloud [12].

2. Community clouds: shared between multiple organizations having common interests. Access to the cloud is restricted to entities within the organizations. The G-Cloud project in the UK started as a community cloud [41].

3. Public clouds are infrastructures that lease computing resources to the public. They are typically operated and managed by business or academic organizations. The cloud resources are shared between the customers. Amazon's EC2, RackSpace and Salesforce are all examples of public clouds.

4. Hybrid clouds describes distinct cloud systems (public, private or community) that have mutual agreements and technologies enabling data and application portability. The model allows one cloud entity to extend its capacity by using resources from another cloud entity. In addition, the model allows load balancing between the partner clouds. The recently announced hybrid cloud between Netapp, Equinix and Amazon [40] is an example of this deployment model.

1.3 Cloud Middlewares

Cloud middlewares are software systems used to manage cloud computing systems. These middlewares should be designed to provide the essential characteristics of the cloud computing model. Cloud middlewares should provide APIs and abstraction layers through which the customer can submit requests for resources and monitor resource usage. The middlewares need to manage admission control, resource pooling, fault tolerance, on demand provisioning, resource placement, and possibly automatic elasticity. In addition, there should be enough hardware resources to enable efficient resource pooling and rapid elasticity. The middleware is also responsible for enforcing all QoS guarantees. The service and deployment models supported by the cloud also affects the middleware design.

Current cloud datacenters are huge in size. For example, Rackspace has more than 94000 servers hosted in 9 datacenters serving more than 200000 customers. Typically, a server has between 4 cores and 128 cores. Management of such huge and complex systems requires some automation in the management software. On demand provisioning and resource pooling requires the middleware to be able to handle the provisioning demands for the customer and allocate the resources required by the service autonomically and transparently [24]. Fault tolerance should be transparent to the user and the applications with minimal effect on the QoS of the service. Resource usage monitoring

should be done frequently and transparently. The main focus of this thesis is the design of algorithms that enable the automation of cloud middlewares with an emphasis on algorithms for automated rapid elasticity.

Chapter 2

Rapid Elasticity: Cloud Capacity Auto-Scaling

NIST describes *rapid elasticity* as a cloud essential characteristic that enables capabilities to be elastically provisioned and released, to scale rapidly according to applications' demand. To the cloud user, the resources available often appear unlimited. The NIST definition does not require elasticity to be automated although it can be automated in some cases [31]. Elasticity control can be divided in to two classes, namely, horizontal elasticity and vertical elasticity. Horizontal elasticity is the ability of the cloud to rapidly vary the number of VMs allocated to a service according to demand [4]. Vertical elasticity is the ability of the cloud to rapidly change configurations of virtual resources allocated to a service to vary with demand, e.g., adding more CPU power, memory or disk space to already running VMs [57]. This thesis focuses on automated horizontal elasticity or resource auto-scaling and the challenges associated with the automation.

2.1 Elasticity Controller Requirements

Considered by some as the game-changing characteristic of cloud computing [34], elasticity has gained considerable research interest [7, 28, 32, 50, 55]. Most of the research on cloud elasticity has focused on the design of automated elasticity controllers. We believe there are essential characteristics for automated elasticity [3] controllers to be useful, namely,

1. Scalability: It has been estimated recently that Amazon's EC2 operates around half a million servers [30]. One single service can have up to a few thousand machines [15]. Some services run on the cloud for a short period of time while others services can run for years entirely on the cloud, e.g., reddit has been running on EC2 entirely since 2009 [37]. Algorithms used

for automated elasticity must be scalable with respect to the amount of resources running, the monitoring data analyzed and the time for which the algorithm has been running.

2. Adaptiveness: Workloads of Internet and cloud applications are dynamic [23, 5]. An automated elasticity controller should be able to adapt to the changing workload dynamics or changes in the system models, e.g., new resources added or removed from the system.

3. Rapid: An automated elasticity algorithm should compute the required capacity rapidly enough to preserve the QoS requirements. Sub-optimal decisions that preserves the QoS requirements are better than optimal decisions that might take longer to process than can be tolerated. Limited lookahead control is a very accurate technique for the estimation of the required resources but according to one study, it requires almost half an hour to come up with an accurate solution for 15 physical machines each hosting 4 Virtual Machines (VMs) [26].

4. Robustness: The changing load dynamics might lead to a change in the controller behavior [33]. A robust controller should prevent oscillations in resource allocation. While adaptiveness describes the ability of the controller to adapt to changing workloads, robustness describes the behavior of the controller with respect to its stability. A controller might be able to adapt to the workload but with oscillations in resource allocation that results in system instability. Another controller might not be able to adapt to the changing workload, but is stable with changing workload or system dynamics.

5. QoS and Cost awareness: The automated elasticity algorithm should be able to vary the capacity allocated to a service according to demand while enforcing the QoS requirements. If the algorithm provisions resources less than required, then QoS may deteriorate leading to possible losses. When the algorithm provisions extra capacity that is not needed by the service, then there is a waste of resources. In addition, the costs of the extra unneeded capacity increases the costs of running a service in the cloud.

We note that adaptivity and scalability were also identified by Padala et. al. [35] as design goals for their controller design.

2.2 Cloud Middlewares and Elasticity Aspects

Cloud middlewares typically have different components managing different functionalities such as elasticity, resource and data placement, security, monitoring, admission control, accounting and billing. We discuss the effect of having an automated elasticity component on different middleware components.

2.2.1 Monitoring

Monitoring is an integral part of datacenter management. Almost all components of a cloud middleware are dependent on the presence of reliable monitoring data. Monitoring of cloud services has been identified as a significant challenge to cloud systems adoption [20].

Elasticity algorithms are dependent on the available monitoring data since the data is used to calculate and predict the current and future load based on the monitored demand. There are significant challenges when the automated elasticity component is managed by the cloud provider. Different services have different measures of performance, e.g., response time, CPU utilization, memory utilization, network bandwidth utilization, request arrival rates or any specific metric for that particular service. The monitoring component should have the ability to monitor different metrics, including application specific metrics, for different services running on the cloud.

2.2.2 Placement

Resource placement and elasticity are two complementary problems that highly affect each other. The placement problem is concerned with the assignment of actual hardware in the datacenter to a service, i.e., where the VMs of a service run [27]. When the required capacity is predicted by an elasticity algorithm and new VMs are to be deployed, the placement component chooses where should these VMs run in the datacenter. Similarly, when the decision is to remove some VMs allocated to a service, the placement component is responsible for choosing which VMs to remove. Therefore, intelligent placement algorithms are required to make sure that the QoS does not deteriorate due to bad placement decisions, e.g., placing a VM on an already overloaded physical machine.

2.2.3 Security

Almost all the security threats to traditional online systems are present for services running on a cloud. Automatic elasticity adds a new dimensionality to Denial-of-Service attacks (DoS). DoS attacks are usually performed by saturating the servers of a service by bogus requests. In traditional online systems, when the servers are saturated, the service responsiveness deteriorates and the service may crash. In cloud environments having automated elasticity, if such attacks are not discovered early on, additional resources are added to the service to serve the bogus requests. These resources are paid for while not doing any actual work resulting in an economical Denial of Service attack [2, 25, 51].

2.2.4 Admission Control

Admission control is the process concerned with accepting new customer services. Admission control mechanisms aim to keep the cloud infrastructure

highly utilized while avoiding overloading that may results in QoS deterioration. Admission control can be easily done when all the deployed services are static and no changes occur in the amount of resources allocated to any of the services. On the other hand, for elastic applications, admission control becomes more complex since the admission control mechanism has to take into account the current and predicted future load for all services running on the cloud [56]. Careful resource overbooking can increase the profit of a cloud provider but it requires accurate elasticity predictions [52].

2.2.5 Elasticity and Accounting

Since the amount of resources allocated to a service change dynamically, the accounting component must be designed to handle these volatile resource usages [16]. Current Cloud providers typically charge for resources in billing cycles of length one hour each [48, 46]. For a public cloud, An automated elasticity algorithm should be aware of the billing cycle length. Removing a VM before the end of its billing cycle is a waste of resources since the price for that VM is already paid.

2.3 Thesis Contributions

This research on automated elasticity algorithms extends on the research done on dynamic resource provisioning that started more than a decade ago [6, 14]. Designing an automated elasticity controller that meets the desired requirements for a wide spectrum of applications and workloads is not an easy task. Most of the proposed elasticity controllers lack at least one of the identified properties. Some elasticity controller designs assume a certain model for the infrastructure and certain operating conditions [8, 28]. These controllers lack robustness against changes in the infrastructure and changes in workload dynamics. Other controller designs are not scalable with respect to time [21]. Yet other designs are not scalable with respect to the amount of resources allocated to a service [26, 49]. Some of the proposed solutions do not take into account costs associated with dropped requests [29] or neglects overprovisioning costs by not scaling down the extra resources [53]. Almost all the controllers proposed in the literature we are aware off were evaluated with less than three real workloads [29], typically one or less [21, 22, 49, 53] sometimes for a period equivalent to less than a day [21].

The first contribution of this thesis is the design of two automated adaptive hybrid elasticity controller that uses the slope of a workload to predict its future values [4, 19]. The controller's design is further extended and evaluated with additional workloads of different natures [3]. Since no controller is able to have good performance on all different workloads, our second contribution is a workload analysis and classification tool that assigns a workload to the most suitable controller out of a set of implemented elasticity controllers [5].

The assignment is calculated based on the workload characteristics and service level objectives defined by the cloud customer. Thesis contributions include scientific publications addressing the design of algorithms for cloud capacity auto-scaling and the design and implementation of a tool for workload analysis and classification. In addition, software artifacts using the proposed algorithms for auto-scaling were developed.

Chapter 3

Paper Summary and Future Work

As previously stated, the main focus of this thesis is the design and implementation of algorithms that enable the automation of cloud middlewares with an emphasis on algorithms for auto-scaling. Although all our publications assume that the middleware is for an IaaS public or private cloud, the algorithms and techniques developed are suitable for all cloud models.

3.1 Paper I

The first paper in the thesis [4] introduces two proactive elasticity algorithms that can be used to predict future workload for an application running on the cloud. Resources are then provisioned according to the controllers' predictions. The first algorithm predicts the future load based on the workload's rate of change with respect to time. The second algorithm predicts future load based on the rate of change of the workload with respect to the average provisioned capacity. The designs of the two algorithms are explained.

The paper also discusses the nine approaches to build hybrid elasticity controllers that have a reactive elasticity component and a proactive component. The reactive elasticity component is a step controller that reacts to the changes in the workload after they occur. The proactive component is a controller that has a prediction mechanism to predict future load based on the load's history. The two introduced controllers are used as the proactive component in the nine approaches discussed. Evaluation is done using webserver traces. The performances of the resulting hybrid controllers are compared and analyzed. Best practices in designing hybrid controllers are discussed. The performance of the top performing hybrid controllers is compared to a state-of-the-art hybrid elasticity controller that uses a different proactive component. In addition, the effect of the workload size on the performance of the proposed controllers is

evaluated. The proposed controller is able to reduce SLA violations by a factor of 2 to 10 compared to the state-of-the-art controller or a completely reactive controller.

3.2 Paper II

The design of the algorithms proposed in the first paper include some simplifying assumptions that ignore multiple important aspects of the cloud infrastructure and the workload's served. Aspects such as VM startup time, workload heterogeneity, and the changing request service rate of a VM are not considered in the first paper. In addition, it is assumed that delayed requests are dropped.

Paper II, [3] extends on the first paper by enhancing the cloud model used for the controller design. The new model uses a G/G/N queuing model, where N is variable, to model a cloud service provider. The queuing model is used to design an enhanced hybrid elasticity controller that takes into account the VM startup time, workload heterogeneity and the changing request service rate of a VM. The new controller allows the buffering of delayed requests and takes into account the size of the delayed requests when predicting the amount of resources required for the future load. The designed controller's performance is evaluated using webserver traces and traces from a cloud computing cluster with long running jobs. The results are compared to a controller that only has reactive components. The results show that the proposed controller reduces the cost of underprovisioning compared to the reactive controller at the cost of using more resources. The proposed controller requires a smaller buffer to keep all requests if delayed requests are not dropped.

3.3 Paper III

The third paper extends on the first two papers. The performance of the designed controllers in the first two papers varies with different workloads due to different workload characteristics. Paper III discusses the effect of different workload characteristics on the performance of different elasticity controllers. The design and implementation of an automatic workload analysis and classification tool is proposed as a solution to the performance variations. The tool can be used by cloud providers to assign workloads to elasticity controllers based on the workloads' characteristics. The tool has two main components, the analyzer and the classifier.

The analyzer analyzes a workload and extracts it periodicity and burstiness. Periodicity is measured using the autocorrelation of the workload since autocorrelation is a standard method to measure the periodicity of a workload. The use of Sample Entropy (SampEn) as a measure of burstiness is proposed. SampEn has been used in biomedical systems research for more than a decade and has proven robust. To the best of our knowledge, this is the first paper

proposing SampEn usage for characterizing bursts in cloud computing workloads. The classifier component uses a K-Nearest-Neighbors (KNN) algorithm to assign a workload to the most suitable elasticity controller based on the results from the analysis. The classifier requires training using training data. Three different training datasets are used for the training. The first set consists of 14 real workloads, the second set consists of 55 synthetic workloads and the third set consists of the previous two sets combined. The analysis results of 14 real workloads are described.

Paper III also proposes a methodology to compare the performance of an application's workload on different elasticity controllers based on a set of predefined business level objectives by the application's owner. The performance of the training set is the workloads in the training set is discussed using the proposed method. The paper then describes the training of the classifier component and the classification accuracy and results. The results show that the tool is able to assign between 92% and 98.3% of the workloads to the best suitable controller.

3.4 Future Work

There are several directions identified for future work starting from this thesis, some of which already started while others are planned. The design of more efficient cloud management systems depends on better understanding of the workloads running on the cloud systems. The workload analysis and classification component proposed in Paper III has used only two characteristics to analyze the different workloads. We have currently started investigating what other additional characteristics can be used for workload analysis. We plan to use the identified important characteristics to analyze longer workloads to better understand the evolution of a workload with time. Since the available real cloud workloads are scarce, the analysis results will be used to improve the workload generator used in Paper III to generate synthetic traces. requires The algorithms presented in Paper I and Paper II are useful for short term predictions. The proposed algorithms do not predict long term capacity requirements. Predictions of long term capacity requirements are important for different reasons such as admission control of new services and resource placement. Accurate admission controllers require some knowledge about the predicted aggregate load on the infrastructure in order to preserve QoS guarantees for the running services. Since resources are typically consolidated in a cloud, the middleware should consolidate orthogonal loads on the same physical machine in order to preserve the QoS requirements, e.g., computationally intensive workloads with predicted high peaks in CPU usage should not be consolidated on the same physical server but rather with memory intensive workloads. We are currently working on a design of an elasticity controller that can predict short term and long term capacity requirements with high accuracy. The workload analysis results will also be taken in to consideration

for the new controller's design. Resource provisioning should be based on a combination of both the short term and long term predictions.

The workload analysis and classification tool described in Paper III is used to assign workloads to elasticity algorithms. In principle, the tool can be used to assign workloads to a group of predefined classes in general, e.g., elasticity algorithms, placement algorithms or even different computing environments. We plan to extend and adapt the current tool to cover different use-cases.

Bibliography

[1] Keith Adams and Ole Agesen. A comparison of software and hardware techniques for x86 virtualization. In *ACM SIGOPS Operating Systems Review*, pages 2–13. ACM, 2006.

[2] Fahd Al-Haidari, Mohammed H Sqalli, and Khaled Salah. Enhanced EDoS-shield for mitigating EDoS attacks originating from spoofed IP addresses. In *IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 2012*, pages 1167–1174. IEEE, 2012.

[3] Ahmed Ali-Eldin, Maria Kihl, Johan Tordsson, and Erik Elmroth. Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control. In *Proceedings of the 3rd workshop on Scientific Cloud Computing Date*, pages 31–40. ACM, 2012.

[4] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE, 2012.

[5] Ahmed Ali-Eldin, Johan Tordsson, Erik Elmroth, and Maria Kihl. Workload classification for efficient cloud infrastructure elasticity control. Technical report, UMINF 13.13, Umeå University, 2013.

[6] Guillermo A Alvarez, Elizabeth Borowsky, Susie Go, Theodore H Romer, Ralph Becker-Szendy, Richard Golding, Arif Merchant, Mirjana Spasojevic, Alistair Veitch, and John Wilkes. Minerva: An automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems (TOCS)*, 19(4):483–518, 2001.

[7] Ganesh Ananthanarayanan, Christopher Douglas, Raghu Ramakrishnan, Sriram Rao, and Ion Stoica. True elasticity in multi-tenant data-intensive compute clusters. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 24. ACM, 2012.

[8] Ala Arman, Ahmad Al-Shishtawy, and Vladimir Vlassov. Elasticity controller for cloud-based key-value stores. In *Parallel and Distributed Systems*

(ICPADS), 2012 IEEE 18th International Conference on, pages 268–275. IEEE, 2012.

[9] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[10] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *ACM SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[11] Carsten Binnig, Donald Kossmann, Tim Kraska, and Simon Loesing. How is the weather tomorrow?: towards a benchmark for the cloud. In *Proceedings of the Second International Workshop on Testing Database Systems*, page 9. ACM, 2009.

[12] Nathanael Burton. "Keynote: OpenStack at the National Security Agency (NSA)". Accessed: May, 2013, http://www.openstack.org/summit/portland-2013/session-videos/presentation/keynote-openstack-at-the-national-security-agency-nsa.

[13] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[14] Jeffrey S Chase, Darrell C Anderson, Prachi N Thakar, Amin M Vahdat, and Ronald P Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116. ACM, 2001.

[15] CycleComputing. New CycleCloud HPC Cluster Is a Triple Threat, September 2011. http://blog.cyclecomputing.com/2011/09/new-cyclecloud-cluster-is-a-triple-threat-30000-cores-massive-spot-instances-grill-chef-monitoring-g.html.

[16] Erik Elmroth, Fermin Galan Marquez, Daniel Henriksson, and David Perales Ferrera. Accounting and billing for federated cloud infrastructures. In *Eighth International Conference on Grid and Cooperative Computing, 2009. GCC'09.*, pages 268–275. IEEE, 2009.

[17] EN Mootaz Elnozahy, Michael Kistler, and Ramakrishnan Rajamony. Energy-efficient server clusters. In *Power-Aware Computer Systems*, pages 179–197. Springer, 2003.

[18] Benjamin Farley, Ari Juels, Venkatanathan Varadarajan, Thomas Ristenpart, Kevin D Bowers, and Michael M Swift. More for your money: Exploiting performance heterogeneity in public clouds. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 20. ACM, 2012.

[19] Ana Juan Ferrer, Francisco Hernandez, Johan Tordsson, Erik Elmroth, Ahmed Ali-Eldin, Csilla Zsigri, Raul Sirvent, Jordi Guitart, Rosa M. Badia, Karim Djemame, Wolfgang Ziegler, Theo Dimitrakos, Srijith K. Nair, George Kousiouris, Kleopatra Konstanteli, Theodora Varvarigou, Benoit Hudzia, Alexander Kipp, Stefan Wesner, Marcelo Corrales, Nikolaus Forgo, Tabassum Sharif, and Craig Sheridan. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012.

[20] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008.

[21] Waheed Iqbal, Matthew N Dailey, David Carrera, and Paul Janecek. Adaptive resource provisioning for read intensive multi-tier applications in the cloud. *Future Generation Computer Systems*, 27(6):871–879, 2011.

[22] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155–162, 2012.

[23] Xiaozhu Kang, Hui Zhang, Guofei Jiang, Haifeng Chen, Xiaoqiao Meng, and Kenji Yoshihira. Understanding internet video sharing site workload: A view from data center design. *Journal of Visual Communication and Image Representation*, 21(2):129–138, 2010.

[24] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[25] Soon Hin Khor and Akihiro Nakao. spow: On-demand cloud-based eddos mitigation mechanism. In *Fifth Workshop on Hot Topics in System Dependability*, 2009.

[26] Dara Kusic, Jeffrey O Kephart, James E Hanson, Nagarajan Kandasamy, and Guofei Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[27] Wubin Li, Johan Tordsson, and Erik Elmroth. Virtual machine placement for predictable and time-constrained peak loads. In *Economics of Grids, Clouds, Systems, and Services*, pages 120–134. Springer Berlin Heidelberg, 2012.

[28] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, pages 1–10. ACM, 2010.

[29] Minghong Lin, Adam Wierman, Lachlan LH Andrew, and Eno Thereska. Online dynamic capacity provisioning in data centers. In *49th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2011*, pages 1159–1163. IEEE, 2011.

[30] Huan Liu. "Amazon data center size". Accessed: May, 2013, http://huanliu.wordpress.com/2012/03/13/amazon-data-center-size/.

[31] Peter Mell and Timothy Grance. The NIST definition of cloud computing. *NIST special publication*, 800:145, 2011.

[32] Shicong Meng, Ling Liu, and Vijayaraghavan Soundararajan. Tide: achieving self-scaling in virtualized datacenter management middleware. In *Proceedings of the 11th International Middleware Conference Industrial track*, pages 17–22. ACM, 2010.

[33] Manfred Morari. Robust stability of systems with integral control. *IEEE Transactions on Automatic Control*, 30(6):574–577, 1985.

[34] Dustin Owens. Securing elasticity in the cloud. *Commun. ACM*, 53(6):46–51, June 2010.

[35] Pradeep Padala, Kai-Yuan Hou, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, and Arif Merchant. Automated control of multiple virtualized resources. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 13–26. ACM, 2009.

[36] David Price. Is Cloud Computing Still a Dream? Accessed: May, 2013, http://www.pcworld.com/article/221268/Is_Cloud_Computing_Still_a_Dream.html.

[37] Amazon AWS. AWS Case Study: reddit. Accessed: May, 2013, http://aws.amazon.com/solutions/case-studies/reddit/.

[38] Amazon Web Service. "Case Studies". Accessed: May, 2013, http://aws.amazon.com/solutions/case-studies/.

[39] Amazon Web Services. "Amazon EC2 instances". Accessed: May, 2013, http://aws.amazon.com/ec2/instance-types/.

[40] Equinix. "Rethink Your Storage Strategy for the Digital Economy". Accessed: May, 2013, http://www.equinix.com/solutions/partner-solutions/netapp/.

[41] G-Cloud. "The G-Cloud Programme". Accessed: May, 2013, http://gcloud.civilservice.gov.uk/.

[42] Google. " google app engine". Accessed: May, 2013, https://developers.google.com/appengine/.

[43] Microsofot. Windows Azure. Accessed: May, 2013, http://www.windowsazure.com/en-us/.

[44] Oracle. "Oracle Cloud". Accessed: May, 2013, https://cloud.oracle.com/mycloud/f?p=service:home:0.

[45] Rackspace. " The Rackspace Cloud". Accessed: May, 2013, http://www.rackspace.com/cloud.

[46] Rackspace. "cloud servers pricing". Accessed: May, 2013, http://www.rackspace.com/cloud/servers/pricing/.

[47] Salesforce. "CRM and Cloud Computing To Grow Your Business". Accessed: May, 2013, http://www.salesforce.com/.

[48] Windows Azure. "pricing at-a-glance". Accessed: May, 2013, http://www.windowsazure.com/en-us/pricing/overview/.

[49] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. Efficient autoscaling in the cloud using predictive models for workload forecasting. In *IEEE International Conference on Cloud Computing (CLOUD), 2011*, pages 500–507. IEEE, 2011.

[50] Dan Schatzberg, Jonathan Appavoo, Orran Krieger, and Eric Van Hensbergen. Why elasticity matters. Technical report, Boston University, 2012.

[51] Mohammed H Sqalli, Fahd Al-Haidari, and Khaled Salah. EDoS-shield-a two-steps mitigation technique against EDoS attacks in cloud computing. In *Fourth IEEE International Conference on Utility and Cloud Computing (UCC), 2011*, pages 49–56. IEEE, 2011.

[52] Luis Tomas and Johan Tordsson. Improving cloud infrastructure utilization through overbooking. In *The ACM Cloud and Autonomic Computing Conference (CAC 2013), to appear*, 2013.

[53] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1):1, 2008.

[54] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 281–292. ACM, 2012.

[55] David Villegas, Athanasios Antoniou, Seyed Masoud Sadjadi, and Alexandru Iosup. An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2012*, pages 612–619. IEEE, 2012.

[56] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. Sla-based admission control for a software-as-a-service provider in cloud computing environments. *Journal of Computer and System Sciences*, 78(5):1280–1299, 2012.

[57] Lenar Yazdanov and Christof Fetzer. Vertical scaling for prioritized vms provisioning. In *Second International Conference on Cloud and Green Computing (CGC), 2012*, pages 118–125. IEEE, 2012.

