IJCC Editorial Board

Editors-in-Chief
Hemant Jain, University of Wisconsin–Milwaukee, USA
Rong Chang, IBM T.J. Watson Research Center, USA

Associate Editor-in-Chief
Bing Li, Wuhan University, China

Editorial Board
Danilo Ardagna, Politecnico di Milano, Italy
Janaka Balasooriya, Arizona State University, USA
Roger Barga, Microsoft Research, USA
Viraj Bhat, Yahoo, USA
Rajdeep Bhowmik, Cisco Systems, Inc., USA
Jiannong Cao, Hong Kong Polytechnic University, Hong Kong
Buqin Cao, Hunan University of Science and Technology, China
Keke Chen, Wright State University, USA
Haopeng Chen, Shanghai Jiao Tong University, China
Malolan Chetlur, IBM India, India
Alfredo Cuzzocrea, ICAR-CNR & University of Calabria, Italy
Ernesto Damiani, University of Milan, Italy
De Palma, University Joseph Fourier, France
Claude Godart, Nancy University and INRIA, France
Nils Gruschka, University of Applied Sciences, Germany
Paul Hofmann, Saffron Technology, USA
Ching-Hsien Hsu, Chung Hua University, Taiwan
Patrick Hung, University of Ontario Institute of Technology, Canada
Hai Jin, HUST, China
Li Kuang, Central South University, China
Grace Lin, Institute for Information Industry, Taiwan
Xumin Liu, Rochester Institute of Technology, USA
Shiyong Lu, Wayne State University, USA
J.P. Martin-Flatin, EPFL, Switzerland
Vijay Naik, IBM T.J. Watson Research Center, USA
Surya Nepal, Commonwealth Scientific and Industrial Research Organisation, Australia
Norbert Ritter, University of Hamburg, Germany
Josef Schiefer, Vienna University of Technology, Austria
Jun Shen, University of Wollongong, Australia
Weidong Shi, University of Houston, USA
Liuba Shrira, Brandeis University, USA
Kwang Mong Sim, University of Kent, UK
Wei Tan, IBM T.J. Watson Research Center, USA
Tao Tao, IBM T. J. Watson Research Center, USA
Kunal Verma, Accenture Technology Labs, USA
Raymond Wong, University of New South Wales & NICTA, Australia
Qi Yu, Rochester Institute of Technology, USA
Jia Zhang, Carnegie Mellon University – Silicon Valley, USA
Gong Zhang, Oracle Corporation, USA
Call for Articles
International Journal of Cloud Computing

Mission
Cloud Computing has become the *de facto* computing paradigm for Internet-scale service development, delivery, brokerage, and consumption in the era of Services Computing, fueling innovative business transformation and connected human society. 15 billion smart devices would be communicating dynamically over inter-connected clouds by 2015 as integral components of various industrial service ecosystems. The technical foundations of this trend include Service-Oriented Architecture (SOA), business & IT process automation, software-defined computing resources, elastic programming model & framework, and big data management and analytics. In terms of the delivered service capabilities, a cloud service could be, among other as-a-service types, an infrastructure service (managing compute, storage, and network resources), a platform service (provisioning generic or industry-specific programming API & runtime), a software application service (offering email-like ready-to-use application capabilities), a business process service (providing a managed process for, e.g., card payment), a mobile backend service (facilitating the integration between mobile apps and backend cloud storage and capabilities) or an Internet-of-things service (connecting smart machines with enablement capabilities for industrial clouds).

The International Journal of Cloud Computing (IJCC) aims to be a reputable resource providing leading technologies, development, ideas, and trends to an international readership of researchers and engineers in the field of Cloud Computing. IJCC only considers extended versions of conference papers published at reputable conferences such as IEEE International Conference of Cloud Computing.