[58] Katherine Yelick, Susan Coghlan, Brent Draney, and Richard Shane Canon. The magellan report on cloud computing for science. Technical report, Technical report, US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), 2011.

I

Paper I

An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures*

Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth

Dept. Computing Science, Umeå University, SE-901 87 Umeå, Sweden
{ahmeda,tordsson,elmroth}@cs.umu.se
www.cloudresearch.org

Abstract: Cloud elasticity is the ability of the cloud infrastructure to rapidly change the amount of resources allocated to a service in order to meet the actual varying demands on the service while enforcing SLAs. In this paper, we focus on horizontal elasticity, the ability of the infrastructure to add or remove virtual machines allocated to a service deployed in the cloud. We model a cloud service using queuing theory. Using that model we build two adaptive proactive controllers that estimate the future load on a service. We explore the different possible scenarios for deploying a proactive elasticity controller coupled with a reactive elasticity controller in the cloud. Using simulation with workload traces from the FIFA world-cup web servers, we show that a hybrid controller that incorporates a reactive controller for scale up coupled with our proactive controllers for scale down decisions reduces SLA violations by a factor of 2 to 10 compared to a regression based controller or a completely reactive controller.

An Adaptive Hybrid Elasticity Controller for Cloud Infrastructures

Ahmed Ali-Eldin, Johan Tordsson and Erik Elmroth
Department of Computing Science, Umeå University
Umeå, Sweden
Email:{ahmeda, tordsson, elmroth}@cs.umu.se

***Abstract*—Cloud elasticity is the ability of the cloud infrastructure to rapidly change the amount of resources allocated to a service in order to meet the actual varying demands on the service while enforcing SLAs. In this paper, we focus on horizontal elasticity, the ability of the infrastructure to add or remove virtual machines allocated to a service deployed in the cloud. We model a cloud service using queuing theory. Using that model we build two adaptive proactive controllers that estimate the future load on a service. We explore the different possible scenarios for deploying a proactive elasticity controller coupled with a reactive elasticity controller in the cloud. Using simulation with workload traces from the FIFA world-cup web servers, we show that a hybrid controller that incorporates a reactive controller for scale up coupled with our proactive controllers for scale down decisions reduces SLA violations by a factor of 2 to 10 compared to a regression based controller or a completely reactive controller.**

I. Introduction

With the advent of large scale data centers that host outsourced IT services, cloud computing [1] is becoming one of the key technologies in the IT industry. A cloud is an elastic execution environment of resources involving multiple stakeholders and providing a metered service at a specified level of quality [2]. One of the major benefits of using cloud computing compared to using an internal infrastructure is the ability of the cloud to provide its customers with elastic resources that can be provisioned on demand within seconds or minutes. These resources can be used to handle flash crowds. A flash crowd, also known as a slashdot effect, is a surge in traffic to a particular service that causes the service to be virtually unreachable [3]. Flash crowds are very common in today's networked world. Figure I shows the traces of the FIFA 1998 world cup website. Flash crowds occur frequently before and after matches. In this work, we try to automate and optimize the management of flash crowds in the cloud by developing an autonomous elasticity controller.

Autonomous elastic cloud infrastructures provision resources according to the *current* actual demand on the infrastructure while enforcing service level agreements (SLAs). Elasticity is the ability of the cloud to rapidly vary the allocated resource capacity to a service according to the current load in order to meet the quality of service (QoS) requirements specified in the SLA agreements. Horizontal elasticity is the ability of the cloud to rapidly increase or decrease the number of virtual machines (VMs) allocated to a service according to the current load. Vertical elasticity is the ability of the cloud to change the hardware configuration of VM(s) already running to increase or decrease the total amount of resources allocated to a service running in the cloud.

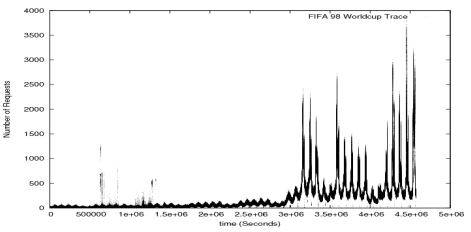


Fig. 1. Flash crowds illustrating the rapid change in demand for the FIFA world cup website.

Building elastic cloud infrastructures that scale up and down with the actual demand of the service is a problem far from being solved [2]. Scale up should be fast enough in order to prevent breaking any SLAs while it should be as close as possible to the actual required load. Scale down should not be premature, i.e., scale down should occur when it is anticipated that the service does not need these resources in the near future. If scale down is done prematurely, resources are allocated and deallocated in a way that causes oscillations in the system. These resource oscillations introduce problems to load balancers and add some extra costs due to the frequent release and allocation of resources [4]. In this paper we develop two adaptive horizontal elasticity controllers that control scale up and scale down decisions and prevent resource oscillations.

This paper is organized as follows; in Section II, we describe the design of our controllers. In Section III we describe our simulation framework, our experiments and discuss our results. In Section IV we describe some approaches to building elasticity controllers in the literature. We conclude in Section V.

II. Building an Elastic Cloud Controller

In designing our controllers, we view the cloud as a control system. Control systems are either closed loop or open loop systems [5]. In an open loop control system, the control action does not depend on the system output making open loop control generally more suited for simple applications where no

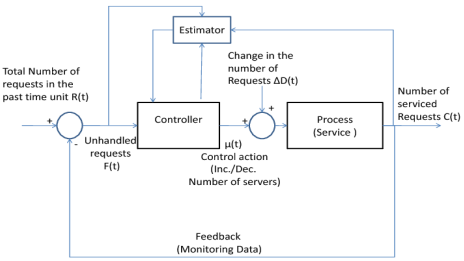


Fig. 2. Adaptive Proactive Controller Model.

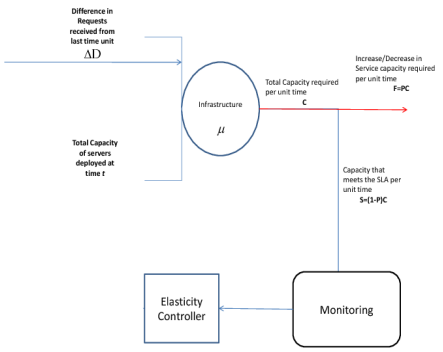


Fig. 3. Queuing Model for a service deployed in the cloud.

feedback is required and no system monitoring is performed. Contrarily, a closed loop control system is more suited for sophisticated application as the control action depends on the system output and on monitoring some system parameters. The general closed-loop control problem can be stated as follows: The controller output $\mu(t)$ tries to force the system output $C(t)$ to be equal to the reference input $R(t)$ at any time t irrespective of the disturbance ΔD. This general statement defines the main targets of any closed loop control system irrespective of the controller design.

In this work, we model a service deployed in the cloud as a closed loop control system. Thus, the horizontal elasticity problem can be stated as follows: The elasticity controller output $\mu(t)$ should add or remove VMs to ensure that the number of service requests $C(t)$ is equal to the total number of requests received $R(t) + \Delta D(t)$ at any time unit t with an error tolerance specified in the SLA irrespective of the change in the demand ΔD while maintaining the number of VMs to a minimum. The model is simplified by assuming that servers start up and shut down instantaneously.

We design and build two adaptive proactive controllers to control the QoS of a service as shown in Figure 2. We add an estimator to adapt the output of the controller with any change in the system load and the system model.

A. Modeling the state of the service

Figure 3 shows a queuing model representing the cloud infrastructure. The infrastructure is modeled as a $G/G/N$ stable queue in which the number of servers N required is variable [6]. In the model, the total number of requests per second R_{total} is divided into two inputs to the infrastructure, the first input $R(t)$ represents the total amount of requests the infrastructure is capable of serving during time unit t. The second input, ΔD represents the change in the number of requests from the past time unit. Since the system is stable, the output of the queue is the total service capacity required per unit time and is equal to R_{total}. P represents the increase or decrease in the number of requests to the current service capacity $R(t)$.

The goal of a cloud provider is to provide all customers with enough resources to meet the QoS requirements specified in the SLA agreements while reducing over provisioning to a minimum. The cloud provider monitors a set of parameters stated in the SLA agreements. These parameters represent the controlled variables for the elasticity controller. Our controllers are parameter independent and can be configured to use any performance metric as the controlled parameter. For the evaluation of our controllers, we choose the number of concurrent requests received for the past time unit to be the monitored parameter because this metric shows both the amounts of over provisioned and under provisioned resources which is an indicator to the costs incurred due to the elasticity engine. Most of the previous work on elasticity considers response time to be the controlled parameter. Response time is software and hardware dependent and is not well suited for comparing the quality of different elasticity approaches [7].

B. Estimating future usage

From Figure 3, the total future service capacity required per unit time, $C(t+1)$, is $C(t+1) = \Delta D(t) + R(t)$, where $R(t)$ is the current service capacity and $\Delta D(t)$ is the change in the current service capacity required in order to meet the SLA agreement while maintaining the number of VMs to minimum. A successful proactive elasticity engine is able to estimate the change in future demand $\Delta D(t)$ and add or remove VMs based on this proactive estimation. $\Delta D(t)$ can be estimated by

$$\Delta D(t) = P(t)C(t) \tag{1}$$

where $P(t)$ is the gain parameter of the controller. $P(t)$ is positive if there is an increase in the number of requests, negative if there is a decrease in the number of requests, or zero if the number of requests is equal to the current service capacity.

We define $\widehat{C}$ to be the infrastructure's average periodical service rate over the past T_d time units. $\widehat{C}$ is calculated

for the whole infrastructure and not for a single VM. Thus, $\widehat{C} = \frac{\sum_i^{T_d} n_i t_i}{T_d}$, where T_d is a system parameter specifying the period used for estimating the average periodical service rate and t_i is the time for which the number of requests received per unit time for the whole infrastructure stay constant at n_i requests per unit time before the demand changes. Thus, $\sum_i^{T_d} t_i = T_d$. We also define $\overline{n}$, the average service rate over time as $\overline{n} = \frac{\sum_t n(t)}{T}$.

From equation 1 and since the system is stable ,

$$F = \widehat{C}\ P, \tag{2}$$

where F, the estimated increase or decrease of the load, is calculated using the gain parameter of the controller P every time unit. The gain parameter represents the estimated rate of adding or removing VMs. We design two different controllers with two different gain parameters.

For the first controller P_{C1}, the gain parameter P_1 is chosen to be the periodical rate of change of the system load,

$$P_1 = \frac{\Delta D_{T_d}}{T_D}. \tag{3}$$

As the workload is a non-linear function in time, the periodical rate of change of the load is the derivative of the workload function during a certain period of time. Thus, the gain parameter represents the load function changes over time.

For the second controller P_{C2}, the gain parameter P_2 is the ratio between the change in the load and the average system service rate over time,

$$P_2 = \frac{\Delta D_{T_d}}{\overline{n}}. \tag{4}$$

This value represents the load change with respect to the average capacity. By substituting this value for P in Equation 1, the predicted load change is the ratio between the current service rate and the average service rate multiplied by the change in the demand over the past estimation period.

C. Determining suitable estimation intervals

The interval between two estimations, T_d, represents the period for which the estimation is valid, is a crucial parameter affecting the controller performance. It is used for calculating $\widehat{C}$ for both controllers and for P_1 in the case of the first controller. T_d controls the controllers' reactivity. If T_d is set to one time unit, the estimations for the system parameters are done every time unit and considers only the system load during past time unit. At the other extreme, if T_d is set to ∞, the controller does not perform any predictions at all. As the workload observed in data centers is dynamic [8], setting an adaptive value for T_d that changes with the load dynamics is one of our goals.

We define K to be the tolerance level of a service i.e. the number of requests the service does not serve on time before making a new estimation, in other words,

$$T_d = \frac{K}{\widehat{C}}. \tag{5}$$

TABLE I
OVERVIEW OF THE NINE DIFFERENT WAYS TO BUILD A HYBRID CONTROLLER.

Engine Name	Scale up mechanism	Scale down mechanism
UR-DR	Reactive	Reactive
UR-DP	Reactive	Proactive
UR-DRP	Reactive	Reactive and Proactive
URP-DRP	Reactive and Proactive	Reactive and Proactive
URP-DR	Reactive and Proactive	Reactive
URP-DP	Reactive and Proactive	Proactive
UP-DP	Proactive	Proactive
UP-DR	Proactive	Reactive
UP-DRP	Proactive	Reactive and Proactive

K is defined in the SLA agreement with the service owner. If K is specified to be zero, T_d should always be kept lower than the maximum response time to enforce that no requests are served slower by the system.

D. An elasticity engine for scale-up and scale-down decisions

The main goal of any elasticity controller is to enforce the SLAs specified in the SLA agreement. For today's dynamical network loads [3], it is very hard to anticipate when a flash crowd is about to start. If the controller is not able to estimate the flash crowd on time, many SLAs are likely to be broken before the system can adapt to the increased load.

Previous work on elasticity considers building hybrid controllers that combines reactive and proactive controllers [9] and [10]. We extend on this previous work and consider all possible ways of combining reactive and proactive controllers for scaling of resources in order to meet the SLAs. We define an elasticity engine to be an elasticity controller that considers both scale-up and scale-down of resources. There are nine approaches in total to build an elasticity engine using a reactive and a proactive controller. These approaches are listed in Table I. Some of these combinations are intuitively not good, but for the sake of completeness we evaluate the results of all of these approaches. In order to facilitate our discussion, we use the following naming convention to name an elasticity engine; an elasticity engine consists of two controllers, a scale up (U) and a scale down (D) controller. A controller can be either reactive (R) or proactive (P). P_{C1} and P_{C2} are a special case from proactive controllers e.g. URP-DRP elasticity engine has a reactive and proactive controller for scale up and scale down while a UR-DP$_{C1}$ is an engine having a reactive scale up controller and P_{C1} for scale down.

III. EXPERIMENTAL EVALUATION

In order to validate the controllers, we designed and built a discrete event simulator that models a service deployed in the cloud. The simulator is built using Python. We used the complete traces from the FIFA 1998 world cup as input to our model [11]. The workload contains 1.3 billion Web requests recorded in the period between April 30, 1998 and July 26, 1998. We have calculated the aggregate number of requests per second from these traces. They are by far the most used traces in the literature. As these traces are quite old, we multiply the

number of requests received per unit time by a constant in order to scale up these traces to the orders of magnitude of today's workloads. Although there are newer traces available such as the Wikipedia trace [12], but they do not have the number of peaks seen in the FIFA traces. We assume perfect load balancing and quantify the performance of the elasticity engines only.

A. Nine Approaches to build an elasticity engine

In this experiment we evaluate the nine approaches to a hybrid controller and quantify their relative performance using P_{C1} and P_{C2}. We use the aggregate number of requests per unit time from the world cup traces multiplied by a constant equal to 50 as input to our simulator. This is almost the same factor by which the number of Internet users increased since 1997 [13]. To map the number of service requests to the number of servers, we assume that each server can serve up to 500 requests per unit time. This number is an average between the number of requests that can be handled by a Nehalem Server running the MediaWiki application [14] and a Compaq ProLiant DL580 server running a database application [15]. We assume SLAs that specify the maximum number of requests not handled per unit time to be fewer than 5% of the maximum capacity of one server.

The reactive controller is reacting to the current load while the proactive controller is basing its decision on the history of the load. Whenever a reactive controller is coupled with a proactive controller and the two controllers give contradicting decisions, the decision of the reactive controller is chosen. For the UR-DR controller, scale down is only done if the number of unused servers is greater than two servers in order to reduce oscillations.

To compare all the different approaches, we monitor and sum the number of servers the controllers fail to provision on time to handle the increase in the workload, S^-. This number can be viewed as the number of extra servers to be added to avoid breaking all SLAs, or as the quality of estimation. $\overline{S^-}$ is the average number of requests the controller fails to provision per unit time. Similarly, we monitor the number of extra servers deployed by the infrastructure at any unit time. The summation of this number indicates the provisioned unused server capacity, S^+. $\overline{S^+}$ is the averaged value over time. These two aggregate metrics are used to compare the different approaches.

Table II shows the aggregate results when P_{C1} and P_{C2} are used for the proactive parts of the hybrid engine. The two right-most columns in the table show the values of S^- and S^+ as percentages of the total number of servers required by the workload respectively. We compare the different hybridization approaches with a UR-DR elasticity engine [16].

The results shown in the two tables indicate that using an UR-DP$_{C2}$ engine reduces S^- by a factor of 9.1 compared to UR-DR elasticity engine, thus reducing SLA violations by the same ratio. This comes at the cost of using 14.33% extra servers compared to 1.4% in the case of a UR-DR engine. Similar results are obtained using a URP$_{C2}$-DP$_{C2}$ engine. These results are obtained because the proactive scale down controller does not directly release resources when the load decreases instantaneously but rather makes sure that this decrease is not instantaneous. Using a reactive controller for scale down on the other hand reacts to any load drop by releasing resources. It is also observed that the second best results are obtained using an UR-DP$_{C1}$ elasticity engine. This setup reduces S^- by a factor of 4, from 1.63% to 0.41% compared to a UR-DR engine at the expense of increasing the number of extra servers used from 1.4% to 9.44%.

A careful look at the table shows that elasticity engines with reactive components for both scale up and scale down show similar results even when a proactive component is added. We attribute this to the premature release of resources due to the reactivity component used for the scale down controller. The premature release of resources causes the controller output to oscillate with the workload. The worst performance is seen when a proactive controller is used for scale up with a reactivity component in the scale down controller. This engine is not able to react to sudden workload surges. In addition it releases resources prematurely.

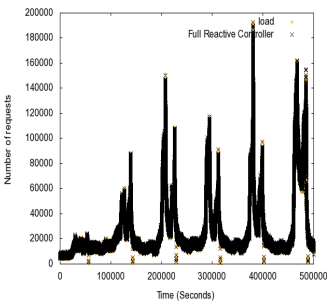
Figures 4(a), 4(b) and 4(c) shows the performance of a UR-DR, UR-DP$_{C1}$ and a UR-DP$_{C2}$ elasticity engines over part of the trace from 06:14:32, the 21^{st} of June,1998 to 01:07:51 27^{th} of June,1998. Figures 4(d), 4(e) and 4(f) shows an in depth view of the period between 15:50:00 the 23^{rd} of June, 1998 till 16:07:00 on the same day (between time unit 208349 and 209349 on the left hand side figures).

The UR-DR elasticity engine releases resources prematurely as seen in Figure 4(d). These resources are then reallocated when there is an increase in demand causing resource allocation and deallocation to oscillate. The engine is always following the demand but is never ahead. On the other hand, figures 4(e) and 4(f) show different behavior where the elasticity engine tries not to deallocate resources prematurely in order to prevent oscillations and to be ahead of the demand. It is clear in Figure 4(f) that the elasticity engine estimates the future load dynamics and forms an *envelope* over the load. An envelope is defined as the smooth curve that takes the general shape of the load's amplitude and passes through its peaks [17]. This delay in the deallocation comes at the cost of using more resources. These extra resources improve the performance of the service considerably as it will be always ahead of the load. We argue that this additional cost is well justified considering the gain in service performance.

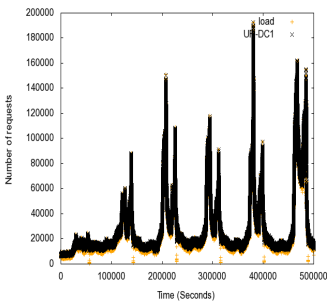
1) Three classes of SLAs: An infrastructure provider can have multiple controller types for different customers and different SLA agreements. The results shown in table II suggest having three classes of customers namely, gold, silver and bronze. A gold customer pays more in order to get the best service at the cost of some extra over-provisioning and uses a UR-DP$_{C2}$ elasticity engine. A silver customer uses the UR-DP$_{C1}$ elasticity engine to get good availability while a bronze customer uses the UR-DR and gets a reduced, but acceptable, QoS but with very little over-provisioning. These three different elasticity engines with different degrees

TABLE II
S^- AND S^+ FOR P_{C1} AND P_{C2}

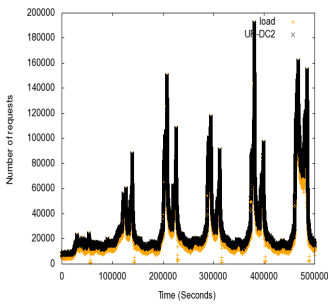
Name	S^-	$\overline{S^-}$	S^+	$\overline{S^+}$	$S^-\%$	$S^+\%$
UR-DR	-1407732	-0.3	120641	0.026	-1.63%	1.4%
P_{C1} results						
UR-DP$_{C1}$	-354814	-0.077	8159220	1.78	-0.41%	9.44%
UR-DRP$_{C1}$	-1412289	-0.3	1202806	0.26	-1.63%	1.4%
URP$_{C1}$-DRP$_{C1}$	-1411678	-0.3	1203170	0.26	-1.63%	1.4%
URP$_{C1}$-DR	-1407036	-0.3	1206391	0.26	-1.62%	1.4%
URP$_{C1}$-DP$_{C1}$	-354127	-0.077	8160627	1.78	-0.41%	9.4%
UP$_{C1}$-DP$_{C1}$	-4147953	-0.9	1827431	0.399	-4.8%	2.1%
UP$_{C1}$-DR	-8474040	-1.85	408447	0.399	-9.8%	2.1%
UP$_{C1}$-DRP$_{C1}$	-11408704	-2.49	190427.0	0.041	-10%	0.27%
P_{C2} results						
UR-DP$_{C2}$	-159029	-0.0347	12386346.0	2.7	-0.18%	14.33%
UR-DRP$_{C2}$	-1418949.0	-0.31	1176239.0	0.257	-1.64%	1.36%
URP$_{C2}$-DRP$_{C2}$	-1419269.0	-0.31	1175393.0	0.257	-1.64%	1.35%
URP$_{C2}$-DR	-1407732.0	-0.31	1206407.0	0.263	-1.63%	1.4%
URP$_{C2}$-DP$_{C2}$	-159029	-0.0347	12386346.0	2.707	-0.18%	14.33%
UP$_{C2}$-DP$_{C2}$	-4350841.0	-0.951	2216866.0	0.485	-5.03%	2.6%
UP$_{C2}$-DR	-11245521	-2.458	396697	0.0867	-13%	0.46%
UP$_{C2}$-DRP$_{C2}$	-11408704	2.49	190427	0.0416	-13.2%	0.22%



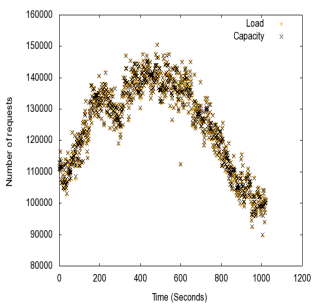
(a) UR-DR performance in a period of 6 days.



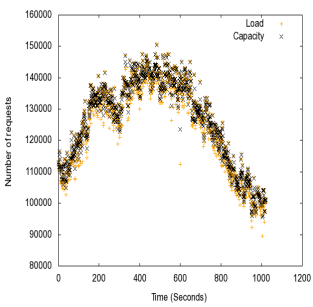
(b) UR-DP$_{C1}$ performance in a period of 6 days.



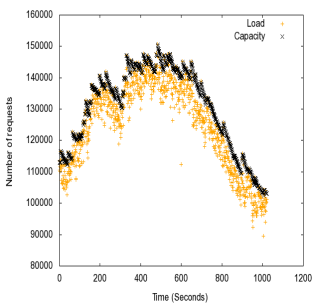
(c) UR-DP$_{C2}$ performance in a period of 6 days.



(d) UR-DR: Zooming on a period of 17 minutes.



(e) UR-DP$_{C1}$: Zooming on a period of 17 minutes.



(f) UR-DP$_{C2}$: Zooming on a period of 17 minutes.

Fig. 4. Performance of UR-DR, UR-DP$_{C1}$ and, UR-DP$_{C2}$ elasticity engines with time: The Figures show how the different engines detect future load. It can be observed that the UR-DR engine causes the capacity to oscillate with the load while UR-DP$_{C1}$ and UR-DP$_{C2}$ predict the envelope of the workload.

TABLE III
COMPARISON BETWEEN THE UR-DREGRESSION, UR-DP$_{C1}$, UR-DP$_{C2}$, AND UR-DR ELASTICITY ENGINES

Name	S^-	S^+	S^-	S^+
UR-DRegression	-74791.7	1568047.8	-2.24%	47%
UR-DP$_{C1}$	-50307.2	1076236.3	-1.51%	32.24%
UR-DP$_{C2}$	-35818.6	1326841.7	-1.07%	39.75%
UR-DR	-99801.8	653082.9	-2.99%	19.57%

of over provisioning and qualities of estimation give cloud providers convenient tools to handle customers of different importance classes and thus increase their profit and decrease their penalties. Current cloud providers usually have a general SLA agreement for all their customers. RackSpace [18] for example guarantees 100% availability with a penalty equal to 5% of the fees for each 30 minutes of network or data center downtime for the cloud servers. It guarantees 99.9% availability for the cloud files. The network is considered not available in case of [18]: (i) The Rackspace Cloud network is down, or (ii) the Cloud Files service returns a server error response to a valid user request during two or more consecutive 90 second intervals, or (iii) the Content Delivery Network fails to deliver an average download time for a 1-byte reference document of 0.3 seconds or less, as measured by The Rackspace Cloud's third party measuring service. For an SLA similar to the RackSpace SLA or Amazon S3 [19], using one of our controllers significantly reduces penalties paid due to server errors, allowing the provider to increase profit.

B. Comparison with regression based controllers

In this experiment we compare our controllers with the controller designed by Iqbala et al. [10] who design a hybrid elasticity engine with a reactive controller for scale-up decisions and a predictive controller for scale-down decisions. When the capacity is less than the load, a scale up decision is taken and new VMs are added to the service. For scale down, their predictive component uses second order regression. The regression model is recomputed for the full history every time a new measurement data is available. If the current load is less than the provisioned capacity for k time units, a scale down decision is taken using the regression model. If the predicted number of servers is greater than the current number of servers, the result is ignored. Following our naming convention, we denote their engine UR-DRegression. As regression is recomputed every time a new measurement data is available on the full history, simulation using the whole world cup traces would be time consuming. Instead, in this experiment we used part of the trace from 09:47:41 on the 13^{th} of May, 1998 to 17:02:49 on the 25^{th} of May, 1998. We multiply the number of concurrent requests by 10 and assume that the servers can handle up to 100 requests. We assume that the SLA requires that a maximum of 5% of the capacity of a single server is not serviced per unit time.

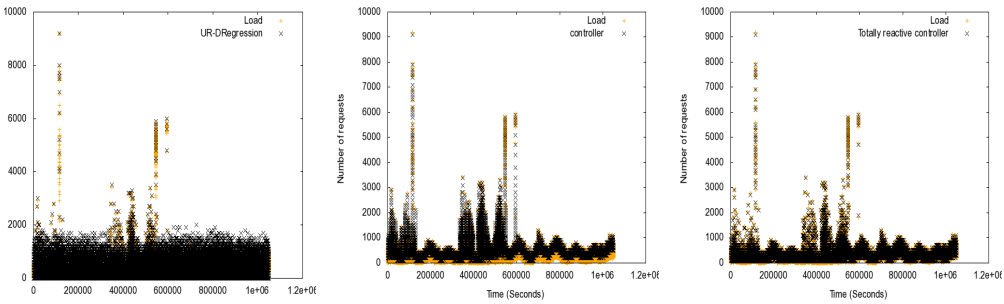
Table III shows the aggregated results for four elasticity engines; UR-DRegression, UR-DP$_{C1}$, UR-DP$_{C2}$ and UR-DR. Although all the proactive approaches reduce the value of S^- compared to a UR-DR engine, P$_{C2}$ still shows superior results. The number of unused server that get provisioned by the regression controller S^+ is 50% more than for P$_{C1}$ and 15% more than P$_{C2}$ although both P$_{C1}$ and P$_{C2}$ reduces S^- more. The $UR-DR$ controller has a higher SLA violation rate (3%) while maintaining a much lower over-provisioning rate (19.57%). As we evaluate the performance of the controller on a different part of the workload and we multiply the workload by a different factor, the percentages of the capacity the controller fail to provision on time and the unused provisioned capacity changed from the previous experiment.

Figures 5(a), 5(b) and 5(c) show the load compared to the controller outputs for the UR-DR, UR-DP$_{C2}$, and UR-DRegression approaches. The amount of unused capacity using a regression based controller is much higher than the unused capacity for the other controllers. The controller output for the UR-DRegression engine completely over-estimates the load causing prediction oscillations between the crests and the troughs. One of the key advantages of P$_{C1}$ and P$_{C2}$ is that they depend on simple calculations. They are both scalable with time compared to the regression controller. The highest observed estimation time for the UR-DRegression is 6.5184 seconds with an average of 0.97695 seconds compared to 0.000512 seconds with an average of 5.797×10^{-6} in case of P$_{C1}$ and P$_{C2}$.

C. Performance impact of the workload size

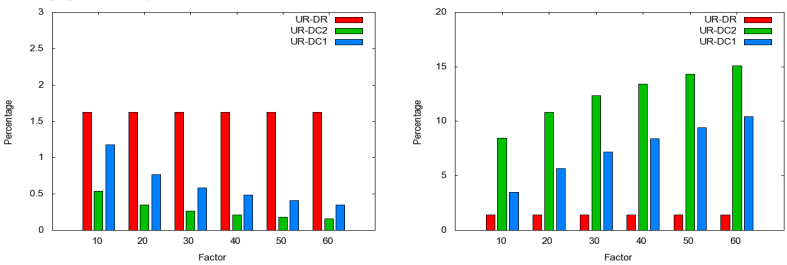
In this experiment we investigate the effect of changing the load and server power on the performance of our proposed elasticity engines. We constructed six new traces using the world cup workload traces by multiplying the number of requests per second in the original trace by a factor of 10, 20, 30, 40, 50, and 60. We ran experiments with the new workloads using the UR-DR, UR-DP$_{C1}$ and UR-DP$_{C2}$ elasticity engines. For each simulation run, we assume that the number of requests that can be handled by any server is 10 times the factor by which we multiplied the traces, e.g., for an experiment run using a workload trace where the number of requests is multiplied by 20, we assume that the server capacity is up to 200 requests per second. We also assume that for each experiment the SLA specifies the maximum unhandled number of requests to be 5% of the maximum capacity of a single server.

Figure 6(a) shows the percentage of servers the engines failed to provision on time to handle the increase in demand for each workload size (S^-) while Figure 6(b) shows the percentages of extra servers provisioned for each workload size. It is clear that the UR-DR engine exhibits the same performance with changing workloads. For the UR-DP$_{C1}$ and the UR-DP$_{C2}$ engines on the other hand, the performance depends on the workload and the server capacity. As the factor by which we multiply the workload increases, the percentage of servers the two engines failed to provision on time decreases. Inversely, the percentage of extra servers provisioned increases. These results indicate that the quality of estimation changes with any change in the workload. We



(a) UR-DRegression elasticity engine. (b) UR-DP$_{C2}$ elasticity engine. (c) UR-DR elasticity engine

Fig. 5. Performance Comparison of UR-DR, UR-DP$_{C2}$ and UR-DRegression elasticity engines. The UR-DRegression controller over-provisions many servers to cope with the changing workload dynamics.



(a) The effect of changing load size on the percentage of S^- to the total number of servers. (b) The effect of changing load size on the percentage of S^+ to the total number of servers.

Fig. 6. The effect of changing the workload size and the server capacity on the UR-DR, UR-DP$_{C1}$ and UR-DP$_{C2}$ elasticity engines.

attribute the improvement in the quality of estimation when the load increases using the UR-DP$_{C1}$ and UR-DP$_{C2}$ engines to the ability of both estimators to predict the envelope of the workload, thus decreasing the number of prematurely deallocated resources. Although the number of requests increases in the different workloads, the number of times the controllers deallocate resources prematurely also increases, but at a slower rate than the load. We have performed similar experiments with the Wikipedia traces [12] and obtained similar results [20]. Due to lack of space we omit those results.

D. Discussion

Although our proactive controllers P$_{C1}$ and P$_{C2}$ are designed using the change in the load as the controller parameter, they can be generalized to be used with any hardware parameter such as CPU load, memory consumption, network load and disk load or any server level parameter such as response time. When P$_{C1}$ or P$_{C2}$ controller is used with hardware measured parameter, e.g., CPU load, $C(t)$ becomes the total CPU capacity needed by the system to handle the CPU load per unit time. ΔD is the change in the load. $\widehat{C}$ becomes the average periodical measurement of the CPU load and $\overline{n}$ the average measurement of the CPU load over time. The definition of the two controllers remains the same.

Both the UR-DP$_{C1}$ and UR-DP$_{C2}$ engines can be integrated in the model proposed by Lim et al. [21] to control a storage cloud. In storage clouds, adding resources does not have an instantaneous effect on the performance since data must be copied to the new allocated resources before the effect of the control action takes place. For such a scenario, P$_{C1}$ and P$_{C2}$ are very well suited since they predict the envelope of the demand. The engines can also replace the elasticity controllers designed by Urgaonkar et al. [9] or Iqbala et al. [10] for a multi-tier service deployed in the cloud.

IV. Related Work

The problem of dynamic provisioning of resources in computing clusters has been studied for the past decade. Cloud elasticity can be viewed as a generalization of that problem. Our model is similar to the model introduced in [22]. In that work, the authors tried to estimate the availability of a machine in a distributed storage system in order to replicate its data.

Toffetti et al. [23] use Kriging surrogate models to approximate the performance profile of virtualized, multi-tier Web applications. The performance profile is specific to an

application. The Kriging surrogate model needs offline training. A change in the workload dynamics results in a change in the service model. Adaptivity of the service model of an application is vital to cope with the changing load dynamics in todays Internet [3].

Lim et al. [21] design an integral elasticity controller with proportional thresholding. They use a dynamic target range for the set point. The integral gain is calculated offline making this technique suitable for a system where no sudden changes to the system dynamics occur as the robustness of an integral controller is affected by changing the system dynamics [24].

Urgaonkar et al. [9] propose a hybrid control mechanism that incorporates both a proactive controller and a reactive controller. The proactive controller maintains the history of the session arrival rate seen. Provisioning is done before each hour based on the worst load seen in the past. No short term predictions can be done. The reactive controller acts on short time scales to increase the resources allocated to a service in case the predicted value is less than the actual load that arrived. No scale down mechanism is available.

In [25], the resource-provisioning problem is posed as one of sequential optimization under uncertainty and solved using limited look-ahead control. Although the solution shows very good theoretical results, it exhibits an exponential increase in computation time as the number of servers and VMs increase. It takes 30 minutes to compute the required elasticity decision for a system with 60 VMs and 15 physical servers. Similarly, Nilabja et al. use limited lookahead control along with model predictive control for automating elasticity decisions. Improving the scalability of their approach is left as a future direction to extend their work.

Chacin and Navaro [26] propose an elastic utility driven overlay network that dynamically allocate instances to a service using an overlay network. The instances of each services construct an overlay while the non-allocated instances construct another overlay. The overlays change the number of instances allocated to a service based on a combination of an application provided utility function to express the service's QoS, with an epidemic protocol for state information dissemination and simple local decisions on each instance.

There are also some studies discussing vertical elasticity [27]. Jung et al. [4] design a middleware for generating cost sensitive adaptation actions such as elasticity and migration actions. Vertical elasticity is enforced using adaptation action in fixed steps predefined in the system. To allocate more VMs to an application a migration action is issued from a pool of dormant VMs to the pool of the VMs of the target host followed by an increase adaptation action that allocates resources on the migrated VM for the target application. These decisions are made using a combination of predictive models and graph search techniques reducing scalability. The authors leave the scalability of their approach for future work.

V. Conclusions and Future Work

In this paper, we consider the problem of autonomic dynamic provisioning for a cloud infrastructure. We introduce two adaptive hybrid controllers P_{C1} and P_{C2}, that use both reactive and proactive control to dynamically change the number of VMs allocated to a service running in the cloud based on the current and the predicted future demand. Our controllers detect the workload envelope and hence do not deallocate resources prematurely. We discuss the different ways of designing a hybrid elasticity controller that incorporates both reactive and proactive components. Our simulation results show that using a reactive controller for scale up and one of our proactive controllers for scale down improves the SLA violations rate two to ten times compared to a totally reactive elasticity engine. We compare our controllers to a regression based elasticity controller using a different workload and demonstrate that our engines over-allocate between 32% and 15% less resources compared to a regression based engine. The regression based elasticity engine SLA violation rate is 1.48 to 2.1 times the SLA violation rate for our engines. We also investigate the effect of the workload size on the performance of our controllers. For increasing loads, our simulation results show a sublinear increase in the number of SLAs violated using our controllers compared to a linear increase in the number of SLAs violations for a reactive controller. In the future, we plan to integrate vertical elasticity control in our elasticity engine and modify the controllers to consider the delay required for VM start up and shut down.

VI. Acknowledgments

This work is supported by the OPTIMIS project (http://www.optimis-project.eu/) and the Swedish government's strategic research project eSSENCE. It has been partly funded by the European Commissions IST activity of the 7th Framework Program under contract number 257115 . This research was conducted using the resources of High Performance Computing Center North (http://www.hpc2n.umu.se/).

References

[1] P. Mell and T. Grance, "The NIST definition of cloud computing," *National Institute of Standards and Technology*, vol. 53, no. 6, 2009.

[2] D. Kossmann and T. Kraska, "Data management in the cloud: Promises, state-of-the-art, and open questions," *Datenbank-Spektrum*, vol. 10, pp. 121–129, 2010, 10.1007/s13222-010-0033-3. [Online]. Available: http://dx.doi.org/10.1007/s13222-010-0033-3

[3] I. Ari, B. Hong, E. Miller, S. Brandt, and D. Long, "Managing flash crowds on the Internet," 2003.

[4] G. Jung, K. R. Joshi, M. A. Hiltunen, R. D. Schlichting, and C. Pu, "A cost-sensitive adaptation engine for server consolidation of multitier applications," in *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*, ser. Middleware '09. Springer-Verlag New York, Inc., 2009, pp. 9:1–9:20.

[5] K. Ogata, *Modern control engineering*. Prentice Hall, 2009.

[6] H. Li and T. Yang, "Queues with a variable number of servers," *European Journal of Operational Research*, vol. 124, no. 3, pp. 615 – 628, 2000.

[7] P. Bodık, "Automating datacenter operations using machine learning," Ph.D. dissertation, University of California, 2010.

[8] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM '09. New York, NY, USA: ACM, 2009, pp. 51–62.

[9] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier Internet applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 3, no. 1, p. 1, 2008.

[10] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871 – 879, 2011.

[11] M. Arlitt and T. Jin. (1998, August) "1998 world cup web site access logs". [Online]. Available: http://www.acm.org/sigcomm/ITA/

[12] G. Urdaneta, G. Pierre, and M. van Steen, "Wikipedia workload analysis for decentralized hosting," *Elsevier Computer Networks*, vol. 53, no. 11, pp. 1830–1845, July 2009, http://www.globule.org/publi/WWADH_comnet2009.html.

[13] (2011, July) Internet growth statistics. [Online]. Available: http://www.internetworldstats.com/emarketing.htm

[14] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. Culler, and R. Katz, "Napsac: design and implementation of a power-proportional web cluster," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 1, pp. 102–108, 2011.

[15] P. Dhawan. (2001, October) Performance comparison: Exposing existing code as a web service. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms978401.aspx

[16] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *Grid Computing, 2009 10th IEEE/ACM International Conference on*. IEEE, 2009, pp. 50–57.

[17] W. M. Hartmann, *Signals, sound, and sensation*. Amer Inst of Physics, 1997.

[18] (2009, June) Rackspace hosting: Service level agreement. [Online]. Available: http://www.rackspace.com/cloud/legal/sla/

[19] (2007, October) Amazon S3 service level agreement. [Online]. Available: http://aws.amazon.com/s3-sla/

[20] A. J. Ferrer, F. Hernndez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forg, T. Sharif, and C. Sheridan, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66 – 77, 2012.

[21] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceeding of the 7th international conference on Autonomic computing*, ser. ICAC '10. New York, NY, USA: ACM, 2010, pp. 1–10.

[22] A. Duminuco, E. Biersack, and T. En-Najjary, "Proactive replication in distributed storage systems using machine availability estimation," in *Proceedings of the 2007 ACM CoNEXT conference*, ser. CoNEXT '07. New York, NY, USA: ACM, 2007, pp. 27:1–27:12.

[23] G. Toffetti, A. Gambi, M. Pezzé, and C. Pautasso, "Engineering autonomic controllers for virtualized web applications," *Web Engineering*, pp. 66–80, 2010.

[24] M. Morari, "Robust stability of systems with integral control," *Automatic Control, IEEE Transactions on*, vol. 30, no. 6, pp. 574–577, 1985.

[25] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang, "Power and performance management of virtualized computing environments via lookahead control," *Cluster Computing*, vol. 12, no. 1, pp. 1–15, 2009.

[26] P. Chacin and L. Navarro, "Utility driven elastic services," in *Distributed Applications and Interoperable Systems*. Springer, 2011, pp. 122–135.

[27] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th international conference on Autonomic computing*, ser. ICAC '09. New York, NY, USA: ACM, 2009, pp. 117–126.

II

Paper II

Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control*

Ahmed Ali-Eldin [1], Maria Kihl[2], Johan Tordsson[1], and Erik Elmroth[1]

[1] Dept. Computing Science, Umeå University
{ahmeda,tordsson,elmroth}@cs.umu.se, http://www.gird.se
[2]Dept. of Electrical and Information Technology, Lund University,
Maria.Kihl@eit.lth.se

Abstract: Elasticity is the ability of a cloud infrastructure to dynamically change the amount of resources allocated to a running service as load changes. We build an autonomous elasticity controller that changes the number of virtual machines allocated to a service based on both monitored load changes and predictions of future load. The cloud infrastructure is modeled as a G/G/N queue. This model is used to construct a hybrid reactive-adaptive controller that quickly reacts to sudden load changes, prevents premature release of resources, takes into account the heterogeneity of the workload, and avoids oscillations. Using simulations with Web and cluster workload traces, we show that our proposed controller lowers the number of delayed requests by a factor of 70 for the Web traces and 3 for the cluster traces when compared to a reactive controller. Our controller also decreases the average number of queued requests by a factor of 3 for both traces, and reduces oscillations by a factor of 7 for the Web traces and 3 for the cluster traces. This comes at the expense of between 20% and 30% over-provisioning, as compared to a few percent for the reactive controller.

Efficient Provisioning of Bursty Scientific Workloads on the Cloud Using Adaptive Elasticity Control

Ahmed Ali-Eldin
Dept. of Computing Science
Umeå University, Sweden
ahmeda@cs.umu.se

Maria Kihl
Dept. of Electrical and Information Technology,
Lund University
Maria.Kihl@eit.lth.se

Johan Tordsson
Dept. of Computing Science
Umeå University, Sweden
tordsson@cs.umu.se

Erik Elmroth
Dept. of Computing Science
Umeå University, Sweden
elmroth@cs.umu.se

ABSTRACT

Elasticity is the ability of a cloud infrastructure to dynamically change the amount of resources allocated to a running service as load changes. We build an autonomous elasticity controller that changes the number of virtual machines allocated to a service based on both monitored load changes and predictions of future load. The cloud infrastructure is modeled as a $G/G/N$ queue. This model is used to construct a hybrid reactive-adaptive controller that quickly reacts to sudden load changes, prevents premature release of resources, takes into account the heterogeneity of the workload, and avoids oscillations. Using simulations with Web and cluster workload traces, we show that our proposed controller lowers the number of delayed requests by a factor of 70 for the Web traces and 3 for the cluster traces when compared to a reactive controller. Our controller also decreases the average number of queued requests by a factor of 3 for both traces, and reduces oscillations by a factor of 7 for the Web traces and 3 for the cluster traces. This comes at the expense of between 20% and 30% over-provisioning, as compared to a few percent for the reactive controller.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems; C.4 [**Performance of systems**]

General Terms

Algorithms, Performance, Reliability

Keywords

Cloud computing, Elasticity, Proportional Control

ScienceCloud'12, June 18, 2012, Delft, The Netherlands.

1. INTRODUCTION

Elasticity of the cloud infrastructure is the ability of the infrastructure to allocate resources to a service based on the running load as fast as possible. An *elasticity controller* aims to allocate enough resources to a running service while at the same time avoiding costly over-provisioning. The problem for an elasticity controller is thus to decide when, and how much, to scale up or down. Scaling can be done either horizontally, by increasing or decreasing the number of Virtual Machines (VMs) allocated, or vertically, by changing the hardware configuration for CPU, memory, etc. of already running VMs. The resources allocated to a service can vary between a handful of VMs to tens of thousands of VMs depending on the load requirements. Most Infrastructure as a Service (IaaS) providers does not host a single service but rather quite a few scalable services and applications. Given the scale of the current and future cloud datacenters and services, these are impossible to manage manually, making autonomic management a key issue for clouds.

Recently, the scientific computing community started discussing the potential use of cloud computing infrastructures to run scientific experiments such as medical NLP processing [6] and workflows for astronomical data released by the Kepler project [25]. Most of the applications are embarrassingly parallel [11]. There are some limitations to the wide adoption of the cloud paradigm for scientific computing as identified by Truong et al. [23] such as the lack of cost evaluation tools, cluster machine images and, as addressed in this paper, autonomic elasticity control.

There are many approaches to solve the elasticity problem [5, 7, 10, 17, 18, 20, 24, 27, 28], each with its own strengths and weaknesses. Desired properties of an elasticity controller include the following:

- Fast: The time required by the controller to make a decision is a key factor for successful control, for example, limited look-ahead control is shown to have superior accuracy but requires 30 minutes to control 60 VMs on 15 physical servers [14].

- Scalable: The controller should be scalable with respect to the number of VMs allocated to a service and with respect to the time of running the algorithm. There are many techniques that can be used for esti-

mation of the load and elasticity control which are not scalable with either time or scale e.g., regression based control is not scalable with respect to the algorithm execution time [1].

- Adaptive: Scientific workloads and Internet traffic are very dynamic in nature [2, 15]. Elasticity controllers should have a proactive component that predicts the future load to be able to provision resources a priori. Most prediction techniques such as neural networks build a model for the load in order to predict the future. Another desired property of an adaptive controller is the ability to change the model whenever the load dynamics change.
- Robust and reliable: The changing load dynamics might lead to a change in the controller behavior [9, 19]. A controller should be robust against changing load dynamics. A robust controller should prevent oscillations in resource allocation i.e., the controller should not release resources prematurely. A reactive controller (step controller) is a controller that only allocates new VMs to a service when the load increases and deallocates the VMs once the load decreases beyond a certain level. This type of controller thus reduces the number of VMs provisioned and minimizes the provisioning costs, at the expense of oscillations.

Our previous work [1] studies different ways to combine reactive and proactive control approaches for horizontal elasticity. The two simple hybrid controllers proposed combine reactive scaling up with proactive scale-down. These controllers act on the monitored and predicted service load, but ignore multiple important aspects of infrastructure performance and service workload. In this paper, our previous work is extended by an enhanced system model and controller design. The new controller takes into account the VM startup time, workload heterogeneity, and the changing request service rate of a VM. It thus controls the allocated capacity instead of only the service load. The controller design is further improved by adding a buffer to the controller to store any delayed requests for future processing. This buffer model characterizes many scientific workloads where jobs are usually queued for future processing. The proposed controller can be used by both the cloud service provider and the cloud user to reduce the cost of operations and the cost of running a service or an experiment in the cloud. The controller can be used also to control the elasticity of a privately run cloud or cluster.

The performance of the controller is tested using two sets of traces, a Web workload from the FIFA world cup [3] and a recently published workload from a Google cluster composed of around 11 thousand machines [26]. The Web trace is selected as it is a well known and rather bursty workload and thus challenging for an elasticity controller. The cluster traces, consisting mostly of MapReduce jobs, are chosen to evaluate the behavior of our approach on traces more similar to scientific workloads.

The rest of this paper is organized as follows. Section 2 describes the system model and the design of the proposed controller. In Section 3, the simulation framework and the experiments are described and the results are discussed. Section 4 discusses some of the different approaches available in the literature for building elasticity controllers. Section 5 contains the conclusions.

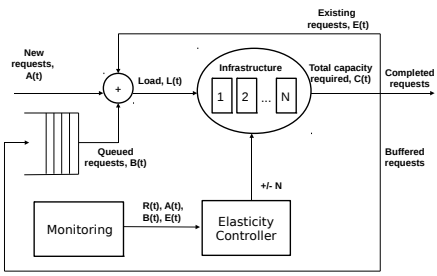


Figure 1: Queueing model and elasticity control for a cloud service.

2. CONTROLLER DESIGN

2.1 System model

In this work, the cloud infrastructure is modeled as a closed loop control system and queueing models are used to design a feedback elasticity controller. The cloud infrastructure is modeled as a $G/G/N$ stable queue in which the number of servers N required is variable [16] as shown in Figure 1. This is a generalization of the work by Khazaei et al. [13] where a cloud is modeled as an $M/G/m$ queue with a constant number of servers, m. We assume that the system serves generic requests that can be anything from a Web query to Pubmed [12] to a job in a workflow to process astronomical data [25].