Topics
The International Journal of Cloud Computing (IJCC) covers state-of-the-art technologies and best practices of Cloud Computing, as well as emerging standards and research topics which would define the future of Cloud Computing. Topics of interest include, but are not limited to, the following:

- ROI Model for Infrastructure, Platform, Application, Business, Social, Mobile, and IoT Clouds
- Cloud Computing Architectures and Cloud Solution Design Patterns
- Self-service Cloud Portal, Business Dashboard, and Operations Management Dashboard
- Autonomic Process and Workflow Management in Clouds
- Cloud Service Registration, Composition, Federation, Bridging, and Bursting
- Cloud Orchestration, Scheduling, Autoprovisioning, and Autoscaling
- Cloud Enablement in Storage, Data, Messaging, Streaming, Search, Analytics, and Visualization
- Software-Defined Resource Virtualization, Composition, and Management for Cloud
- Security, Privacy, Compliance, SLA, and Risk Management for Public, Private, and Hybrid Clouds
- Cloud Quality Monitoring, Service Level Management, and Business Service Management
- Cloud Reliability, Availability, Serviceability, Performance, and Disaster Recovery Management
- Cloud Asset, Configuration, Software Patch, License, and Capacity Management
- Cloud DevOps, Image Lifecycle Management, and Migration
- Cloud Solution Benchmarking, Modeling, and Analytics
- High Performance Computing and Scientific Computing in Cloud
- Cloudlet, Cloud Edge Server, Cloud Gateway, and IoT Cloud Devices
- Cloud Programming Model, Paradigm, and Framework
- Cloud Metering, Rating, and Accounting
- Innovative Cloud Applications and Experiences
- Green Cloud Computing and Cloud Data Center Modularization
- Economic Model and Business Consulting for Cloud Computing
Table of Contents

EDITOR-IN-CHIEF PREFACE
iv  Hemant Jain, University of Wisconsin–Milwaukee, USA
    Rong Chang, IBM T.J. Watson Research Center, USA

RESEARCH ARTICLES

1  QoS-Based Resource Allocation Framework for Multi-Domain SLA Management in Clouds
   Kuan Lu, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
   Ramin Yahyapour, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
   Philipp Wieder, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
   Constantinos Kotsokalis, IT & Media Center of Dortmund University of Technology, Germany
   Edwin Yaqub, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
   Ali Imran Jehangiri, Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany

13  Rapidly Alternating Bottlenecks: A Study of Two Cases in n-Tier Applications
    Qingyang Wang, Georgia Institute of Technology, USA
    Yasuhiko Kanemasa, Georgia Institute of Technology, USA
    Jack Li, Georgia Institute of Technology, USA
    Toshihiro Shimizu, Fujitsu Laboratories Ltd., Japan
    Masazumi Matsubara, Fujitsu Laboratories Ltd., Japan
    Motoyuki Kawaba, Fujitsu Laboratories Ltd., Japan
    Calton Pu, Georgia Institute of Technology

26  Cross Cloud Mapreduce: A Result Integrity Check Framework on Hybrid Clouds
    Yongzhi Wang, Florida international University, Miami, USA
    Jinping Wei, Florida international University, Miami, USA
    Mudhakar Srivatsa, IBM T.J. Watson Research Center, USA

40  Implementation and Empirical Assessment of a Web Application Cloud Deployment Tool
    Américo Sampaio, Universidade de Fortaleza (UNIFOR), Brazil
    Roberto Costa Jr, Universidade de Fortaleza (UNIFOR), E-NOVAR Soluções Tecnológicas, Brazil
    Nabor C. Mendonça, Universidade de Fortaleza (UNIFOR), Brazil
    Ricardo Hollanda Filho, E-NOVAR Soluções Tecnológicas, Brazil

53  A Queuing Model to Achieve Proper Elasticity for Cloud Cluster Jobs
    Khaled Salah, Khalifa University of Science, Technology and Research (KUSTAR), UAE

65  Call for Papers: IEEE CLOUD/ICWS/SCC/MS/BigData/SERVICES 2014
    Call for Articles: International Journal of Services Computing (IJSC)
    Call for Articles: International Journal of Big Data (IJBD)
Editor-in-Chief Preface:  
Cloud Management and Assessment