The number of VMs allocated to a service at any time unit, N, changes according to the controller output. When the load increases, VMs are added and when it decreases VMs are removed. We assume that it takes one time unit for a VM to boot and get intialized. In practice, it also takes time to shut-down a VM, but for most applications, no more requests are sent to a VM after a shutdown command is issued. It is thus assumed that the effect of VM shut down on capacity is instantaneous.

In our model, requests not served are buffered and delayed as shown in Figure 1. We make no assumptions about a finite buffer size, but the designed controller uses the number of buffer requests as one criteria for adding VMs. The buffer length is also used as a performance metric in the evaluation section. A buffered request is delayed and thus the larger the number of buffered requests, the slower the request response time. Assuming the queue is stable, the average service rate over time is equal to the average arrival rate over time. Whenever the system is at risk of instability due to increase in the demand, the elasticity controller increases N to enforce system stability. The elasticity control problem can be stated as follows: the elasticity controller should add or remove VMs to ensure system stability, i.e., over a long period of time, the number of serviced requests (the service capacity) is equal to the total number of received requests received with an error tolerance (number of

Table 1: Overview of used notation.

Variable	Description
N	Number of VMs deployed
$L(t)$	Total service load at time t
$R(t)$	Total service capacity available at time t
$C(t)$	Service capacity required at time t
$A(t)$	Arriving (new) requests at time t
$D(t)$	Increase/decrease in required capacity at time t
$B(t)$	Size of buffer at time t
$E(t)$	Amount of already processing requests at time t
K	Number of queued requests before starting a new VM
r	Number of time units needed to start all buffered requests
T_d	Estimation interval (time between two estimations)
$\overline{L_{T_d}}$	Average load over the last estimation interval
$\overline{L_t}$	Average load over all time
$\tilde{D}$	Predicted value for request change rates over next T_d time units
P	Estimated ratio between $\tilde{D}$ and average load
M_{Avg}	The average of the median request service rates per unit time over the the T_d

buffered requests). This should be achieved irrespective of the change in the request arrival rate and while maintaining the number of VMs to a minimum.

2.2 Estimating future usage

The optimal total service capacity, $C(t)$, required for time t is:

$$C(t) = C(t-1) + D(t), \tag{1}$$

where $C(t-1)$ is the capacity required in the last time step and $D(t)$ is the increase or decrease in capacity needed in order to meet SLAs while maintaining the number of VMs to a minimum. The controller is activated each T_d time units. When evoked at time t, it estimates the change in workload for the next T_d time units, $D(t+1), D(t+2), \ldots, D(t+T_d)$. VM allocations are adjusted at times $t+1, t+2, \ldots, t+T_d$ according to these predictions, followed by a new prediction for $t+T_d \ldots t+2T_d$. We define $A(t)$ as the arrival rate of new requests to the service. A suitable initial service configuration could be $C(0) = A(0)$.

We define the total workload of the service, $L(t)$, as the sum of the arriving requests, the existing, already processing requests, $E(t)$, and any buffered requests to be served. No assumptions are thus made about the time needed to serve a request, which can vary from seconds to hours. We use $B(t)$ to denote the number of requests in the buffer at time t. If enough VMs are allocated to initialize all buffered requests in the next time unit, these machines may become idle and be released shortly after, causing oscillations in resource allocations. We thus define r, a system parameter specifying over how many time units the currently buffered load should be started. Now, the total workload at time t can be written as:

$$L(t) = A(t) + E(t) + \frac{B(t)}{r}. \tag{2}$$

The capacity change required can be written as

$$D(t) = L(t) - R(t) \tag{3}$$

where $R(t)$ denotes the currently allocated capacity. Assuming that $A(t)$ remains constant for the next time unit, the estimated change in the current service capacity required, $\tilde{D}$ for the future T_d time units can be estimated by

$$\tilde{D} = P\overline{L_{T_d}} \tag{4}$$

where P represents the estimated rate of adding or removing VMs. We define $\overline{L_{T_d}}$ to be the average periodical service load over the past T_d time units,

$$\overline{L_{T_d}} = \frac{\sum_{i=0}^{T_d} L(t-i)}{T_d}. \tag{5}$$

Similarly, we define $\overline{L_t}$, as the average load over all time as follows:

$$\overline{L_t} = \frac{\sum_{i=0}^{t} L(i)}{t}. \tag{6}$$

Now, P represents the estimated ratio of the average change in the load to the average load over the next T_d time units and $\overline{L_{T_d}}$ is the estimated average capacity required to keep the buffer size stable for the next T_d time units. P is positive if there is an increase in total workload (new service requests, buffered requests, and requests that need to be processed longer); negative if this sum decreases (with the buffer empty); and zero if the system is at a steady state and the buffer is empty.

We define P to be the ratio between $D(t)$ and the average system load over time,

$$P = \frac{D(t)}{\overline{L_t}}. \tag{7}$$

This value represents the change in the load with respect to the average capacity. By substituting Equations 7 in Equation 4,

$$\tilde{D} = \frac{\overline{L_{T_d}}}{\overline{L_t}} D(t). \tag{8}$$

This formulation is a proportional controller [21] where $D(t)$ is the error signal and $\overline{L_{T_d}}/\overline{L_t}$, the normalized service capacity, is the gain parameter of the controller. By substituting Equation 5 in Equation 8, we obtain

$$\tilde{D} = \frac{\sum_{i=0}^{T_d} L(t-i)}{\overline{L_t}} \frac{D(t)}{T_d}. \tag{9}$$

If T_d is optimal, i.e., estimations occur when the rate of change of the load changes, then $D(t)/T_d$ is the slope of the changing load multiplied by the ratio between the instantaneous load and the overtime average load.

2.3 Determining suitable estimation intervals

The interval between two estimations, T_d, is a crucial parameter affecting the controller performance. It is used to calculate P and $\tilde{D}$ and T_d also controls the reactivity of the controller. If T_d is set to one, the controller performs predictions every time unit. At the other extreme, if T_d is set to ∞, the controller performs no predictions at all. As the workloads observed in datacenters are dynamic [2], setting an adaptive value for T_d that changes according to the load dynamics is important.

We define the maximum number of buffered requests, K, as the tolerance level of a service i.e., the maximum number of requests queued before making a new estimation or adding a new VM, thus:

$$T_d = \begin{cases} K/|\tilde{D}| & \text{if K>0 and } |\tilde{D}| \neq 0 \\ 1 & \text{if K=0 or } \tilde{D} = 0 \end{cases} \quad (10)$$

The value of K can be used to model SLAs with availability guarantees. A low value for K provides quicker reaction to load changes, but will also result in oscillations as resources will be provisioned and released based on the last few time units only. Conversely, K is large, the system reacts slowly to changing load dynamics. Similarly, r affects the rate with which buffered requests should be started, and thus impose similar tradeoffs between oscillations and quickly reacting to load increases.

2.4 Hybrid elasticity control

The main goal of an elasticity controller is to allocate enough resources to enforce the SLAs while decreasing total resource usage. It is very hard to anticipate whether an observed increase in load will continue to rise to a large peak [2] as there are no perfect estimators or controllers. Using pure estimation for scale-up decisions is dangerous as it can lead to system instability and oscillations if the load dynamics change suddenly while the controller model is based on the previous load.

Data: r, K
Result: Perform resource (de)allocation to keep the system stable

1 $Proactive\_Aggregator \leftarrow 0$;
2 $T_d \leftarrow 1$;
3 **for** *each time step* t **do**
4 Update $R(t), A(t), B(t)$, and $E(t)$ from monitoring data;
5 Calculate $D(t)$ using Equation 3;
6 **if** *Time from last estimation* $\geq T_d$ **then**
7 Calculate $\overline{L_{T_d}}$ from Equation 5;
8 Calculate $\overline{L_t}$ from Equation 6;
9 Calculate P from Equation 7;
10 Calculate $\tilde{D}$ from Equation 8;
11 Update M_{Avg};
12 Calculate T_d from Equation 10;
13 $N_{Reactive} \leftarrow \lceil D(t)/M_{Avg} \rceil$;
14 $Proactive\_Aggregator += \tilde{D}/M_{Avg}$;
15 $N_{Proactive} \leftarrow \lfloor Proactive\_Aggregator \rfloor$;
16 $Proactive\_Aggregator -= N_{Proactive}$;
17 **if** $N_{Reactive} > K$ **then**
18 **if** $N_{Proactive} > 0$ **then**
19 Deploy $N_{Proactive} + N_{Reactive}$ servers
20 **else**
21 Deploy $N_{Reactive}$ servers
22 **else**
23 (Un)deploy $N_{Proactive}$ servers

Algorithm 1: Hybrid elasticity controller with both proactive and reactive components.

Our design is a hybrid controller where a reactive component is coupled with the proposed proactive component for scale up and a proactive only component for scale down. The details of the implementation are given in Algorithm 1. The controller starts by receiving monitoring data from the monitoring subsystem in Line 4. If T_d time units passed since the last estimation, the system model is updated by reestimating $\tilde{D}$ and T_d as shown from Line 6 to Line 12. The unit of $\tilde{D}$ is requests per time unit.

The actual calculation of the number of VMs to be added or removed by the different controllers is done between lines 13 and 16. In some applications, $\tilde{D}$ is divided by M_{Avg} in Line 14 to find the number of servers required. The rate of the proactive controller can be steered by multiplying $\tilde{D}$ by a factor, e.g., to adding double the estimated VMs for some critical applications. The reactive controller is coupled with the proactive controller to reach a unified decision as shown from Line 17 to Line 22. For scale up decisions, when the decisions of both the reactive component and the proactive component are to scale up, the decisions are added. For example, if the reactive component decides that two more VMs are required while the proactive component decides that three VMs are needed, five VMs are added. The reactive component is reacting for the current load while the proactive component is estimating the future load based on the past load. If the reactive component decides that a scale up is needed while the proactive decides that a scale down is needed then the decision of the reactive component alone is performed because the reactive component's decision is based on the current load while the proactive component's decision may be based on a skewed model that needs to be changed.

For the reactive and proactive components to calculate the number of VMs required for a given load, the controller needs to know the service capacity of a VM i.e., the number of requests serviced per VM every time unit. This number is variable as the performance of a VM is not constant.

In addition, there are different request types and each request takes different time to service. As a solution, we calculate M_{Avg}, the average of the median request service rates per VM per unit time over the past estimation period T_d. M_{Avg} is used by the reactive and proactive components to calculate the number of VMs required per unit time to service all the requests while meeting the different SLAs. The median is chosen as it is a simple and efficient statistical measure which is robust to outliers and skewed distributions. No assumptions are made about the service rate distribution for a VM. M_{Avg} is a configurable parameter which can be changed based on deployment requirements.

3. EXPERIMENTAL EVALUATION

To validate the controller, a three-phase discrete-event simulator [4] was built using python that models a service deployed in the cloud. Two different workloads are used for the evaluation, the complete traces from the FIFA 1998 world cup [3] and a set of Google cluster traces [26].

In all evaluations, the controller time step, i.e., the time it takes to start a VM is selected to be 1 minute, which is a reasonable assumption [22]. The effects of the controller's decision to increase the resources provisioned does thus not appear until after one minute has elapsed and the new VMs are running. The granularity of the monitoring data used by the controller is also 1 minute.

The controller's performance is compared to a completely reactive controller similar to the one proposed by Chieu et al. [8]. The design of the reactive controller is shown in

Data: r, K
Result: Perform resource (de)allocation to keep the system stable

for *each time step* t **do**
  Update $R(t), A(t), B(t)$, and $E(t)$ from monitoring data;
  Calculate M_{Avg};
  Calculate $D(t)$ using Equation 3;
  $N_{Reactive} \leftarrow \lceil D(t)/M_{Avg} \rceil$;
  if $N_{Reactive} > 0$ *and* $D(t) > K$ **then**
    Deploy $N_{Reactive}$ servers
  if $N_{Reactive} < -2$ **then**
    Undeploy $N_{Reactive}$ servers

Algorithm 2: Reactive elasticity controller.

Algorithm 2. The calculation of the median service rate is done every minute as this is the time between two load estimations. In order to reduce oscillations in the reactive controller, scale down is not done until the capacity change $D(t)$ is less than the current provisioned capacity $C(t)$ by $2M_{Avg}$, i.e., scale down is only done when there are more than two extra VMs provisioned. In the experiments, we refer to our controller in Algorithm 1 as C_{Hybrid} and the reactive controller in Algorithm 2 as C_{Reactive}.

With a few exceptions, most of the work available on elasticity control compares performance with static provisioning. However, we chose to compare our controller with a reactive controller to highlight the tradeoffs between over-provisioning, SLA violations, and oscillations.

3.1 Performance Metrics

Different metrics can be used to quantify the controller performance. We define OP to be the number of over-provisioned VMs by the controller per time unit aggregated over the whole trace. $\overline{OP}$ is the average number of over-provisioned VMs by the controller per minute. Similarly, UP and $\overline{UP}$ are the aggregate and the average number of under-provisioned VMs by the controller per minute. We also define V, the average number of servers required to service the buffered load per minute,

$$V = \Sigma \frac{\text{Buffered Load}}{\text{Median Service rate of a VM}}. \quad (11)$$

V represents the way the buffers get loaded. It does not represent the optimal number of servers required to service the load but rather represents average required number of VMs due to the queue build up. We use $\overline{N}$ to denote the average number of VMs deployed over time.

3.2 Web workload performance evaluation

The Web workload contains 1.3 billion Web requests recorded at servers for the 1998 FIFA world cup in the period between April 30, 1998 and July 26, 1998. The aggregate number of requests per second were calculated from these traces. In the simulation, the requests are grouped by time of arrival. The focus is not on individual requests but rather on the macro-system performance. For the experiments, the average service rate of a VM is drawn from a Poisson distribution with an average equal to λ requests per second. It is assumed that the time required to process one request is 1 second. The tolerance level K is chosen to be 5, i.e., 5 requests may be buffered before the controller reacts.

Assuming perfect load balancing, the performance of the controller is the only factor affecting performance in the simulation. We set r to 60 seconds, i.e., queued requests are emptied over one minute.

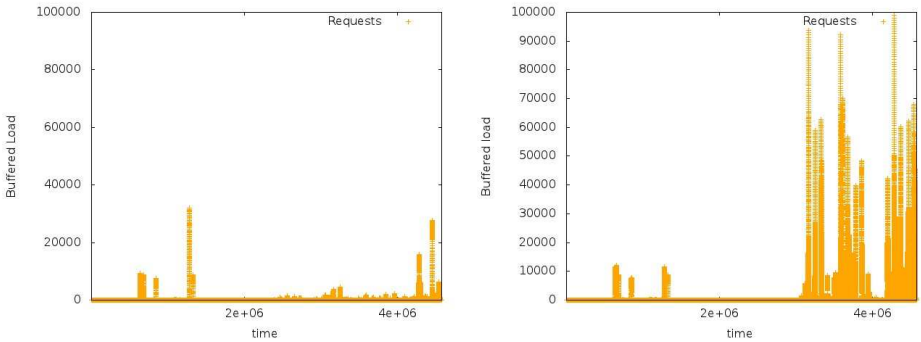
The controller is configured for the worst case scenario by using the maximum load recorded during the past minute as the input request arrival rate to the controller. This assumption can be relaxed by monitoring the average or median load for the past minute instead. Using the median or the average will result in provisioning less resources for most workloads.

As the Web traces are quite old, we have multiplied the number of requests by a factor F in order to stress test the controller performance under different load dynamics. For different experiments, λ is also changed. Table 2 shows the performance of the two controllers when F takes the values of 1, 10, 20, 30, and 40 while λ takes the values of 100, 200, 300, and 400. Due to the size of the trace, the aggregate metrics UP and OP are quite large.

For the over-provisioning metrics, OP and $\overline{OP}$, it is clear that C_{Hybrid} has a higher over-provisioning rate compared to C_{Reactive}. This is intuitive because C_{Hybrid} provisions resources ahead in time to be used in the future and delays the release of resources in order to decrease oscillations. When λ changes such that the ratio between λ and F is constant at 10, OP and $\overline{OP}$ are reduced for both controllers compared to when only F increases and the rate of increase of both values is lower. Notably, these values are quite small if we compare them to static provisioning. If capacity would be statically provisioned for the workload, 42 servers would be needed when $F = 1$ and $\lambda = 100$, whereas using the proactive controller or even the reactive controller, the average number of provisioned servers is around 3, reducing resource use by around 92.5% compared to static provisioning but at the cost of an increase in the number of delayed requests.

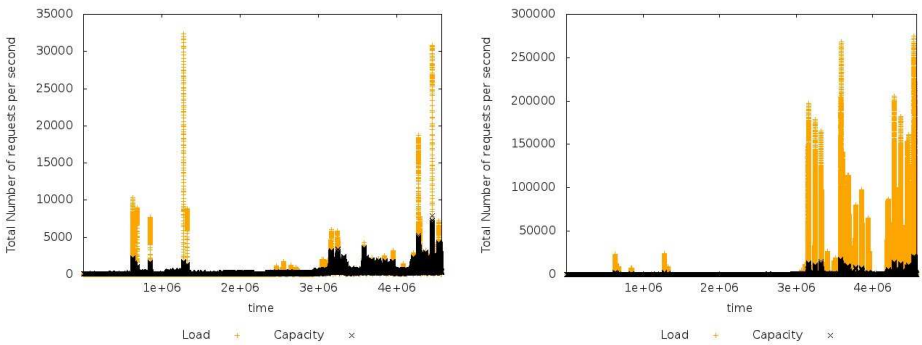
Looking at the aggregate and the average under-provisioning metrics, UP and $\overline{UP}$, C_{Hybrid} is superior to C_{Reactive}. Since C_{Hybrid} scales up and down proactively, it prevents oscillations. It proactively allocates VMs to and does not release resources prematurely, thus decreasing the amount of buffered and delayed requests. In fact, C_{Reactive} shows a very high rate of under-provisioning due to the fact that all delayed requests are buffered. The average number of under-provisioned servers of C_{Reactive} is 70 times that of $C_{\text{Proactive}}$ when $F = 1$ and $\lambda = 100$. Since C_{Reactive} is always lagging the load and releases resources prematurely causing oscillations, the performance of the controller is quite bad. In real life, the buffer is not infinite and thus requests are dropped. In comparison, using the proposed controller, C_{Hybrid}, the request drop rate is quite low. Again, we note that for a ratio between λ and F of 10, under-provisioning (UP and $\overline{UP}$) is reduced for both controllers compared to when only F increases. Actually, for C_{Hybrid}, UP and $\overline{UP}$, are almost constant while for C_{Reactive}, they decrease significantly with the increase of λ and F while having a constant internal ratio. We attribute this to the higher service capacity of the VMs allowing the system to cope with higher system dynamics. For future work, we plan to investigate the effect of the VM capacity on the performance of an elasticity controller with different system dynamics and if the property we have just noted is general for any arrival process.

The V columns in Table 2 show the average number of



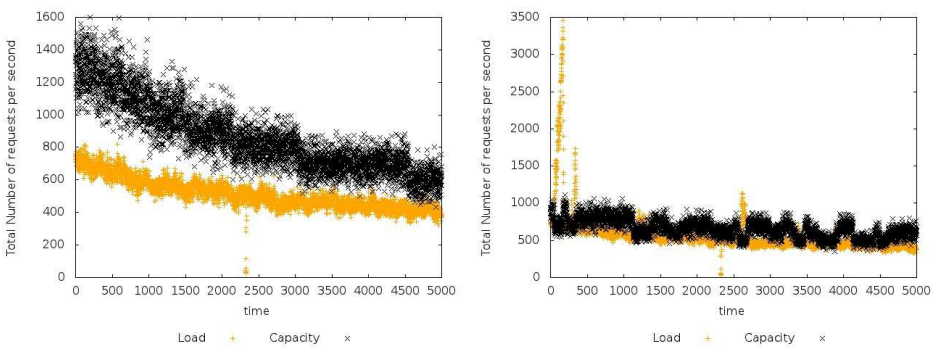
(a) $C_{Proactive}$ (b) C_{Reactive}

Figure 2: Number of buffered requests over time for the Web workload.



(a) C_{Hybrid} (b) C_{Reactive}

Figure 3: Load and the provisioned capacity over time for the Web workload.



(a) C_{Hybrid} (b) C_{Reactive}

Figure 4: Load and the provisioned capacity for 1.5 hours of the Web workload.

Table 2: Web workload performance overview.

C_{Hybrid} results								C_{Reactive} results					
F	λ	OP	$\overline{OP}$	UP	$\overline{UP}$	V	$\overline{N}$	OP	$\overline{OP}$	UP	$\overline{UP}$	V	$\overline{N}$
1	100	41905	0.548	3883	0.05	2.5	3	35600	0.47	267402	3.49	5.98	2.95
10	100	535436	6.99	8315	0.1	19.7	26.09	206697	2.7	8835898	115.45	135.97	23.28
20	100	1075447	14.05	98678	1.29	38.9	51.76	380059	4.966	19611571	256.26	297.29	46.14
30	100	1617452	21.14	148896	1.9	58.1	77.46	555503	7.25	30944637	404.35	466	69.15
40	100	2155408	28.16	197660	2.58	77.3	103.11	732157	9.567	42265699	552.28	634.57	92.14
20	200	654596	8.55	35380	0.46	19.3	27.57	225979	2.95	5187614	67.78	87.63	22.86
30	300	761956	9.96	30951.0	0.4	19.3	28.94	235436	3.07	3783052	49.4	69	22.71
40	400	857608	11.2	30512	0.4	19.3	30.16	241854	3.16	3180898	41.56	61.04	22.7

Table 3: Number of VMs added and removed for the Web workload with $F = 1$ and $\lambda = 100$.

	X_R	X_P
$C_{\text{Proactive}}$	1141	1152
C_{Reactive}	15029	N/A

VMs required to empty the buffer in one minute or the average number of minutes required by a single VM to empty the buffer. This is illustrated by figures 2(a) and 2(b) that show the average buffered requests per VM at any second for C_{Hybrid} and C_{Reactive} respectively when $N = 100$ and $K = 1$. In Figure 2(a) there are three major peaks when the buffered load per VM is above 1000 requests resulting in a relatively small V and $\overline{UP}$ in the table. These three peaks are a result of sudden large load increases. On the other hand, Figure 2(b) shows more peaks with buffered load more than 50000 requests per VM, resulting in a relatively high V and $\overline{UP}$. The average required buffer size for C_{Reactive} per VM in order to service all requests is almost 3 times the buffer size required by $C_{\text{Proactive}}$. Thus, for a limited buffer size, C_{Reactive} drops many more requests than C_{Hybrid}.

Figures 3(a) and 3(b) show the load and the provisioned capacity using both controllers for the full trace when $\lambda = 100$ and $F = 1$. These plots show the macro behavior of the controllers. The total number of buffered requests is the reason for having very high load peaks for C_{Reactive}. The largest spike seen in Figure 3(a) was around the fifth of May at 11:15. For ten minutes, the load suddenly increases tenfold and then starts oscillating causing some instability in $C_{\text{Proactive}}$. Notable, as the buffered load is emptied over r time units, the capacity does not increase with the same rate as the numbered of buffered requests increase.

To study the micro behavior of the controllers, figures 4(a) and 4(b) show the load and controller output for one and half hour from 21:21:12 on 25 June, 1998 to 22:44:31 on the same day. These figures show that $C_{\text{Proactive}}$ tracks the envelope of the load by keeping the provisioned capacity slightly higher than the load while C_{Reactive} oscillates the provisioned capacity with the increase or decrease of the load. Note that as the capacity of a single VM is variable, the capacity in Figure 4(a), which is measured in number of requests, appears to be oscillating. What actually happens is that the number of provisioned VMs drops gradually from 13 to 6 with no oscillations.

Table 4: Properties of the cluster workload.

	execution time	queue time	total time
Median	347.4 s	3.6 s	441.6 s
Average	3961.8 s	220.76 s	4182.5 s
$90th$ percentile	3803 s	3325 s	4409 s

Table 3 shows X_R, the total number of servers added and removed by the reactive component of a controller, and X_P, the total number of servers added and removed by the proactive component, using C_{Hybrid} and C_{Reactive} for a simulation run with $F = 1$ and $\lambda = 100$. The total number of server added or removed by $C_{\text{Proactive}}$ is 2293 servers almost one seventh of the total number of server added or removed by the reactive controller. These results illustrate how the reactive controller increases resource oscillations.

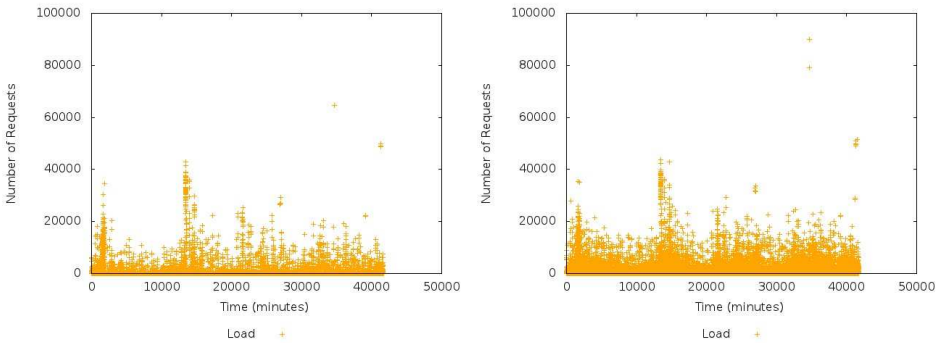
3.3 Cluster workload performance evaluation

Recently, Google published a new sample dataset of resource usage information from a Google production cluster [26]. The traces are from an cluster with 11000 servers. As this cluster is used to run a mixture of MapReduce and other computationally intensive jobs, the traces are representative for scientific workloads.

In this experiment, there is no risk of causing oscillations if $r = 1$ since most of the requests take more than 1 minute to serve. The median job length in the workload is 347.5 seconds or almost 6 minutes. Table 4 summarizes the statistical properties of the tasks in the workload. Due to lack of space we do not comment more on the properties of the workload. K is set to 5 jobs also for this experiment. We define that a job is delayed if it remains in the queue for more than 1 minute. The median number of tasks that can be processed on a single machine in the trace is 170, while the minimum is 120. To be more conservative, we set the number of tasks assigned to a server to 100.

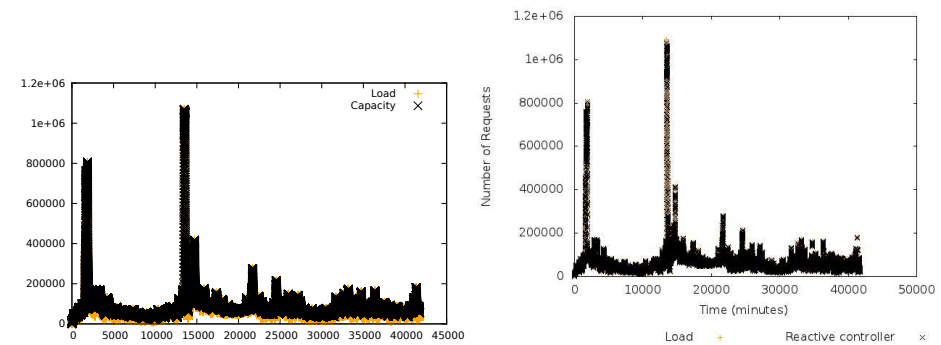
Table 5 shows the performance of the proactive and reactive controllers. The amount of under-provisioning using the C_{Reactive} is almost three times that of $C_{\text{Proactive}}$. This comes at the cost of over-provisioning on average 164 VMs compared to around 1.4 VMs for the reactive controller. However, the amount of over-provisioning is still low, around 25%, as $C_{\text{Proactive}}$ used 847 VMs on average, as compared to 687 VMs for the reactive controller.

While $\overline{OP}$ and $\overline{UP}$ may be crucial for a workload like the Web trace, they are less important for a workload of jobs like the cluster trace where a job can wait in the queue



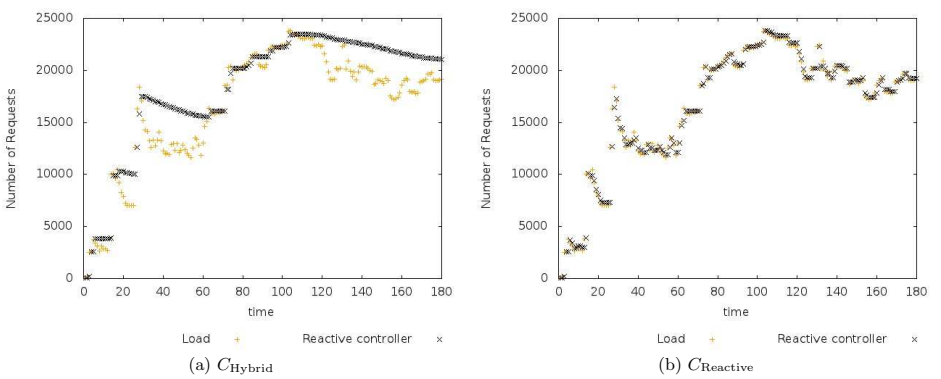
(a) $C_{Proactive}$ (b) C_{Reactive}

Figure 5: Number of buffered requests over time for the cluster workload.



(a) $C_{Proactive}$ (b) C_{Reactive}

Figure 6: Load and the provisioned capacity over time for the cluster workload.



(a) C_{Hybrid} (b) C_{Reactive}

Figure 7: Load and the provisioned capacity for 3 hours of the cluster workload.

Table 5: Cluster workload performance overview.

	$C_{\text{Proactive}}$	C_{Reactive}
$\overline{OP}$	164	1.369
$\overline{UP}$	1.76	5.384
$\overline{N}$	847	686.9
V	3.48	10.22
X_R	75415.0	505289
X_P	78564.0	N/A

for minutes. More importantly for this workload type is V, the average number of buffered tasks. $C_{\text{Proactive}}$ keeps the average number of buffered tasks below K. On the contrary, the reactive controller's average buffer length is double the allowed buffer size K and three times that of the proactive controller. This is illustrated in figures 5(a) and 5(b) that show the number of buffered requests over time.

We also note that in total, the number of VMs added and removed by the reactive controller is 505289 compared to 153979 by the proactive controller. This means that the reactive controller results in more oscillations also for the cluster workload.

Figures 6(a) and 6(b) show the load and provisioned capacity for $C_{\text{Proactive}}$ and C_{Reactive}. The proactive controller tracks the envelope of the workload, i.e., the capacity stays ahead of the load most of the time, whereas the reactive controller always lags the load by at least one time unit. Due to the large number of points plotted, the load appears as if it is completely covered with the capacity. In order to see the micro behavior of the two controllers we plot the load and capacity for both controllers for the first 3 hours of the trace in figures 7(a) and 7(b). The figures show how oscillations are reduced using the proactive controller. For example, for the sudden decreases in load at minutes 15 and 30, C_{Reactive} quickly deallocated VMs followed by reallocations as load increased again. In contrast, $C_{\text{Proactive}}$ kept most of the allocated VMs, causing less oscillations. To summarize the experiments, the workload characteristics and the SLA requirements influence the performance of both controllers considerably. We also note that our elasticity controller is highly scalable with respect to service workload and infrastructure size. In the performed evaluations, the controller required on average a few milliseconds to make a decision.

4. RELATED WORK

Elasticity is an incarnation of the dynamic provisioning problem which has been studied for over a decade [7] from the perspectives of both server provisioning and cloud computing. Different approaches have been proposed to solve the problem in both its old and new incarnations. Some previous research considered only vertical elasticity [17, 27]. while many others considered horizontal elasticity in different contexts [20, 28].

Urgaonkar et al. [24] were among the first to discuss the effect of virtualization on the provisioning problem or what we call horizontal elasticity. They proposed an adaptive controller composed of a predictive and a reactive components. The predictive component acts in the time scale of hours or days. It provisions resources based on the tail distribution of the load. The reactive component acts in the time scale of minutes to handle flash crowds by scaling up the resources provisioned. The model of the predictive controller is tuned according to the under-provisioning of resources seen in the past a few hours. Scale down is not considered.

Gandhi et al. [10] propose a similar controller. The main difference is in the predictive controller design. Their predictive controller identifies patterns in the workload using a workload forecaster which discretizes it into consecutive, disjoint time intervals with a single representative demand value. Workload forecasting is done on the time scale of days i.e., the model of the predictive controller is changed at most once a day. In their approach there is no way to tune the model of the predictive controller and they do not consider scale down of resources.

Malkowski et al. [18] propose a controller for n-tiered applications. They add to the predictive and reactive controller a database of previous system states with good configurations. The elasticity controller starts by looking up if the current state of the system in the database. If the state is found then the configuration corresponding to the state is used. Otherwise, the reactive controller determines the underutilized state or over-utilized state and provisions resources according to the load. In addition, the predictive controller uses Fourier transforms to forecast the future workload for each tier from the past.

A much simpler approach is proposed by Calheiros et al. [5]. They model a cloud provider using basic queueing theory techniques. They assume heterogeneous requests that take constant time to process.

5. CONCLUSION

In this paper, we consider the problem of autonomic elasticity control for cloud infrastructures. The infrastructure is modeled as a $G/G/N$ queue with variable N. The model is used to design an adaptive proportional controller that proactively adapts based on the changes in the load dynamics. The controller takes into account resource heterogeneity, delayed requests, and variable VM service rates. A hybrid controller combines the designed controller with a reactive controller that reacts to sudden increases in the load. The combined controller tries to follow the workload envelope and avoids premature resource deallocation.

Using simulations we compare the proposed controller to a completely reactive controller. Two traces with different characteristics are used, Web traces from the FIFA world cup that are quite bursty in nature with simple requests and cluster traces from Google with jobs as long as 1 hour. Simulation results show that our proposed controller outperforms the reactive controller by decreasing the SLA violation rate by a factor between 70 for the Web workload and 3 for the cluster one. The reactive controller required three times larger buffers compared to our controller. The results also show that the proposed controller reduces resource oscillations by a factor of seven for the Web workload traces and a factor of three for the cluster traces. As a tradeoff, the hybrid controller over-provisions between 20% and 30% resources as compared to a few percent for the reactive one.

6. ACKNOWLEDGMENTS

We would like to thank the reviewers for their constructive comments. Financial support has been provided in part by the European Community's Seventh Framework Programme under grant agreement #257115, the Lund Center for Con-

trol of Complex Engineering Systems, and the Swedish Government's strategic effort eSSENCE.

7. REFERENCES

[1] A. Ali-Eldin, J. Tordsson, and E. Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *NOMS 2012, IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2012. in press.

[2] I. Ari, B. Hong, E. Miller, S. Brandt, and D. Long. Managing flash crowds on the Internet. 2003.

[3] M. Arlitt and T. Jin. "1998 world cup web site access logs", August 1998.

[4] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice-Hall, Upper Saddle River, N.J., fourth edition, 2005.

[5] R. N. Calheiros, R. Ranjan, and R. Buyya. Virtual machine provisioning based on analytical performance and qos in cloud computing environments. In *International Conference on Parallel Processing (ICPP)*, pages 295 –304, sept. 2011.

[6] K. Chard, M. Russell, Y. Lussier, E. Mendonca, and J. Silverstein. Scalability and cost of a cloud-based approach to medical nlp. In *Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on*, pages 1–6. IEEE, 2011.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 103–116. ACM, 2001.

[8] T. Chieu, A. Mohindra, A. Karve, and A. Segal. Dynamic scaling of web applications in a virtualized cloud computing environment. In *e-Business Engineering, 2009. ICEBE '09. IEEE International Conference on*, pages 281 –286, oct. 2009.

[9] A. J. Ferrer, F. HernÃąndez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. ForgÃş, T. Sharif, and C. Sheridan. Optimis: A holistic approach to cloud service provisioning. *Future Generation Computer Systems*, 28(1):66 – 77, 2012.

[10] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah. Minimizing data center sla violations and power consumption via hybrid resource provisioning. In *International Green Computing Conference and Workshops (IGCC)*, pages 1 –8, july 2011.

[11] T. Gunarathne, T. Wu, J. Qiu, and G. Fox. Cloud computing paradigms for pleasingly parallel biomedical applications. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 460–469. ACM, 2010.

[12] J. Herskovic, L. Tanaka, W. Hersh, and E. Bernstam. A day in the life of PubMed: analysis of a typical day's query log. *Journal of the American Medical Informatics Association*, 14(2):212, 2007.

[13] H. Khazaei, J. Misic, and V. Misic. Modelling of cloud computing centers using m/g/m queues. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 87 –92, june 2011.

[14] D. Kusic, J. Kephart, J. Hanson, N. Kandasamy, and G. Jiang. Power and performance management of virtualized computing environments via lookahead control. *Cluster Computing*, 12(1):1–15, 2009.

[15] H. Li. Realistic workload modeling and its performance impacts in large-scale escience grids. *IEEE Transactions on Parallel and Distributed Systems*, 21(4):480–493, 2010.

[16] H. Li and T. Yang. Queues with a variable number of servers. *European Journal of Operational Research*, 124(3):615–628, 2000.

[17] X. Liu, X. Zhu, P. Padala, Z. Wang, and S. Singhal. Optimal multivariate control for differentiated services on a shared hosting platform. In *46th IEEE Conference on Decision and Control*, pages 3792–3799. IEEE, 2007.

[18] S. J. Malkowski, M. Hedwig, J. Li, C. Pu, and D. Neumann. Automated control for elastic n-tier workloads based on empirical modeling. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 131–140. ACM, 2011.

[19] M. Morari. Robust stability of systems with integral control. *IEEE Transactions on Automatic Control*, 30(6):574–577, 1985.

[20] D. Narayanan, A. Donnelly, E. Thereska, S. Elnikety, and A. Rowstron. Everest: Scaling down peak loads through i/o off-loading. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 15–28. USENIX Association, 2008.

[21] K. Ogata. *Modern control engineering*. Prentice Hall PTR, 2001.

[22] P. Svärd, B. Hudzia, J. Tordsson, and E. Elmroth. Evaluation of delta compression techniques for efficient live migration of large virtual machines. *SIGPLAN Not.*, 46:111–120, Mar. 2011.

[23] H. Truong and S. Dustdar. Cloud computing for small research groups in computational science and engineering: current status and outlook. *Computing*, pages 1–17, 2011.

[24] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Trans. Auton. Adapt. Syst.*, 3:1:1–1:39, March 2008.

[25] J. Vöckler, G. Juve, E. Deelman, M. Rynge, and G. Berriman. Experiences using cloud computing for a scientific workflow application. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, pages 15–24. ACM, 2011.

[26] J. Wilkes. More google cluster data, November 2011.

[27] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif. On the use of fuzzy modeling in virtualized data center management. *International Conference on Autonomic Computing*, page 25, 2007.

[28] Q. Zhu and G. Agrawal. Resource provisioning with budget constraints for adaptive applications in cloud environments. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pages 304–307. ACM, 2010.

III

Paper III

Workload Classification for Efficient Cloud Infrastructure Elasticity Control

Ahmed Ali-Eldin[1], Johan Tordsson[1],
Erik Elmroth[1], and Maria Kihl[2]

[1]Department of Computing Science, Umeå University, Sweden
{ahmeda, tordsson, elmroth}@cs.umu.se
[2]Department of Electrical and Information Technology,
Lund University, Sweden
Maria.Kihl@eit.lth.se

Abstract: Elasticity algorithms for cloud infrastructures dynamically change the amount of resources allocated to a running service according to the current and predicted future load. Since there is no perfect predictor, and since different applications' workloads have different characteristics, no single elasticity algorithm is suitable for accurate future predictions for all workloads. In this work, we introduce WAC, a Workload Analysis and Classification tool that analyzes workloads and assigns them to the most suitable elasticity controllers based on the workloads' characteristics and a set of business level objectives. WAC has two main components, an analyzer and a classifier. The analyzer analyzes workloads to extract the workloads' autocorrelations and sample entropies which measure the periodicity and the burstiness of the workloads respectively. Periodicity and burstiness are the two main workload characteristics that effect a controller's performance. These two features are used with the business level objectives by the classifier to assign workloads to elasticity controllers. We start by analyzing 14 real workloads available from different applications. In addition, a set of 55 workloads is generated to test WAC on more workload configurations. We implement four state of the art elasticity controllers to which the classifier assigns workloads. We use a K nearest neighbors classifier and experiment with different workload combinations of training and test sets. Our experiments show that, when the classifier is tuned carefully, WAC correctly assigns between 92% and 98.3% of the workloads to the most suitable elasticity controller.

Workload Classification for Efficient Cloud Infrastructure Elasticity Control

Ahmed Ali-Eldin[1], Johan Tordsson[1],
Erik Elmroth[1], and Maria Kihl[2]

[1]Department of Computing Science, Umeå University, Sweden
{ahmeda, tordsson, elmroth}@cs.umu.se
[2]Department of Electrical and Information Technology,
Lund University, Sweden
Maria.Kihl@eit.lth.se

May 29, 2013

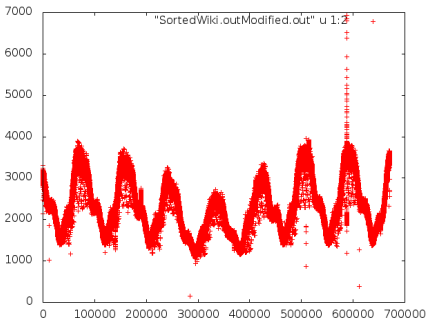
Abstract

Elasticity algorithms for cloud infrastructures dynamically change the amount of resources allocated to a running service according to the current and predicted future load. Since there is no perfect predictor, and since different applications' workloads have different characteristics, no single elasticity algorithm is suitable for accurate future predictions for all workloads. In this work, we introduce WAC, a Workload Analysis and Classification tool that analyzes workloads and assigns them to the most suitable elasticity controllers based on the workloads' characteristics and a set of business level objectives.

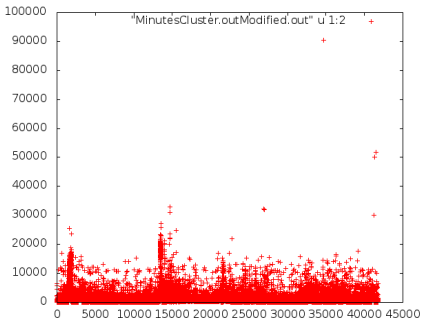
WAC has two main components, an analyzer and a classifier. The analyzer analyzes workloads to extract the workloads' autocorrelations and sample entropies which measure the periodicity and the burstiness of the workloads respectively. Periodicity and burstiness are the two main workload characteristics that effect a controller's performance. These two features are used with the business level objectives by the classifier to assign workloads to elasticity controllers. We start by analyzing 14 real workloads available from different applications. In addition, a set of 55 workloads is generated to test WAC on more workload configurations. We implement four state of the art elasticity controllers to which the classifier assigns workloads. We use a K nearest neighbors classifier and experiment with different workload combinations of training and test sets. Our experiments show that, when the classifier is tuned carefully, WAC correctly assigns between 92% and 98.3% of the workloads to the most suitable elasticity controller.

1 Introduction

Elasticity or auto-scaling can be defined as the ability of a cloud infrastructure (datacenter) to dynamically change the amount of resources allocated to a running application. Resources should be allocated according to the changing load



(a) Workload trace for the Wikipedia workload.



(b) Workload trace for the Google Cluster workload.

Figure 1: Workload traces of the Wikipedia workload and the Google cluster workload.

allowing the addition and removal of resources to preserve Quality of Service (QoS) requirements at reduced cost. Typically, a variety of different applications with different workloads run in a cloud [1]. Even when a single application is running on the infrastructure, in the case for Software-as-a Service [2], different users usually have different usage patterns.

Some workloads have repetitive patterns. For example, the Wikipedia workload shown in Figure 1(a) has a diurnal pattern where the request arrival rate is higher during the day than at night. Other workloads have seasonal patterns, e.g., the workload of an online store may increase drastically before Christmas [3]. Some uncorrelated spikes and bursts can occur in a workload due to an unusual event, e.g., shortly after Michael Jackson's death 15% of all requests directed to Wikipedia targeted the article about him [4] causing a significant spike. On the other hand, some workloads have some weak patterns or no patterns at all, such as the workload shown in Figure 1(b) for a Google cluster workload. Cloud infrastructure providers do not usually know the characteristics of the workloads their customers are planning to run.

Elasticity controllers are used to predict future workload and provision resources based on the predictions [5, 6, 7, 8, 9]. In our previous work, we have designed three adaptive autonomic elasticity controllers and tested them with three different workload traces; the FIFA worldcup 1998 workload trace [10, 11], Wikipedia traces from 2009 [2, 12] and a recently released workload from a production Google cluster [13, 14]. The proposed controllers showed variations in performance with different workloads. These variations are attributed to the different characteristics of different workloads. Most of the previous studies were tested using less than three real workload traces [5, 6, 7, 8, 9].

Since there are no perfect controllers [15] or perfect estimators [16], designing a general elasticity controller that produce accurate predictions for all workloads and scenarios running on a datacenter is infeasible. Elasticity controllers' performance varies with the different workloads and changing system dynamics. A controller tuned for certain workload scenarios can become unrobust if the

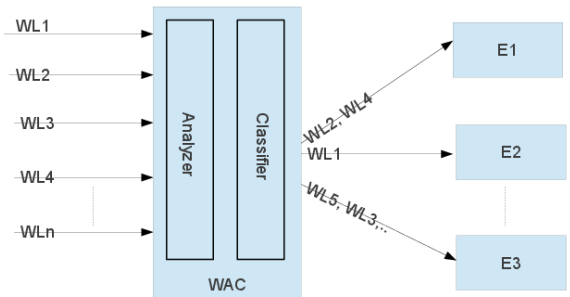


Figure 2: WAC: A Workload Analysis and Classification Tool.

conditions change [17] leading to wrong predictions and thus wrong capacity provisioning decisions. There is a cost if a controller adds more resources to a service than actually required by the service's workload since these extra resources are not utilized, although paid for by the customer. Similarly, if the resources provisioned for a service are not sufficient for its load, the service performance deteriorates and part of the workload is dropped. The service can become unresponsive or crash. Thus, analyzing workloads is an important first step towards choosing the appropriate elasticity controller for each workload. Based on this analysis, the service provider can assign the workload to the most suitable elasticity controller.

This work presents WAC, a Workload Analysis and Classification tool for cloud infrastructure providers. The tool analyzes workloads of different applications and based on the analysis it classifies the workloads into a set of predefined classes. Each class is then assigned to the most suitable elasticity controller for that class, reducing the risk of wrong predictions and improving the provided Quality of Service (QoS). It is assumed that there is some historical data available for the workloads. While not the case for all applications, many applications will have this data if they have been running in the past.

Figure 2 shows the two main components of WAC; the analyzer and the classifier. The analyzer analyzes historical data of a workload to discover some of its statistical properties that can be used by the classifier to find the most suitable elasticity controller for the workload of an application. There are many properties that can be used to characterize a workload such as the mean request

arrival rate, the variance of the request arrival rate, average request processing time, the workload's periodicity and the workload's burstiness [18].

In this work, we concentrate on *periodicity* and *burstiness* to characterize a workload. Periodicity describes how certain parts of the workload repeat with time, i.e., the similarity between the value of a workload at any point in time to the workload's history. Such similarity, if discovered, can be used by the elasticity controller to predict the future values of the workload. Burstiness measures load spikes in the system, when a running service suddenly experiences huge increases in the number of requests. Bursty workloads where the workload does not follow any pattern for these bursts, are hard to predict. We discuss our choice of these two properties and the methods we use to quantify them in more detail in Section 2.

Classification is a form of supervised learning used to predict the type or class of some input data object based on a group of features [19]. A data object can be anything from a single network request to a data stream [20]. For WAC, the data object is a workload and the classes or types are the different elasticity controllers. Supervised learning algorithms require training using labeled training data, i.e., objects for which the classes are known. The training data should be sufficient to enable the classifier to generalize to future inputs that arrive after training, i.e., new applications and workloads which will start running on the infrastructure in the future. To construct our training set, we collect and analyze a set of 14 real workloads. Since available real workloads are scarce [4], and the accuracy of any classifier depends on the size of the training set, we generate 55 additional synthetic workloads that have more general characteristics compared to the real workloads. Workload analysis and generation are discussed in Section 3.

Each class maps to an available elasticity controller implemented and provided by the cloud provider. This enables a cloud provider or a cloud user to choose the right controller for a certain workload reducing the aggregate prediction errors and allowing an improvement in QoS. Many designs for elasticity controllers are suggested in literature using different approaches such as Control theory [5], Neural networks [21], second order regression [7], histograms [6] and the secant method [22]. A cloud provider has to choose which controllers to implement. We have selected and implemented four controllers to use with WAC as discussed in more details in Section 4.

There are plenty of classifiers suggested in the machine learning literature, out of these Support Vector Machines (SVMs) and K-Nearest-Neighbors (KNN) are very popular [23]. KNN is chosen as the classification algorithm for WAC. We discuss the underlying principles of the two methods and our choice of KNN over SVMs in Section 5. We also present the training of the classifier, the classifier accuracy and discuss our results. Section 6 reviews the related work. We conclude in Section 7.

2 The Analyzer: Workload Characterization Techniques

Characterization of workloads is not an easy task since there are many parameters that can be used for the characterization [24]. With respect to predic-

tion and elasticity, the two main characteristics that affect the performance of workloads are *periodicity* and *burstiness*. As will be discussed later, these two measures are normalized for different workloads making their measurements depending only on workloads' patterns and trends.