Hemant Jain  
University of Wisconsin–Milwaukee, USA

Rong Chang  
IBM T.J. Watson Research, USA

Welcome to the inaugural issue of International Journal of Cloud Computing (IJCC), the first open access on-line journal on cloud computing. The increasing importance of cloud computing is evidenced from the rapid adoption of this technology in businesses around the globe. The cloud computing is redefining the business model of various industries from video rental (Netflix is enabled by cloud) to small start-up companies (companies can be started with very little investment using cloud infrastructure). The potential of cloud computing is even more promising. The cloud computing combined with developments like internet of things can significantly change the life as we know today. However, to deliver on these promises and to prevent cloud computing from becoming a passing fad significant technical, economic, and business issues need to be addressed. IJCC is designed to be an important platform for disseminating high quality research on above issues in a timely manner and provide an ongoing platform for continuous discussion on research published in this journal. We aim to publish high quality research that addresses important technical challenges, economics of sustaining this environment, and business issues related to use of this technology including privacy and security concerns, legal protection, etc. We seek to publish original research articles, expanded version of papers presented at high quality conferences, key survey articles that summarizes the research done so far and identify important research issues, and some visionary articles. We will make every effort to publish articles in a timely manner.

This inaugural issue collects the extended version of five IEEE CLOUD 2013 articles in the general area of managing Cloud computing environment.

The first article titled, QOS-Based Resource Allocation Framework for Multi-Domain SLA Management in Clouds by Lu, Yahyapour, Wieder, Kotsokalis, Yaqub, and Jehangiri tackles the issue of downtime and service unavailability due to live migration. They present an OpenStack based implementation of a cloud resource allocation framework, named Generic SLA Manager, that supports downtime-aware VM selection and allocation during live migration of VMs. A simulation based evaluation of the proposed framework is reported as well.

The second article titled, "Rapidly Alternating Bottlenecks: A Study of Two Cases in N-Tier Applications" by Wang, Kanemasa, Li, Shimizu, Matsubara, Kawaba, and Pu reveals the importance of identifying the location of performance bottlenecks when scaling n-tier applications in computing clouds. They propose a bottleneck detection method that could be used to rapidly detect alternating bottlenecks. Experimental evaluation results for the proposed method are reported via two use cases.

The third article titled, "Cross Cloud MapReduce: A Result Integrity Check Framework on Hybrid Clouds" by Wang, Wei, and Srivatsa tackles the trust issue in adopting large-scale MapReduce on public clouds. They present a framework, named Cross Cloud MapReduce (CCMR), which overlays the MapReduce computation on a hybrid cloud where a master ensures correct result. A result integrity check scheme is also presented for accuracy and performance. Both theoretical and experimental analyses are reported.

The fourth article titled, “Implementation and Empirical Assessment of A Web Application Cloud Deployment Tool" by Sampaio, Costa, Mendonça, and Filho tackles the time-consuming issue in migrating applications to an IaaS cloud via application-specific VM images. They present an automated application deployment approach that requires less cataloged VM images. The approach can be supported via a tool, called TREXCLOUD, and an empirical evaluation of the tool is reported.
The fifth article titled, “A Queuing Model to Achieve Proper Elasticity for Cloud Cluster Jobs” by Salah tackles the issue of achieving proper elasticity for parallelized jobs running on cloud clusters. Based on finite queuing systems, the article presents an analytical model that can be used to determine the minimal number of cloud resources needed to satisfy the SLO requirements with constraints. Discrete Event Simulation is reported to verify the correctness of the proposed model.

We would like to thank the authors for their effort in delivering those five quality articles. We would also like to thank the reviewers as well as the Program Committee of IEEE CLOUD 2013 for their help with the review process. Finally, we are grateful for the effort Jia Zhang and Liang-Jie Zhang made in giving birth to this inaugural issue of International Journal of Cloud Computing (IJCC).

About the Editors-in-Chief

**Dr. Hemant Jain** is the Interim Director of Biomedical and Health Informatics Research Institute, Roger L. Fitzsimonds Distinguished Scholar and Professor of Information Technology Management at University of Wisconsin–Milwaukee. Dr. Jain specializes in information system agility through web services, service oriented architecture and component based development. His current interests include development of systems to support real time enterprises which have situational awareness, can quickly sense-and-respond to opportunities and threats, and can track-and-trace important items. He is also working on issues related to providing quick access to relevant knowledge for cancer treatment and to providing medical services through a virtual world. Dr. Jain is an expert in architecture design, database management and data warehousing. He teaches courses in database management, IT infrastructure design and management, and process management using SAP. Dr. Jain was the Associate Editor-in-Chief of IEEE Transactions on Services Computing and is Associate Editor of Journal of AIS, the flagship journal of the Association of Information Systems.