2.1 Measuring Periodicity and Burstiness

The most common method to measure periodicity is to use autocorrelation. Autocorrelation is the cross-correlation of a signal with a time-shifted version of itself [16]. A signal can be any measurement, observation or function that changes with respect to time, space or any other dimension, e.g., a workload, an audio signal or a time series [25]. Autocorrelation takes values between -1 and 1. An autocorrelation of 0 at a time lag of τ means there is no relation between the signal and itself at this time lag. A random signal has an autocorrelation of 0 at all time lags. An autocorrelation of 1 means that the signal strength is exactly the same for this time lag while an autocorrelation near -1 means that the signal has an opposite direction of influence, i.e., if the signal strength is above average, the next value will be likely less than average. The autocorrelation function (ACF) describes the autocorrelation as a function of the time-lag τ. The ACF, R(τ), for a signal X is,

$$R(\tau) = \frac{E[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2}, \tag{1}$$

where E is the expected value operator, X_t is the value of the signal at time t, μ is the mean of the signal and σ is the variance of the signal. We use the ACF as a measure of workload periodicity. For the rest of the paper we use the terms autocorrelation and correlation coefficient interchangeably.

There are many methods to measure burstiness [26], non of them prevalent though. Gusella [27] suggested the use of the index of dispersion. Minh et. al. [28] suggested the use of normalized entropy. We propose the use of Sample Entropy (SampEn) as a measure of burstiness [29]. Sample entropy is a modification of the Kolmogrov Sinai entropy. It is the negative natural logarithm of the conditional probability that two sequences similar for m points are similar at the next point. It has been used successfully in classifying abnormal (bursty) EEG signals and other physiological signals for over a decade and has been proven robust [30]. An advantage of sample entropy over the index of dispersion and normalized entropy is that SampeEn detects periodic bursts.

Two parameters are needed to calculate SampEn for a workload. First parameter is the pattern length m, which is the size of the window in which the algorithm searches for repetitive bursty patterns. The second parameter is the deviation tolerance r which is the maximum increase in load between two consecutive time units before considering this increase as a burst. When choosing the deviation tolerance, the relative and absolute load variations should be taken in account, For example, a workload increase requiring 25 extra servers for a service having 1000 VMs running can probably be considered withing normal operating limits, while if that increase was for a service having only 3 servers running then this is a significant burst. Thus by carefully choosing an adaptive r, SampEn becomes normalized for all workloads. The deviation tolerance defines what is considered a normal increase and what is considered a burst. If

Data: r, m, W
Result: SampEn

$n \leftarrow Length(W)$;
$B_i \leftarrow 0$;
$A_i \leftarrow 0$;
$X_m = \{X_m(i) | X_m(i) = [x(i), x(i+1), ...x(i+m-1)] \; \forall \; 1 < i < n-m+1\}$;
for $(X_m(i), X_m(j))$ *in* X_m*:* $i < j$ **do**
    Calculate $d[X_m(i), X_m(j)] = \max(|x(i+k) - x(j+k)|) \; \forall \; 0 \leq k < m$;
    if $d[X_m(i), X_m(j) \leq r$ **then**
        $B_i = B_i + 1$;
$B^m(r) = \frac{1}{n-m} \sum_{i=1}^{n-m} \frac{1}{n-m-1} B_i$;
$m = m + 1$
$X_m = \{X_m(i) | X_m(i) = [x(i), x(i+1), ...x(i+m-1)] \; \forall \; 1 < i < n-m+1\}$;
for $(X_m(i), X_m(j))$ *in* X_m*:* $i < j$ **do**
    Calculate $d[X_m(i), X_m(j)] = \max(|x(i+k) - x(j+k)|) \; \forall \; 0 \leq k < m$;
    if $d[X_m(i), X_m(j) \leq r$ **then**
        $A_i = A_i + 1$;
$A^m(r) = \frac{1}{n-m} \sum_{i=1}^{n-m} \frac{1}{n-m-1} A_i$;
SampEn$= -\log[\frac{A^m(r)}{B^m(r)}$

Algorithm 1: The algorithm for calculating SampEn.

SampEn is equal to 0 then the workload has no bursts. The higher the value for SampEn, the more bursty the workload is.

We implemented the sample entropy algorithm as described by Aboy et. al. [31]. The algorithm is shown in Algorithm 1. W is the workload for which SampEn is calculated. The first loop in the algorithm calculates $B^m(r)$, the probability that two sequences (parts) in the workload having m measurements do not have bursts. The second loop in the algorithm calculates $A^m(r)$, the probability that two sequences in the workload having $m+1$ measurements do not have bursts. Then SampEn is calculated as the negative logarithm of the conditional probability that two workloads sequences of length m that do not have bursts, also have no bursts when the sequence length is increased by 1.

3 Workload Analysis

Typically, cloud providers such as Amazon [1] and Rackspace [32] host many services with different workloads. These services typically vary from services that use a handful of machines to services that use a few thousand machines [33]. With the exception of a recently released workload trace from a production cluster at Google [13], no cloud provider provides publicly available cloud workloads [4]. On the other hand, there are a few publicly available workloads for various types of Internet applications. We have analyzed some of these workloads in addition to the Google cluster workload in order to understand better the characteristics of the different workloads. We believe that these workloads are representative for many of the workloads running on typical clouds since they include major web sites and core technologies.

Recently, cloud providers started supplying resources for High performance computing and grid applications [34]. Many similar workloads are publicly available of which we use the DAS-2 system workload, the Grid-5000 workload, NorduGrid traces, SHARCNET traces, Large hydrogen collider Computing Grid (LCG) traces and the AuverGrid traces, all available from the grid workload archive [35]. Caching services are another type of application running on cloud infrastructures [1]. We use traces from the IRCache [36] project as representative workloads for this class of applications. The IRCache traces maintain traces from 10 different caching service sites in the USA. We only use traces from 4 sites. In addition, we analyze a one month trace of PubMed server usage [37], traces from the FIFA 1998 world cup [38], the Google cluster data released [13] and traces from Wikipedia [12]. Appendix A contains the graphs for the analyzed traces.

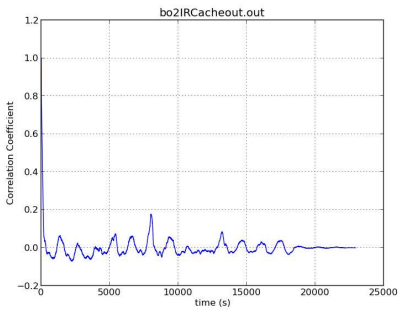
3.1 Periodicity of Real Workloads

A correlogram is a type of plot that shows the ACF of a signal at different lags. The x-axis of the plot represents the lag in minutes while the y-axis represents the autocorrelation. Figures 3, 4 and 5 show the correlograms of the analyzed workloads. Figures 3(a) and 3(b) show high ACFs for short lags meaning that for these workloads the value of the workload depends on its value in the near past. Figures 3(c), 3(d), 5(a), 5(c) and 5(d) show high autocorrelation coefficients meaning that their corresponding workloads have strong periodicity. Figures 4 and 5(b) show that their corresponding workloads are almost random. The second column in Table 1 describes the ACFs for the all workloads qualitatively.

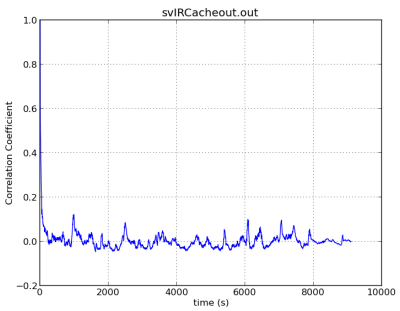
The coefficient of determination is defined as the squared value of the correlation coefficient at a certain lag. It measures the proportion of variance in a dependent variable that can be explained by the independent variable, i.e., the dependence of the future data on the historical data of the workload [39]. Workloads with higher autocorrelation coefficients at certain time lags are easier to predict at these time lags since their coefficient of determination is high.

3.2 Burstiness of Real Workloads

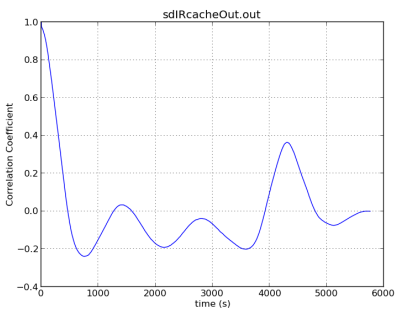
There are two main limitations of SampEn. First, SampEn is expensive to calculate both CPU-wise and memory-wise. The computational complexity (in both time and memory) of SampEn is $O(n^2)$ where n is the number of points in the trace. Second, workload characteristics can change during operation, e.g., when Michael Jackson died 15% of all requests directed to Wikipedia where to the article about him creating spikes in the load [4]. If SampEn is calculated for the whole past, then such recent changes are not discovered easily. To overcome these two limitations, we divide a trace into smaller equal sub-traces and calculate SampEn for each sub-trace allowing us to calculate a weighted average for SampEn giving more weight to more recent SampEn values. This reduces the time required for computing SampEn for a long trace spanning months or years to just a few hours. It is also suitable for online characterization of a workload since SampEn does not have to be recomputed for the whole history but rather for the near past which is much more efficient and fast. Our approach of dividing the signal into smaller sub-traces is similar to the approach used by Costa et. al [40] where they divide a signal to calculate its multi-scale



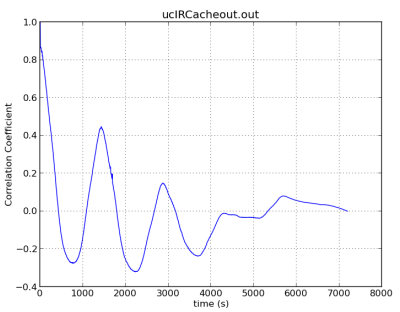
(a) Correlogram for IRCache service runnings at Boulder, Colorado (Bo).



(b) Correlogram for IRCache service running at Silicon Valley, California (SV).

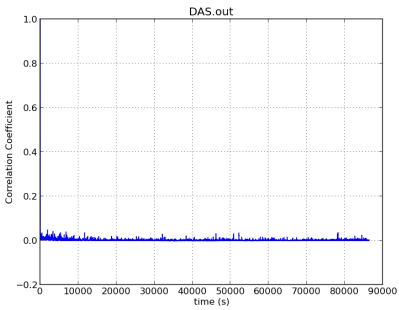


(c) Correlogram for IRCache service running at San Diego, California (SD).

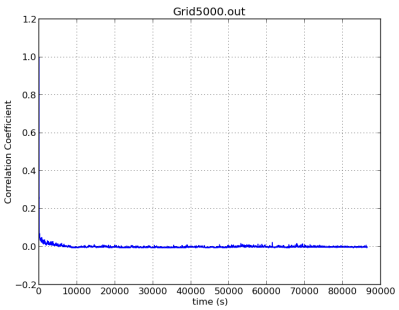


(d) Correlogram for IRCache service running at Urbana-Champaign, Illinois (UC).

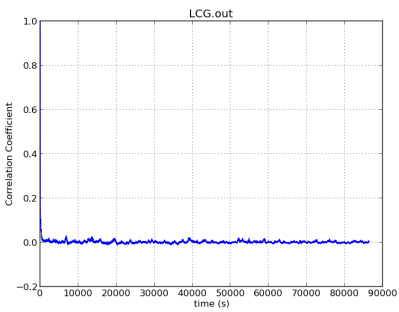
Figure 3: Correlograms of the different Caching Services.



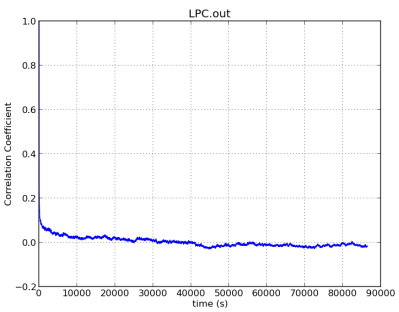
(a) Correlogram for the DAS workload.



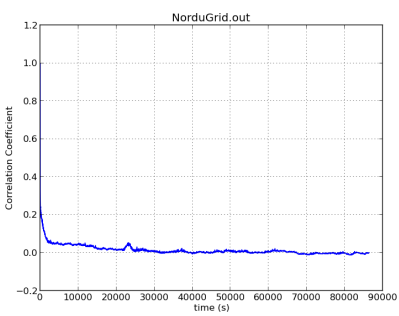
(b) Correlogram for Grid5000 workload.



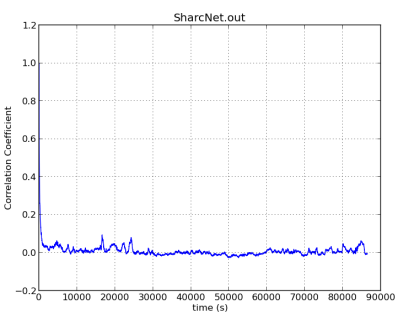
(c) Correlogram for the LCG workload.



(d) Correlogram for LPC workload.

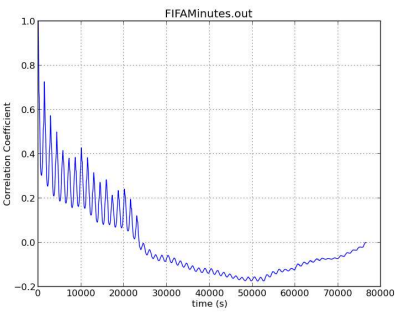


(e) Correlogram for the NorduGrid workload.

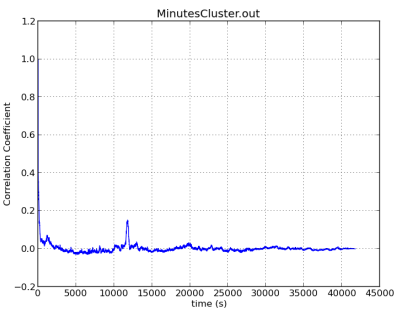


(f) Correlogram for SharcNet workload.

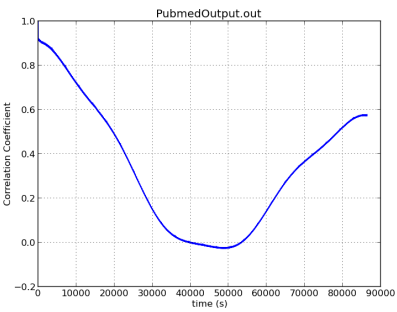
Figure 4: Correlograms of the Grid workloads analyzed.



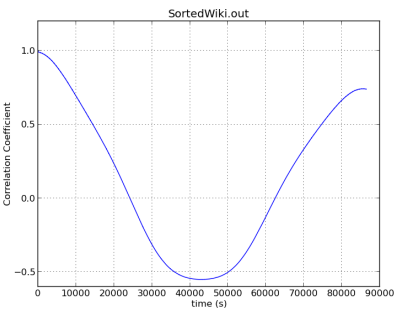
(a) Correlogram for the world cup workload.



(b) Correlogram for the Google Cluster workload.



(c) Correlogram for PubMed access traces.



(d) Correlogram for the Wikipedia workload.

Figure 5: Correlograms of web-hosting workloads and the Google cluster workload analyzed.

Table 1: Workload analysis results.

Workload	ACF	SampEn
Bo-IRCache	Very low. Almost a random signal	0.0
SV-IRCache	Very low. Almost a random signal	0.0017
SD-IRCache	High for small lags. Medium for large lags	1.062
UC-IRCache	High for small lags. Medium for large lags	0.0
DAS	Random signal	0.0
Grid5000	Random signal	236.16
LCG	Random signal	134.0
LPC	Random signal	2.827
NorduGrid	Random signal	8.43
SharcNet	Random signal	2.803
FIFA	High for small lags. Medium for large lags	0.0
Google	Very low, almost a random signal	235.70
PubMed	High	0.0
Wikipedia	High	0.0014

entropy. The main difference between the two approaches is that after SampEn is computed for the smaller sub-trace, Costa et.al plot the results while we take a weighted average.

As already mentioned, there are two main parameters needed to calculate SampEn, the pattern length m and the deviation tolerance r. In addition to these two parameters we add L, the size of each sub-trace when the whole trace is divided. These 3 parameters are set as following:

1. The pattern length m is set to 1 hour.

2. The deviation tolerance r is set to 30% of the seventieth percentile of the workload values. For example, if the seventieth percentile of the workload values seen is 20000 requests per second, then r is set to 6000 requests. If the number of servers required to handle the load is less than 10 servers, the deviation tolerance is set to double the average requests served by a server per unit time, since 30% of the maximum workload can be less than the load handled by one server.

3. The value of L is set to 1 day.

The third column in Table 1 shows the average SampEn computed for the different workloads. From the table we can see that workloads vary greatly in the amount of burstiness they have. With the exception of the DAS workload, workloads with ACFs around zero have significant burstiness as SampEn is greater than 1 for all of them, while the ones with high or medium ACFs do not show significant burstiness.

3.3 Workload Generation

To obtain accurate classification from a classifier, enough training samples are needed. Available real workloads are scarce and using them only as a training

Table 2: Average Sample Entropy computed for the sample generated workloads.

Workload	SampEn
WL1	0.0
WL2	236.0
WL3	2.5
WL4	0.002
WL5	0.0014
WL6	0.069

set for a classifier could result in poor ability to generalize and accurately classify newly deployed applications' workloads. We have generated 55 synthetic workloads using a mixture of functions and probability distributions. Due to space limitations and since this is not the main focus of this work, we omit most of the details on workload generation. For illustration purpose, Equation 2 shows how WL1 whose ACF is shown in Figure 6(a) is generated.

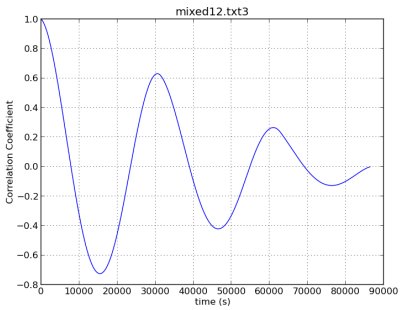
$$x(i) = \log(i+10^3)(|\sin(\frac{3i}{10^4})+100|)(|\sin(\frac{i}{10^4}))+2*10^4|)+W(0.3), \quad \forall\, i \in [0,n], \tag{2}$$

where $x(i)$ is the workload value at time i, $W(0.3)$ is Weibull distribution with a shape parameter equal to 0.3 and n is the length of the synthetic workload required. The log function is used to simulate a workload evolving with time but having similar periodicity. The sinusoids are used to give the workload a pattern similar to diurnal patterns and the Weibull distribution adds some uncertainty in the workload. This is similar to the pattern seen in the Wikipedia workload trace. Figures 6(b) and 6(c) show the correlograms for WL2 and WL3. The two workloads are generated with an equation similar to Equation 2, but with an added component that increases the uncertainty (SampEn) in the workload thus decreasing the ACF. Figures 6(b) and 6(d) show that the ACFs for WL2 and WL4 resemble the ACFs of the DAS, Grid5000 and LCG real workloads whose ACFs are shown in figures 4(a)–4(c) respectively. Figures 6(e) and 6(f) have ACFs different from the ones seen in the 14 investigated workloads. The equations used to generate the 55 synthetic workloads have mostly similar structures, but using different periods for the sinusoids, different amplitudes and different probability distributions such as log normal distributions, gamma distributions and chi-squared distributions as sources of uncertainty.

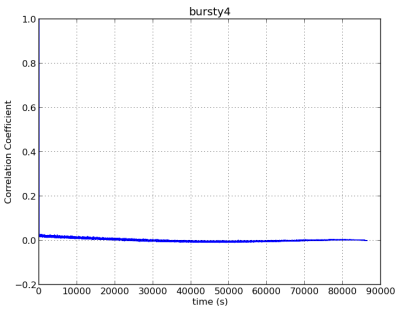
Table 2 shows the SampEn values for the six generated workloads above. Similar to the real workloads, the generated workloads have burstiness levels ranging from no burstiness to very bursty.

4 On the Performance of Different Elasticity Controllers

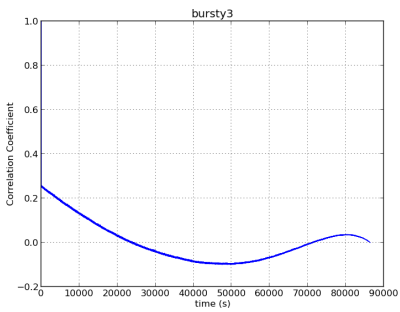
Since different elasticity controllers exhibit different performance with different workloads, a cloud infrastructure provider has to select a number of elasticity controllers to implement. These controllers are the classes to which WAC assigns



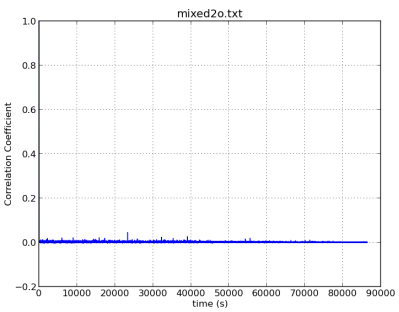
(a) Correlogram for a sample generated workload WL1.



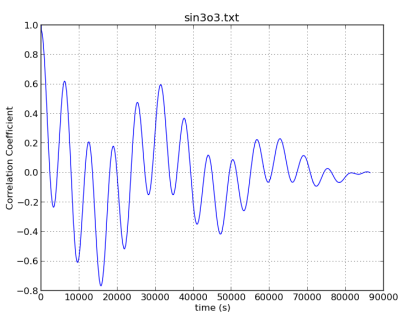
(b) Correlogram for a sample generated workload WL2.



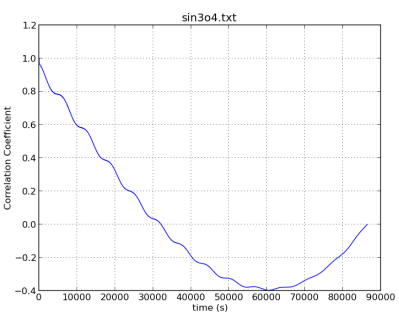
(c) Correlogram for a sample generated workload WL3.



(d) Correlogram for a sample generated workload WL4.



(e) Correlogram for a sample generated workload WL5.



(f) Correlogram for a sample generated workload WL6.

Figure 6: Correlograms of six of the generated workloads.

the different workloads. We implement 3 state-of-the-art elasticity controllers plus a step controller (the vanilla elasticity controller). These controllers represent the classes to which WAC assigns workloads. The implemented controllers are:

1. The step controller (hereafter called *React*) is a simple reactive controller where the capacity provisioned follows the load changes when they happen and does not predict the required future capacity. If the load is constant, no VMs are added or removed. If the load increases by $\Delta Load$, the controller reacts to the load increase by provisioning S servers that can handle this load increase. Similarly if the load decreases by $\Delta Load$ which is less than a certain threshold, the controller reacts by removing the extra machines. This is similar to the controller described by Chieu et al. [41]. If the controller computes the required capacity every time the load changes, the controller never allocates more resources than required since resources are only allocated when the load increases and are removed once the load decreases below a certain threshold.

2. A Histogram based controller (thereafter called $Hist$). This controller builds histograms for the workload observed within a defined time interval (e.g. between 9:00 and 10:00 on Monday) based on the past values observed. The controller estimates the peak demand of the workload to occur within the defined time interval before that interval begins. The histogram stores the history of the maximum arrival rates observed during the defined period over the past several periods. For each period there is a corresponding histogram which can be used to find the probability distribution of the arrival rate for that period. The controller can predict the capacity required based on a higher percentile of the tail, e.g., the 99^{th} percentile, of the distribution. This design allows the controller to capture seasonal trends such as diurnal or yearly trends depending on the period chosen for building the histograms. The period length chosen for the histogram is 1 hour. The controller has a correction mechanism in case the provisioned capacity is less than the actual required capacity. When the service load becomes more than the provisioned capacity can handle, the correction mechanism provisions more resources to the service. This controller was described by Urgaonkar et. al. [6].

3. A Regression based controller (thereafter called *Reg*) proposed by Iqbal et. al. [7]. The controller is a hybrid controller with a reactive controller for scale-up decisions and a predictive controller for scale-down decisions. When the capacity is less than the load, a scale up decision is taken and new VMs are added to the service in a way similar to *React*. For scale down, the predictive component uses second order regression trained using past workload values to predict future workload. The regression model is recomputed using the complete history of the workload every time a new measurement data is available. If the current load is less than the provisioned capacity, a scale down decision is taken using the regression model. If the regression predicts an increase in the number of servers, the results from the regressor are ignored.

4. An adaptive hybrid controller (thereafter called $AKTE$) that uses the workload slope to predict the future workload. This controller is proposed

by Ali-Eldin et. al. [14]. The controller is a hybrid controller having a reactive component similar to *React* and a predictive component that predicts the future workload from the rate of change of the workload. The slope calculations are adaptive depending on the rate of change of the workload. If the load is increasing or decreasing rapidly and thus the rate of change is high, then the rate at which the controller makes decisions is increased in order to adapt to the rapid workload changes.

4.1 How to Compare Elasticity Controllers' Performance

While most of the body of work on elasticity control uses the request response time or service latency as a measure of a controller's performance, Bodik et. al. [42] argue that this is not the correct reference signal that an elasticity control policy should use since latency have very high variance even for fixed workloads. If this is the reference signal used, the elasticity controller ends up oscillating resources due to noise in the reference signal. Smoothing the noise does not help since it leads to significant control delays. Bodik et. al. also raise another concern for which they give an example. Suppose there are two workloads running on two different machines and each of them experiences an average service latency of 50 ms for the requests. If the required QoS is to keep the average service latency below 100 ms, can the two workloads be handled by a single server? This is not true since service latencies do not depend linearly on the amount of resources provisioned.

They also point out to the fact that latency does not show under-provisioning and over-provisioning levels even for non noisy response times that do not vary. This is specially true when we compare the performance of different controllers since two controllers might be achieving the required latency but one of them uses one tenth of the resources used on average by the other controller. Similarly, one controller can be dropping, on average, less requests than another one.

We add to their analysis that different applications have different processing requirements and a cloud infrastructure provider should not make any assumptions on how the resources are used, e.g., serving a static web page page hosted on the cloud is different from sorting a Terabyte of data. In addition, any change in the workload mix changes the achievable latency [9]. The cloud paradigm is centered around resources on demand, so the measure of performance should be resource usage, e.g., average CPU and memory consumption or network utilization, or a metric that can be directly mapped to resource usage, which is not the case for latency. These metrics enable the provider to be as generic as possible when specifying the QoS in the Service Level Agreements (SLAs) with the customers. We choose to use the request arrival rate as the reference signal in the implementation of the different controllers.

4.1.1 Performance Comparison Metrics

To compare the performance of different controllers, we use the following cost metrics:

1. Average Overprovisioning rate ($\overline{OP}$) is the average number of overprovisioned VMs by the controller per unit time. This is calculated by summing the number of overprovisioned VMs over time and dividing the number

by the total number of time units for which the controller was running. A machine is considered overprovisioned if it is of no use for the next 10 minutes. This reduces the penalty of an algorithm that predicts the future workload well in advance. This is a configurable parameter that can be changed.

2. Similarly, average Underprovisioning rate ($\overline{UP}$) is the average number of underprovisioned VMs by the controller per unit time. This is calculated by summing the number of underprovisioned VMs over time and dividing the number by the total number of time units for which the controller was running. Underprovisioning means that the controller failed to provision the resources required to serve all requests on time.

3. Average number of Oscillations ($\overline{O}$) which is the average number of VMs started or shut-down per unit time. The reason to consider ($\overline{O}$) as an important parameter is the cost of starting/stopping a VM. From our experience, starting a machine (physical or virtual) typically takes from 1 minute up to several minutes (almost 20 minutes for an ERP application server) depending on the application running. This time does not include the time required to transfer the images and any data needed but is rather the time for machine boot-up, recontextualization of the images [43] and network setup. Similar time may be required when a machine is shutdown for workload migration and load balancer reconfiguration. In addition, for physical servers, machines consume more energy during bootup and shutdown while not doing any useful work [44].

Since different applications have different provisioning requirements, some applications running on a cloud infrastructure will give more weight to one of the three parameters. For example, a traditional database system might care more about $\overline{O}$ since each change in the number of machines results in a heavy penalty in terms of synchronization to preserve consistency. While an application that uses VM cloning [45] might care more about $\overline{UP}$.

When comparing different controllers, the values of $\overline{OP}$, $\overline{UP}$ and $\overline{O}$ are weighted in order to consider the different scenarios and combinations that an application might have. $\overline{OP}$ is multiplied by a constant α. $\overline{UP}$ is multiplied by a constant β. Similarly, $\overline{O}$ is multiplied by a constant γ. The values for these three weights should be set based on the business level objectives [46] required by the application owner and the quality of service requirements of a service.

We refer to the set of implemented controllers as C_I. To choose the best performing controller for a given workload and a given set of weights, the cost metrics have to be calculated for each implemented controller X. The total cost for using that controller is,

$$Cost_X = \alpha\overline{OP} + \beta\overline{UP} + \gamma\overline{O}. \tag{3}$$

The best controller for the given setup C_Z is thus,

$$C_Z = \{\chi | \chi \in C_I \ \& \ Cost_\chi \leq Cost_Y \ \forall \ Y \in C_I\}. \tag{4}$$

4.2 Performance of the Controllers with Various Workloads

Classification is a supervised learning algorithms, therefore, it requires training using labeled data. For each object in the labeled training set, the object's class in known. For WAC, an object is a workload and a class is one of the implemented elasticity controllers described in Section 4. In order to label each workload in the training set, we need to know the best performing controller for that workload. We have built a discrete event simulator using Python, numpy and scipy [47] to simulate a cloud provider and test the four controllers and their performance on the workloads. The Regression controller was performing badly on almost all workloads for all three metrics. We have modified the design described in the original paper such that the provisioned capacity is always 1.2 times the suggested capacity by the controller output. This modification improves the performance of the controller considerably.

In our experiments, we set the parameters in Equation 3 such that α varies between 0 to 1 for the real workloads and 0 to 0.5 for the generated workloads, with a step of 0.01, β varies between 1 to 20 for the generated workloads and 1 to 10 for the real workloads with a step of 1 and γ varies between 0 to 20 for the generated workloads and 0 to 10 for the real workloads, with a step of 0.5. The combinations of these values with the ACFs and the SampEn of the different workloads form the different scenarios with which we test WAC. For the real workloads, there are in total 280000 scenarios, while for the synthetic workloads there are in total 2090000 scenarios. Using all combinations of the three weighted metrics for both the real and generated workloads, each combination is labeled with the best performing controller. Table 3 shows the percentages of scenarios in which each controller outperforms the other controllers for the different workloads. From the table we see that, *Reg* and $AKTE$ outperform the other controllers considerably for both the real and the synthetic traces. *React* is the worst performing controller for both workload types.

We attribute the performance of the controllers to the way they are designed. $AKTE$ is designed with the aim of reducing $\overline{UP}$ and $\overline{O}$ at the cost of higher $\overline{OP}$. It is more suitable for applications where the cost of underprovisioning and the cost of oscillations should be minimized. $Hist$ is designed for workloads that have periodical patterns and low burstiness. $Hist$ does not give any importance to $\overline{OP}$ since scale down mechanisms when the controller provisions extra resources are absent. *React* has no predictive component. It provisions resources after they are needed. This reduces the amount of $\overline{OP}$ at the cost of increased $\overline{UP}$ and $\overline{O}$. The modified *Reg* uses a second order regressor which is suitable for capturing patterns in the workload. *Reg* scales down resources faster than $Hist$, thus reducing $\overline{OP}$, but slower than *React*.

5 Workload Classification

The KNN and the SVM classifiers are among the most widely used classifiers [23, 48, 49]. SVM is a supervised learning method introduced by Boser et. al. [50]. It constructs separating hyperplanes to separate the training data such that the distance between the constructed hyperplanes and any training point is maximized. Training an SVM classifier and tuning its different parameters is

Table 3: Number of scenarios in which every controller outperforms the other controllers for the different workloads.

Controller	Real workloads scenarios	Generated workloads scenarios
React	6.55%	0.1%
Reg	33.72%	61.33%
AKTE	47.17%	34.3%
Hist	12.56%	4.27%

complex [51]. We have tried using an SVM classifier but the training was very time consuming and was giving low classification accuracy. This means that the classifier parameters required tuning which is even more time consuming. We choose to use KNN instead since it is less complex, fast to train and fast to calculate an assignment once trained.

The KNN algorithm is quite simple. A training set is used to train the classifier. The training set is a set of samples for which the correct classification in known. When a new unknown sample arrives, a majority vote of its K-nearest neighbors is taken. The new sample is then assigned to the class with a majority vote. K is a configurable parameter. The distance to a neighbor is usually the Euclidean distance between the sample and that point. There are two versions of the KNN algorithm. One version gives more weights to votes of closer neighbors while the second version gives equal weights to all K neighbors [23].

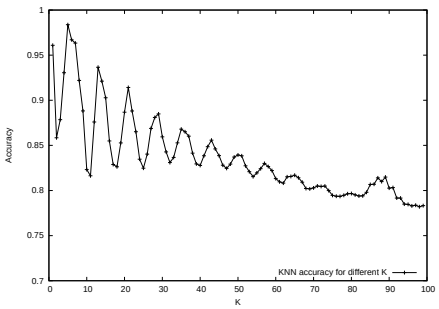
In the experiments, we consider the case when a cloud provider has four controllers deployed and the case when only two controllers, $AKTE$ and Reg, are deployed. $AKTE$ and Reg perform better than $Hist$ and $React$ for more than 80% of the workload scenarios considered. Each scenario in the training set is labeled with the top performing controller out of $AKTE$ and Reg for the experiments with only two controllers.

In order to test the accuracy of KNN, the available workload scenarios are divided into a training set and a test set. The training set is used for training the classifier while the test set is used to evaluate the accuracy of the classifier after training. Both sets are labeled with the right answer for each scenario. When the classifier assigns a workload to an elasticity controller, the assignment is compared to the label. If the assignment matches the label, then the classification is correct. The accuracy of the classifier is calculated by calculating the ratio of correct classifications of the test set to the total size of the test set.

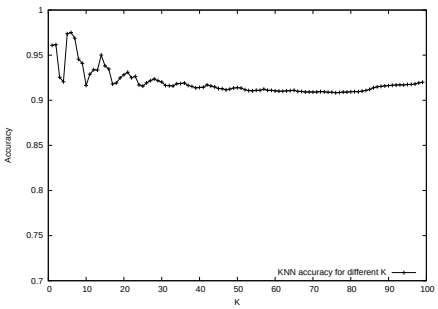
5.1 Classification of Real Workloads

In the experiments, the scenarios described in Section 4 are randomly split into a training set and a test set. The training set contains 90% of all the scenarios while the test set has the remaining 10% . Both the training set and the test set are balanced such that no single class dominates the training set thus skewing the classifier to favor a certain class.

We test the KNN classifier using both its versions, the one that gives equal weights to all K neighbors and the one that gives more weight to closer neighbors. For each version, we repeat the experiment with K ranging from 1 to 100 and calculate the classification accuracy.



(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.



(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

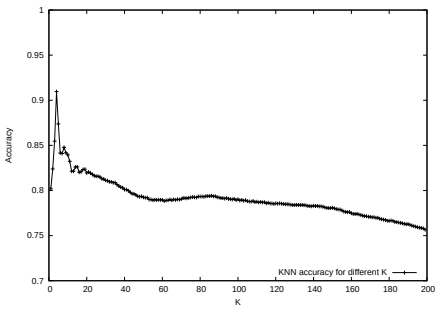
Figure 7: Effect of K on the classifiers' accuracy with only 2 classes considered, $AKTE$ and Reg, for real workloads.

Figure 7 shows the classification accuracy of the classifier when there are only two classes, $AKTE$ and Reg. The x axis of the figure is the value of K. The y axis of the figure is the accuracy of classification. Figure 7(a) shows the accuracy of the classifier with changing K when all K neighbors' votes are given equal weights. Significant oscillations occur in the accuracy as K increases until K is around 50. When closer neighbors' votes are given higher weight, oscillations are much smaller as shown in Figure 7(b) where the accuracy stabilizes at around 0.92 when K is equal to 25. The maximum accuracy achieved is 0.983 using equal weights for all neighbors for $K = 5$.

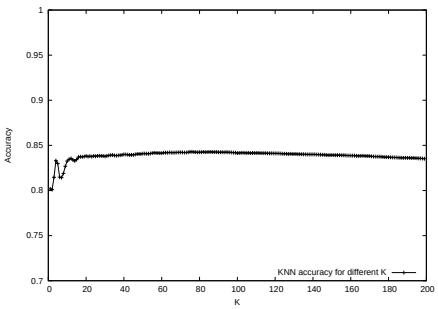
Figure 8 shows the classification accuracy of the classifier when there are four classes, $AKTE$, $React$, $Histo$ and Reg, against K when K is varied from 1 to 200. Figures 8(a) and 8(b), show less oscillations compared to Figure 7. The maximum accuracy achieved is 0.91 using equal weights for all neighbors for $K = 4$. The accuracy of classification is reduced by more than 7% when more controllers are available. This might be due to not having enough training data in the training set for the classifier to be able to generalize for four classes. An accuracy of 91% is still considered good. For the service provider it means that instead of having just one controller, the provider can have 4 controllers with only 9% of the workloads assigned in a non optimal way. Since more training data, means more experience, the provider can further decrease the percentage of workloads assigned sup-optimally by constantly adding feedback to the classifier by adding more training points based on new workload behaviors seen. Figures 7(a) and 8(a) show that accuracy decreases with increasing K when all K neighbors' votes are given equal weights. When neighbors' votes are weighted such that closer neighbors have a higher weight, the accuracy stabilizes as K increases as shown in Figures 7(b) and 8(b).

5.2 Classification of Generated Workloads

We repeat the tests described above using only the generated workloads. Figure 9 shows the classification accuracy of the classifier when there are only two



(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.



(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 8: Effect of K on the classifiers' accuracy with 4 classes considered, mapping to the four implemented controllers, for real workloads.

classes, $AKTE$ and Reg, against K. The maximum accuracy achieved is 0.92 using equal weights for all neighbors for $K = 3$. Figure 10 shows the classification accuracy of the classifier when there are four classes, $AKTE$, $React$, $Histo$ and Reg, against K. The maximum accuracy achieved is 0.94 using equal weights for all neighbors for $K = 3$.

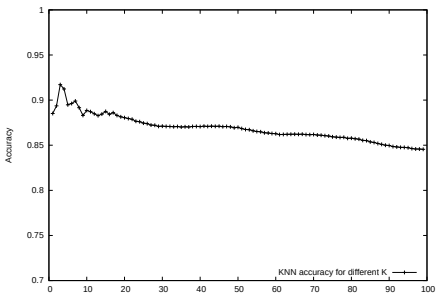
Comparing Figure 7(a) to figures 9 and 10, oscillations in the classifier's accuracy with increasing K is much lower when classifying synthetic workloads compared to when classifying real workloads.

5.3 Classification of Mixed Workloads

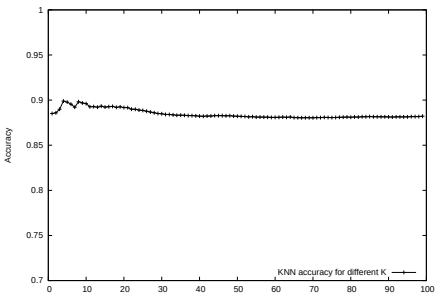
In our last experiment, we combine the scenarios using real and generated workloads in a single set. This set is again randomly shuffled and divided into a training set and a test set. Figure 11 shows the classification accuracy of the classifier when there are only two classes, $AKTE$ and Reg, against K. The maximum accuracy achieved is 0.92 using any of the two versions of the KNN for $K = 5$. Figure 12 shows the classification accuracy of the classifier when there are four classes, $AKTE$, $React$, $Histo$ and Reg, against K. The maximum accuracy achieved is 0.92 using equal weights for all neighbors for $K = 3$. Again, we note that the oscillations in the classifier's accuracy with changing K in figures 11 and 12 are not as severe as in the case of classifying real workloads.

5.4 Discussion of the Results

The oscillations seen in the figures for lower values of K can be attributed to the nature of the KNN algorithm. Let F be the number of features used for classification. KNN constructs an F-dimensional space and places each training point in that space according to the point's features values. It divides the F-dimensional space into neighborhoods where each neighborhood is a group of points neighboring each other and having the same class. These neighborhoods can be scattered in the F-dimensional space. Some of these neighborhoods can

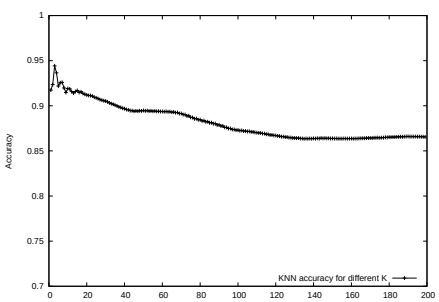


(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

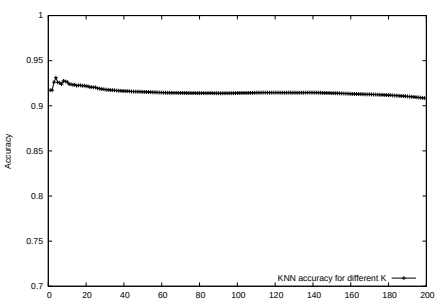


(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 9: Effect of K on the classifiers' accuracy with only 2 classes considered, $AKTE$ and Reg, for generated workloads.

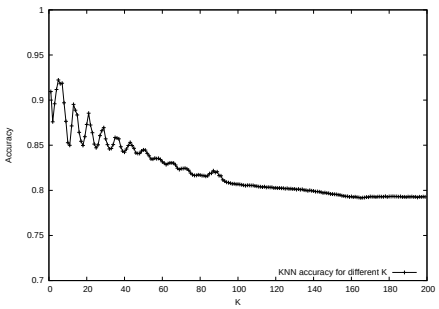


(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

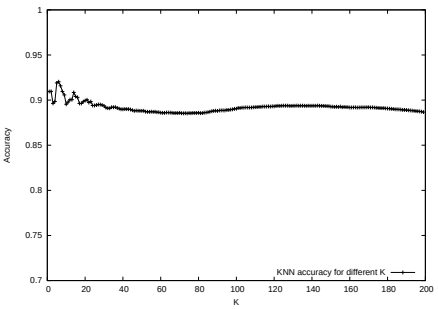


(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 10: Effect of K on the classifiers' accuracy with 4 classes considered, mapping to the four implemented controllers, for generated workloads.

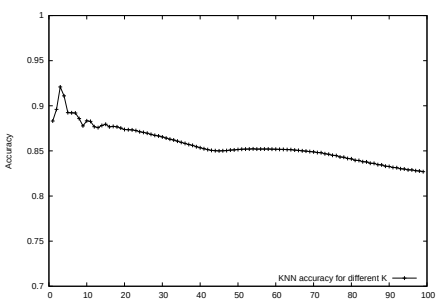


(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.

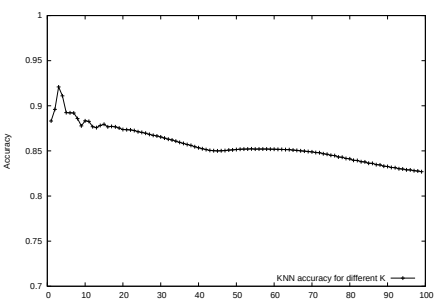


(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 11: Effect of K on the classifiers' accuracy with only 2 classes considered, $AKTE$ and Reg, for a mixed set.



(a) Effect of changing K on the classifier accuracy when all neighbors' votes are given equal weight.



(b) Effect of changing K on the classifier accuracy when closer neighbors' votes are given higher weight.

Figure 12: Effect of K on the classifiers' accuracy with 4 classes considered, mapping to the four implemented controllers, for a mixed set.

be small and surrounded completely by a larger neighborhood of a different class. Picking a small value for K results in correctly classifying these smaller neighborhoods for the training data. On the other hand, it can result in overfitting the training data and forming nasty decision boundaries, e.g., a small neighborhood laying completely surrounded within a large neighborhood of a different class. This makes the classifier more sensitive to noise in the training set. A large value of K results in a less noise sensitive classifier and smoother decision boundaries but at the cost of increased training data error, increased computational burden and loss of locality of the estimations since far away points affect the decision [52].

Oscillations are more obvious when the distance to all K neighbors have equal weights, When closer neighbors are given more weight, the classification accuracy is more stable with increasing K. By looking at figures 7–11, it can be seen that for all of them, the accuracy of classification decreases with the increasing K for a while and then tends to stabilize for a range of K and then sometimes starts to decrease again. The stabilization for the KNN version with weighted distance is faster and with a stable accuracy higher than the other version.

When setting K in real life deployments, a cloud provider should either choose K with the best accuracy regardless of stability or least K with highest stable accuracy depending on the confidence in the training set. If the training set is comprehensive with most workload characteristics expected, K can be chosen to be the one that gives best accuracy on the training set regardless of the stability. Otherwise, if the training set is not comprehensive K should be chosen to be the least K with the highest stable accuracy.

6 Related Work

Previous work on workload characterization has mainly focused on identifying the different properties of different workload traces collected from running systems. Calzarossa and Serazzi [53] discuss the steps for modeling a workload. As a first step, the characterization level, e.g., the sampling rate of the workload, the basic components, the parameters that describe them and the criteria for evaluating the model accuracy of the workload has to be identified. Then, the system is monitored to log the workload. The measured data is statistically analyzed to know if there are any natural partitions in the workload, e.g., long running requests and short running requests. Statistical analysis involves removal of any outliers having one or more parameters with values greater than a certain percentile, workload sampling to be able to process the recorded trace within reasonable time, static analysis which partitions the workload using a clustering algorithm such as K-means and dynamic analysis using techniques such as time series analysis. The authors discuss a large number of workload modeling studies spanning different systems, namely, centralized batch and interactive systems, centralized database systems, network based systems, and multiprocessor systems such as parallel systems and supercomputers.

Downey and Feitelson [24] discuss the problems in generalizing a model based on data from specific observations to an abstract model of the workload. They start by giving some examples on the problems and the difficulties faced when creating a multi-level model for a workload. They focus on the problems of

single-level models.

The aim of modeling is to find the summary statistics of the different workloads. These fall into three families:

1. Moment-based: e.g., mean, variance and the coefficient of variance, skew (3rd moment) and kurtosis (4th moment).

2. Percentile-based, e.g., median, quartiles and the semi-interquartile range (Measure of Dispersion)..

3. Mode-based: e.g., the most common value observed for distributions having discrete components.

According to Downey and Feitelson, reporting the moments for non-symmetric distributions is misleading and might be meaningless since moments are not robust to outliers. In addition, for some distributions such as the Pareto distribution, the moment of a sample does not converge on the moments of the population. One of the warning signs of non convergence is the failure to achieve sample invariance. To test whether a hypothesis about a distribution is right or not, the Chi-square test or the Kolmogorov-Smirnov test (for continuous distributions) can be used [54]. These tests usually fail for workload characterization since they assume that the testing data should converge to the distribution when a large number of points are used. This is not true for most workloads.

The authors discuss the importance of weighing data points. They give an example with the characterization of parallel jobs when it is assumed that all jobs are of the same weight, when the jobs are weighed by the duration and when the jobs are weighed by the area (duration and the number of cores used). They discuss the correlation between these different attributes and the different ways to include them in the model.

In addition to studies on the methodologies that should be used when analyzing a workload, several researchers have analyzed in depth publicly available workloads. Arlit and Jin presented a detailed workload study of the 1998 FIFA World Cup website [38]. The site logs from May 1st, 1998 until July 23rd, 1998 were analyzed. The logs contain 1.35 Billion requests and almost 5 TB of data were sent to the clients. Data was accessed by more than 2 million unique IP addresses.

Kang et. al. crawled Yahoo! videos website for 46 days [55]. They recorded data for 9986 unique videos in total with video durations ranging from 2 to 7518 seconds. Around 76% of the videos are shorter than 5 minutes and almost 92% are shorter than 10 minutes. They discuss the predictability of the arrival rate with different time granularities. A load spike typically lasted for no more than 1 h and the load spikes are dispersed widely along the time making them hard to predict. They also find that the workload is highly correlated in short term. While the cost could be high if a video site operator over-provisioned the data center based on the over-provisioning factor at a small time scale, these load spikes can be provisioned on a cloud on demand to service the extra load while not over-provisioning the Video site datacenter.

Chen. et. al. analyze a short workload trace that was publicly realized by Google [56]. The trace is 6 hours and 15 minutes long with data collected every 5 minutes, i.e., having just 75 points. The short workload makes it hard to infer the true characteristics of the jobs running on Google's backend cluster.

Mishra et. al. take a first step towards modeling the workload running on Google's backend servers by grouping resources with similar task consumptions in classes (task classification) [57]. They define a multidimensional representation of the resources (task shape) used by the tasks using the average memory in GB and the average number of cores used by a task every five minutes. During scheduling, tasks are assigned such that their shape (time in seconds, memory usage and number of cores) fits the free space on the assigned machine. They discuss the problems with using a coarse grained classification and explain their approach for task classification. First, they identify the workload dimensions, e.g., task duration, average core usage and average memory usage. Second, they use K-means clustering algorithms to construct preliminary task classes. Third step is to manually identify the break points for the qualitative coordinates of the workload dimensions. Finally, they merge classes to form the final set of task classes which define the workload. They use this approach to identify the load mix of the tasks.