**Dr. Rong N. Chang** is Manager & Research Staff Member at the IBM T.J. Watson Research Center. He received his Ph.D. degree in computer science & engineering from the University of Michigan at Ann Arbor in 1990 and his B.S. degree in computer engineering with honors from the National Chiao Tung University in Taiwan in 1982. Before joining IBM in 1993, he was with Bellcore researching on B-ISDN realization. He is a holder of the ITIL Foundation Certificate in IT Services Management. His accomplishments at IBM include the completion of a Micro MBA Program, one IEEE Best Paper Award, and many IBM awards, including four corporate-level Outstanding Technical Achievement Awards and six division-level accomplishments. He is an Associate Editor of the IEEE Transactions on Services Computing and the International Journal of Services Computing. He has chaired many conferences & workshops in cloud computing and Internet-enabled distributed services and applications. He is an ACM Distinguished Member/Engineer, a Senior Member of IEEE, and a member of Eta Kappa Nu and Tau Beta Pi honor societies.
QoS-BASED RESOURCE ALLOCATION FRAMEWORK FOR MULTI-DOMAIN SLA MANAGEMENT IN CLOUDS

Kuan Lu\textsuperscript{a}, Ramin Yahyapour\textsuperscript{a}, Philipp Wieder\textsuperscript{a}, Constantinos Kotsokalis\textsuperscript{b}, Edwin Yaqub\textsuperscript{a}, Ali Imran Jehangiri\textsuperscript{a}

\textsuperscript{a}Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen, Germany
\textsuperscript{b}IT & Media Center of Dortmund University of Technology, Germany

\{kuan.lu, ramin.yahyapour, philipp.wieder, edwinyaqub, ali.jehangiri\}@gwdg.de
constantinos.kotsokalis@tu-dortmund.de

Abstract

In clouds, current virtualization technologies of IaaS enable the live migration of running VMs to achieve load balancing, fault-tolerance and hardware consolidation in data centers. However, the downtime / service unavailability due to live migration may be substantial with relevance to the customers’ expectations on responsiveness, as the latter are declared in established SLAs, which define all relevant aspects of the services between service provider and customer. Moreover, the service unavailability may cause significant (potentially exponential) SLA violation penalties to its associated higher-level domains (e.g., PaaS and SaaS). Therefore, in order to deliver high availability service, VM live migration should be arranged and managed carefully. In this paper, we present the OpenStack version of Generic SLA Manager, alongside its strategies for VM selection and allocation during live migration of VMs. Based on the proposed autonomous SLA violation-filtering framework, we simulate a use case where IaaS (OpenStack-SLAM) and PaaS (OpenShift) are combined; and assess performance and efficiency of the aforementioned VM placement strategies, when a multi-domain SLA pricing & penalty model is involved. We find that our proposal is efficient in managing trade-offs between the operational objectives of service providers (including financial considerations) and the customers’ expected QoS requirements.

Keywords: Live migration, Virtual machines, IaaS, PaaS, Availability, SLA pricing, SLA penalties, SLA violation, Resource allocation

1. INTRODUCTION

In Infrastructure-as-a-Service (IaaS), through virtualization technologies (e.g., VMWare (2013), Xen (2013)), physical resources of data centers can be partitioned into flexible and scalable virtual computing units, namely Virtual Machines (VMs). However, large-scale data centers introduce large power consumption costs. Thus, an efficient technique that dynamically reconfigures the IT infrastructure to reduce the total power consumption becomes necessary. As such, VM consolidation emerges to execute the VMs on as few servers as possible, to concentrate the workloads and limit the number of physical servers powered on.