This work is extended by Zhang et. al. [58] who suggest to model task usage shapes by modeling the mean values of run-time tasks resource consumption. They call this model "the mean usage model of tasks usage shapes". This simple model captures the characteristics of the workload running on Google's backend cluster to a high extent. They attribute these results to the low variability of task resource usage in the workload and the characteristics of evaluation metrics. The authors also analyze six month of Map-Reduce traces from a Facebook cluster and a two weeks Map-Reduce trace from another Internet company [59]. They built a Map-Reduce workload synthesizer that analyzes a Map-Reduce workload and uses the results to produce a synthetic but realistic workload that mimics the original seed workload.

A longer trace provided by Google that spans 29 days and around 650000 jobs [13] was analyzed by Reiss et. al. [60]. While the system is over-booked, the actual CPU and memory utilization is rarely higher than 50% and never exceeds about 60%. The workload is dominated by 'normal production' tasks which have regular patterns and a daily peak to mean ratio of 1.3. Lower priority tasks make up between 10% to 20% of the CPU usage in the cluster. These tasks have no regular patterns and are very bursty. Over 80% of the total utilization of the cluster comes from long running jobs which constitute 2% of the total number of jobs.

Iosup and Epema analyze 15 different grid workloads [61]. They report that the average system utilization of some research grids was as low as 10 to 15% but considerably high on production grids. All 15 grids experienced overloads during some short term periods. A few users dominate the workloads. In most workloads, there is a little intra-job parallelism and even parallel jobs running have low parallelism. They note that loosely coupled jobs are dominating most of the workloads. Loosely coupled jobs are suitable to run on the cloud [62]. Similar findings are reported by Zhang et. al. [58] and the Magellan project [62].

Khan et. al. study workloads running on different servers in a cluster in order to find which servers frequently have correlated workload patterns [63]. They use a greedy algorithm to solve a computationally intractable problem to find the clusters that constitutes correlated workloads on different servers. Their proposed solution is slow. It takes more than 6 hours to train the algorithm using data collected every 15 minutes for 17 days and doing predictions every 15 minutes.

Viswanatha et al [64], introduce a provisioning system for private clouds that uses an application's resource usage pattern to place applications on servers with appropriate level of configuration capabilities. The authors use the coefficient of variation to identify unstable workloads. Unstable workloads requires placement on clusters that allow dynamic placement. They base their placement decisions on the different elastic capabilities of the clusters and the expected peak workload for each application.

There has been a lot of interest in data stream classification and clustering [65, 66, 67, 20]. Beringer and Hullermeier modify the K-means clustering algorithm to be able to cluster online data streams efficiently. Their work is considered the first work to try to cluster the data streams themselves, compared to for example the work by Guha et. al. [67] who cluster the elements of the stream. Due to their complexity, the computational costs of clustering data streams is high. In addition, the algorithm has to be adaptive in the sense that new clusters may be created at any time when new data points arrive. Similarity between data streams is calculated using the Euclidean distance between their normalization for a given sliding time window.

Keogh and Kasetty discuss the quality of the published work on time-series data mining [68]. They show that much of the published work is of very little utility. They implement the contributions of more than two dozen papers and test them on 50 real workloads. For classification, they implement 11 published similarity measures and compare their performance to the Euclidean distance measure. None of the 11 proposed similarity measures is able to outperform the Euclidean distance. They show that the error rate for some of the published work was 0.695, that is 69.5% of all classifications were wrong.

Herbst et. al. [69] propose a novel forecasting methodology that adaptively correlates workload characteristics classes and existing time series based forecasting approaches. The authors smooth the noise in a times-series before forecasting. They use a decision tree to assign the smoothed time-series to one of several forecasting methods based on time-series analysis. They show that their approach can be used for dynamic VM provisioning.

Our work complements the previous studies and uses their findings to build an automated workload analyzer and classifier tool that can identify the classes of the different workloads running on the cloud. Although our main focus is elasticity control, the tool also be used for placement of orthogonal workloads (e.g. computationally intensive workloads and I/O intensive workloads). The tool can also provide admission controllers, which accepts new services to run on the datacenters, with insights that can be used when admitting a workload.

7 Conclusion and Future work

In this work, we introduce WAC, a Workload Analysis and Classification tool for assigning workloads to different elasticity controllers in order to improve workload predictions for auto-scaling. The tool uses autocorrelation and sample entropy to analyze the workload periodicity and burstiness respectively. This analysis is used for classifying workloads and assigning each of them to the most suitable elasticity controller. In addition, the tool allows assignment of an elasticity controller to a workload based on specific quality of service objectives. We have collected a set of 14 real workloads and generated a set of 55 synthetic

workloads to test the tool. We have implemented 4 state-of-the-art elasticity controllers. WAC was used to assign the workloads with different quality of service requirements to the implemented controllers in different experiments using a KNN classifier. The tool shows an accuracy of classification ranging from 91% up to 98% in different experiments.

Acknowledgment

The authors would like to thank the Advanced School for Computing and Imaging at Vrije Universiteit, Amestrdam for providing the DAS workloads, the Grid'5000 team for providing the Grid'5000 traces, the AuverGrid team for providing the AuverGrid traces, John Morton and Clayton Chrusch for providing the SHARCNET traces, the e-Science Group of HEP at Imperial College London for providing the LPC traces and the NorduGrid team for providing the NorduGrid traces. We would also like to thank the Grid Workloads Archive for providing access to these traces.

Financial support has been provided in part by the European Community's Seventh Framework Programme under grant agreement #257115, the Lund Center for Control of Complex Engineering Systems, the Swedish Government's strategic effort eSSENCE and the Swedish Research Council (VR) under contract number C0590801 for the project Cloud Control.

References

[1] Amazon Elastic Compute Cloud (Amazon EC2). Accessed: May, 2013. [Online]. Available: https://aws.amazon.com/solutions/case-studies/

[2] A. J. Ferrer, F. Hernndez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner, M. Corrales, N. Forg, T. Sharif, and C. Sheridan, "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66 – 77, 2012.

[3] J. Garside. Amazon's record $21bn christmas sales push shares to new high. Accessed: May, 2013. [Online]. Available: http://www.guardian.co.uk/technology/2013/jan/30/amazon-christmas-record-sales-profits

[4] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proceedings of the 1st ACM symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 241–252. [Online]. Available: http://doi.acm.org/10.1145/1807128.1807166

[5] H. C. Lim, S. Babu, and J. S. Chase, "Automated control for elastic storage," in *Proceedings of the 7th international conference on Autonomic computing*. ACM, 2010, pp. 1–10.

[6] B. Urgaonkar, P. Shenoy, A. Chandra, P. Goyal, and T. Wood, "Agile dynamic provisioning of multi-tier internet applications," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 3, no. 1, p. 1, 2008.

[7] W. Iqbal, M. N. Dailey, D. Carrera, and P. Janecek, "Adaptive resource provisioning for read intensive multi-tier applications in the cloud," *Future Generation Computer Systems*, vol. 27, no. 6, pp. 871–879, 2011.

[8] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, "Towards autonomic workload provisioning for enterprise grids and clouds," in *10th IEEE/ACM International Conference on Grid Computing, 2009.* IEEE, 2009, pp. 50–57.

[9] R. Singh, U. Sharma, E. Cecchet, and P. Shenoy, "Autonomic mix-aware provisioning for non-stationary data center workloads," in *Proceedings of the 7th international conference on Autonomic computing.* ACM, 2010, pp. 21–30.

[10] M. Arlitt and T. Jin. (1998, August) "1998 world cup web site access logs". Accessed: May, 2013. [Online]. Available: http://www.acm.org/sigcomm/ITA/

[11] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE.* IEEE, 2012, pp. 204–212.

[12] G. Urdaneta, G. Pierre, and M. Van Steen, "Wikipedia workload analysis for decentralized hosting," *Computer Networks*, vol. 53, no. 11, pp. 1830–1845, 2009.

[13] J. Wilkes. More google cluster data. Accessed: May, 2013. [Online]. Available: http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html

[14] A. Ali-Eldin, M. Kihl, J. Tordsson, and E. Elmroth, "Efficient provisioning of bursty scientific workloads on the cloud using adaptive elasticity control," in *Proceedings of the 3rd workshop on Scientific Cloud Computing.* ACM, 2012, pp. 31–40.

[15] M. Morari and E. Zafiriou, *Robust process control.* Morari, 1989.

[16] B. Abraham and J. Ledolter, *Statistical methods for forecasting.* Wiley, 2009, vol. 234.

[17] M. Morari, "Robust stability of systems with integral control," *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 574–577, 1985.

[18] D. Feitelson, "Workload modeling for computer systems performance evaluation," *Book Draft, Version 0.38*, 2013.

[19] D. Barber, *Bayesian reasoning and machine learning.* Cambridge University Press, 2012.

[20] C. Aggarwal, J. Han, J. Wang, and P. Yu, "On demand classification of data streams," in *Proceedings of the tenth ACM SIGKDD International conference on Knowledge Discovery and Data mining*. ACM, 2004, pp. 503–508.

[21] S. Islam, J. Keung, K. Lee, and A. Liu, "Empirical prediction models for adaptive resource provisioning in the cloud," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 155–162, 2012.

[22] S. Meng, L. Liu, and V. Soundararajan, "Tide: achieving self-scaling in virtualized datacenter management middleware," in *Proceedings of the 11th International Middleware Conference Industrial track*. ACM, 2010, pp. 17–22.

[23] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: Data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[24] A. Downey and D. Feitelson, "The elusive goal of workload characterization," *ACM SIGMETRICS Performance Evaluation Review*, vol. 26, no. 4, pp. 14–29, 1999.

[25] B. Girod, R. Rabenstein, and A. Stenger, *Signals and systems*. John Wiley & Sons Inc, 2001.

[26] R. Takano, Y. Kodama, T. Kudoh, M. Matsuda, F. Okazaki, and Y. Ishikawa, "Realtime burstiness measurement," in *4th Intl. Workshop on Protocols for Fast Long-Distance Networks (PFLDnet2006)*, 2006.

[27] R. Gusella, "Characterizing the variability of arrival processes with indexes of dispersion," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 2, pp. 203–211, 1991.

[28] T. N. Minh, L. Wolters, and D. Epema, "A realistic integrated model of parallel system workloads," in *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*. IEEE, 2010, pp. 464–473.

[29] J. S. Richman and J. R. Moorman, "Physiological time-series analysis using approximate entropy and sample entropy," *American Journal of Physiology-Heart and Circulatory Physiology*, vol. 278, no. 6, pp. H2039–H2049, 2000.

[30] J. S. Richman, D. E. Lake, and J. R. Moorman, "Sample entropy," *Methods in enzymology*, vol. 384, pp. 172–184, 2004.

[31] M. Aboy, D. Cuesta-Frau, D. Austin, and P. Micó-Tormos, "Characterization of sample entropy in the context of biomedical signal analysis," in *29th Annual International Conference of the IEEE, Engineering in Medicine and Biology Society, 2007. EMBS 2007.* IEEE, 2007, pp. 5942–5945.

[32] The Rackspace Cloud. Accessed: May, 2013. [Online]. Available: http://www.rackspace.com/cloud

[33] CycleComputing. New CycleCloud HPC Cluster Is a Triple Threat. Accessed: May, 2013. [Online]. Available: http://blog.cyclecomputing.com/2011/09/new-cyclecloud-cluster-is-a-triple-threat-30000-cores-massive-spot-instances-grill-chef-monitoring-g.html

[34] Amazon Web Services. High performance computing (HPC) on AWS. Accessed: May, 2013. [Online]. Available: http://aws.amazon.com/hpc-applications/

[35] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. Epema, "The grid workloads archive," *Future Generation Computer Systems*, vol. 24, no. 7, pp. 672–686, 2008.

[36] IRCache. Access to trace files. Accessed: May, 2013. [Online]. Available: http://www.ircache.net/

[37] NCBI. PubMed. Accessed: May, 2013. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/

[38] M. Arlitt and T. Jin, "A workload characterization study of the 1998 world cup web site," *Network, IEEE*, vol. 14, no. 3, pp. 30–37, 2000.

[39] A. Rubin, *Statistics for evidence-based practice and evaluation*. Brooks/Cole, 2012.

[40] M. D. Costa, C.-K. Peng, and A. L. Goldberger, "Multiscale analysis of heart rate dynamics: Entropy and time irreversibility measures," *Cardiovascular Engineering*, vol. 8, no. 2, pp. 88–93, 2008.

[41] T. Chieu, A. Mohindra, A. Karve, and A. Segal, "Dynamic scaling of web applications in a virtualized cloud computing environment," in *IEEE International Conference on e-Business Engineering, 2009. ICEBE '09.*, oct. 2009, pp. 281 –286.

[42] P. Bodık, "Automating datacenter operations using machine learning," Ph.D. dissertation, University of California, 2010.

[43] D. Armstrong, D. Espling, J. Tordsson, K. Djemame, and E. Elmroth, "Runtime virtual machine recontextualization for clouds," in *Proceedings of the 18th international conference on Parallel processing workshops*, ser. Euro-Par'12. Berlin, Heidelberg: Springer-Verlag, 2013, pp. 567–576.

[44] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam, "Managing server energy and operational costs in hosting centers," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 33, no. 1. ACM, 2005, pp. 303–314.

[45] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. De Lara, M. Brudno, and M. Satyanarayanan, "Snowflock: Rapid virtual machine cloning for cloud computing," in *Proceedings of the 4th ACM European conference on Computer systems*. ACM, 2009, pp. 1–12.

[46] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[47] T. E. Oliphant, "Python for scientific computing," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 10–20, 2007.

[48] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[49] B. Schölkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization and beyond*. the MIT Press, 2002.

[50] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.

[51] G. Bakır, L. Bottou, and J. Weston, "Breaking SVM complexity with cross training," *Advances in neural information processing systems*, vol. 17, pp. 81–88, 2005.

[52] L. Kankanala and M. N. Murty, "Hybrid approaches for clustering," in *Pattern Recognition and Machine Intelligence*. Springer, 2007, pp. 25–32.

[53] M. Calzarossa and G. Serazzi, "Workload characterization: A survey," *Proceedings of the IEEE*, vol. 81, no. 8, pp. 1136–1150, 1993.

[54] H. W. Lilliefors, "On the Kolmogorov-Smirnov test for normality with mean and variance unknown," *Journal of the American Statistical Association*, vol. 62, no. 318, pp. 399–402, 1967.

[55] X. Kang, H. Zhang, G. Jiang, H. Chen, X. Meng, and K. Yoshihira, "Understanding internet video sharing site workload: A view from data center design," *Journal of Visual Communication and Image Representation*, vol. 21, no. 2, pp. 129–138, 2010.

[56] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Analysis and lessons from a publicly available Google cluster trace," *University of California, Berkeley, CA, Tech. Rep*, 2010.

[57] A. Mishra, J. Hellerstein, W. Cirne, and C. Das, "Towards characterizing cloud backend workloads: Insights from Google compute clusters," *ACM SIGMETRICS Performance Evaluation Review*, vol. 37, no. 4, pp. 34–41, 2010.

[58] Q. Zhang, J. L. Hellerstein, and R. Boutaba, "Characterizing task usage shapes in Googles compute clusters," in *Large Scale Distributed Systems and Middleware Workshop (LADIS11)*, 2011.

[59] Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "Towards understanding cloud performance tradeoffs using statistical workload analysis and replay," *University of California at Berkeley, Technical Report No. UCB/EECS-2010-81*, 2010.

[60] 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 *Proceedings of the Third ACM Symposium on Cloud Computing*, 2012, pp. 7:1–7:13.

[61] A. Iosup and D. Epema, "Grid computing workloads," *Internet Computing, IEEE*, vol. 15, no. 2, pp. 19–26, 2011.

[62] K. Yelick, S. Coghlan, B. Draney, and R. Canon, "The Magellan report on cloud computing for science," Technical report, US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Tech. Rep., 2011.

[63] A. Khan, X. Yan, S. Tao, and N. Anerousis, "Workload characterization and prediction in the cloud: A multiple time series approach," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 1287–1294.

[64] B. Viswanathan, A. Verma, and S. Dutta, "Cloudmap: Workload-aware placement in private heterogeneous clouds," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE, 2012, pp. 9–16.

[65] J. Beringer and E. Hüllermeier, "Online clustering of parallel data streams," *Data & Knowledge Engineering*, vol. 58, no. 2, pp. 180–204, 2006.

[66] P. Rodrigues, J. Gama, and J. Pedroso, "Hierarchical clustering of time-series data streams," *IEEE Transactions on Knowledge and Data Engineering,*, vol. 20, no. 5, pp. 615–627, 2008.

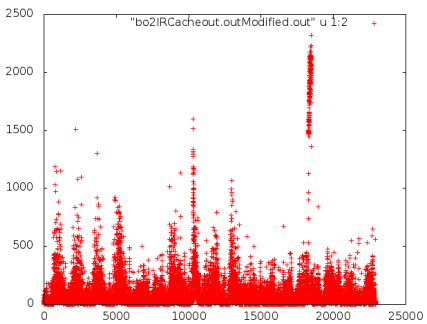
[67] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: Theory and practice," *IEEE Transactions on Knowledge and Data Engineering,*, vol. 15, no. 3, pp. 515–528, 2003.

[68] E. Keogh and S. Kasetty, "On the need for time series data mining benchmarks: A survey and empirical demonstration," in *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data mining*. ACM, 2002, pp. 102–111.

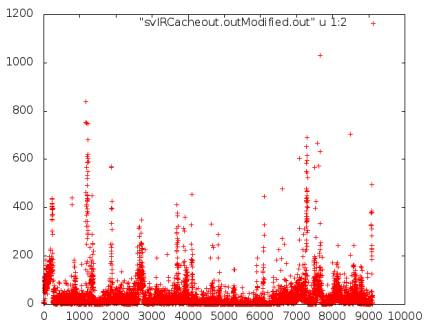
[69] N. R. Herbst, N. Huber, S. Kounev, and E. Amrehn, "Self-adaptive workload classification and forecasting for proactive resource provisioning," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '13. New York, NY, USA: ACM, 2013, pp. 187–198. [Online]. Available: http://doi.acm.org/10.1145/2479871.2479899

A Graphs for the Real Workloads

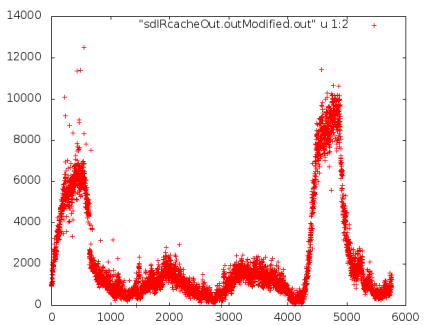
Figures 13, 14 and 15 show the plots of all 14 real workload traces used.



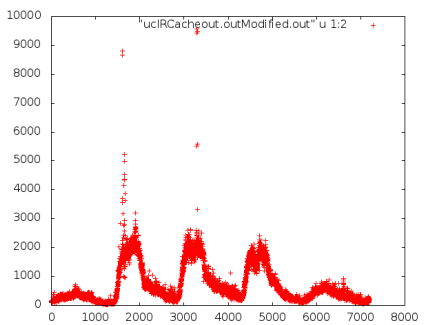
(a) Workload trace for IRCache service runnings ar at Boulder, Colorado (Bo).



(b) Workload trace for IRCache service running at Silicon Valley, California (SV).

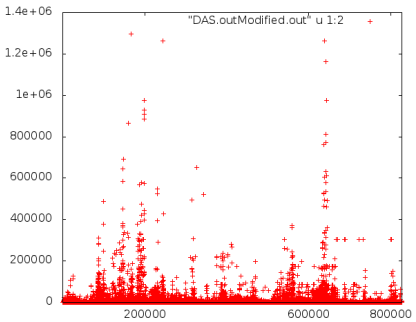


(c) Workload trace for IRCache service running at San Diego, California (SD).

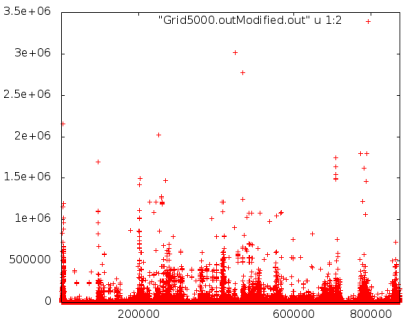


(d) Workload trace for IRCache service running at Urbana-Champaign, Illinois (UC).

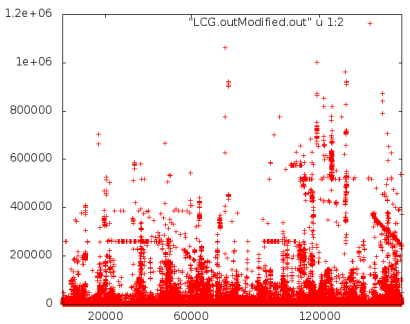
Figure 13: Workload traces of the different Caching Services.



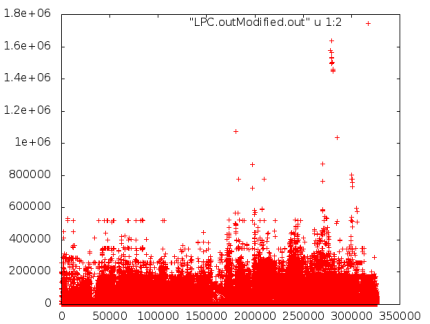
(a) Workload trace for the DAS workload.



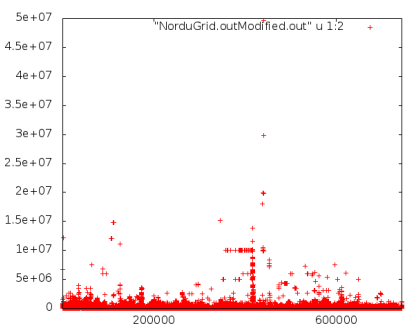
(b) Workload trace for Grid5000 workload.



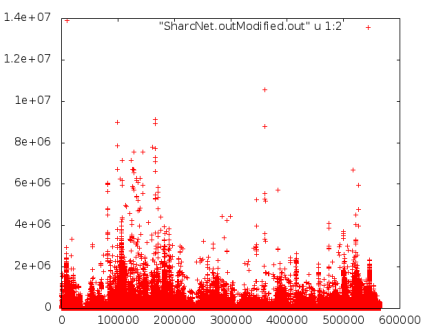
(c) Workload trace for the LCG workload.



(d) Workload trace for LPC workload.

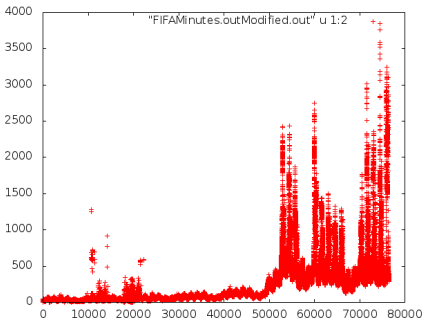


(e) Workload trace for the NorduGrid workload.

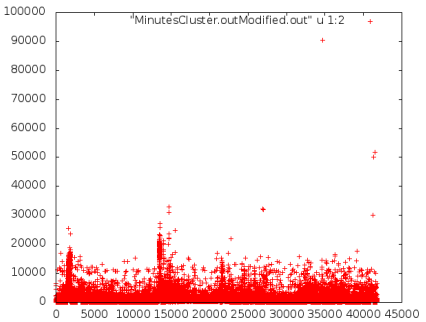


(f) Workload trace for SharcNet workload.

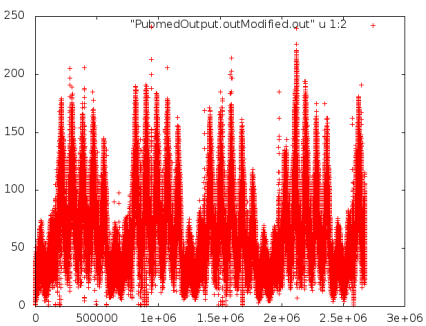
Figure 14: Workload traces of the Grid workloads analyzed.



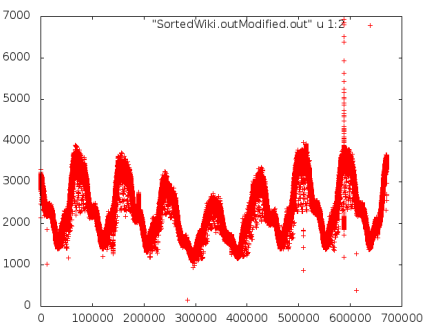
(a) Workload trace for the world cup workload.



(b) Workload trace for the Google Cluster workload.



(c) Workload trace for PubMed access traces.



(d) Workload trace for the Wikipedia workload.

Figure 15: Workload traces of web-hosting workloads and the Google cluster workload analyzed.