VM consolidation is usually treated as an objective of the Service Provider (SP). From customer’s perspective, an automated negotiation may be used to accommodate heterogeneous requirements against an SP’s capabilities and acceptable usage terms. The result of such a negotiation is a Service Level Agreement (SLA), an electronic contract that establishes all relevant aspects of the service. During the SLA negotiation, all terms must be evaluated before a final agreement is reached. In order to commit to the requested Quality-of-Service (QoS) terms (e.g., service performance and availability), SPs have to assess their own resource management strategies so as to trade-off profit making with a guaranteed delivery of service(s) and avoid penalties in case the agreement is violated at runtime. An aggressive consolidation of VMs however may lead to performance degradation when the service faces increasing demand, resulting in unexpected rise of resource utilization.

By means of VM live migration technologies, both VM consolidation and service performance can be coordinated and balanced. However, research by Salfner et al. (2012) revealed that short downtimes of services are unavoidable during VM live migration due to the overheads of moving the running VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and it is possible that the customer’s expectations on responsiveness are not met. Moreover, it might bring exponential service violation penalties to its associated domains (e.g., Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS)). By way of an example, the solutions from Beloglazov et al. (2011) provide service availability from 99.70% to 99.93%. According to the research by Akoush et al. (2010), for e-commerce and other industrial use cases, an availability value below 99.90% is usually considered unacceptable. In order to provide high availability, to avoid service violation and the subsequent penalties, the number of VM live migrations should be monitored and controlled.

In this paper, we present an OpenStack version of the Generic SLA Manager (GSLAM), which could be potentially used into project PaaSage (2012). We apply this software system to combine IaaS (OpenStack-SLAM) and PaaS (OpenShift (2013)) in a use case that features multi-domain SLA management. Via the introduction of a pricing
and penalty model that considers such multi-domain scenarios, we apply our resource allocation strategies for VM selection and allocation during live migration. Based on the proposed autonomous SLA violation-filtering framework, we simulate the full scenario (using the CloudSim platform from Calheiros et al. (2011)) and illustrate the suitability of our proposal for the efficient management of VM live migration. Thereby, the agreed service availability is not violated without paying the extra penalties and a trade-off between the SP’s objectives and the customers’ expected QoS requirements can also be achieved successfully.

The remainder of this paper is structured as follows. In Section 2, we will discuss the related work. Section 3 will present the OpenStack version of the GSLAM. In Section 4, a formal model of SLA pricing and penalty will be provided. Section 5 introduces our autonomous SLA violation-filtering strategy. In Section 6, a description of our approach to process resource allocation and how it achieves important resource management objectives will be given. Then, in Section 7, we will validate our mechanisms performing discrete event simulations. Finally, we conclude the paper in Section 8.

2. RELATED WORK

Many publications, such as Kertesz et al. (2008), Stantchev and Schröpfer (2009) and Brandic et al. (2009), discuss the topic of SLA management for IT clouds, but most of them are looking at it from a conceptual and architectural point of view. To the best of our knowledge, there is no prior work for SLA management in OpenStack that considers multi-domain pricing and penalties, and that can be further applied to develop QoS-aware resource allocation strategies.

Many works, such as Kosinski et al. (2008), Becker et al. (2008) and Rana et al. (2008), also discuss SLA violations. However, there are few to model the consequences of the violations, namely, SLA penalties. Currently, although some approaches describe penalties, they do not satisfy all of the following requirements for formulating complex penalty expressions in a single unambiguous model:

- Able to present the chain effect on violation penalties among multi-domains in IaaS.
- Full flexibility regarding QoS levels agreed and/or achieved, without being constrained (e.g., by pre-specified classes of service).
- Openness and applicability to different domains, without dependence on specific languages or taxonomies.

As regards VM live migration, there are certain approaches already widely utilized. Through shared storage such as iSCSI Storage Area Network (SAN) or Network Attached Storage (NAS), the process of live migration is reduced to only copying memory state and CPU registers from source host to destination, without transferring the whole VM. In contrast to the pure stop-and-copy strategy of offline VM migration, live migration fine-tunes the migration into several rounds of pre-copying and one last round of stop-and-copy with much less downtime. Nevertheless, based on the impact of VM migration from Hermenier (2009), there are still a few issues to address. Firstly, the more iteration in the pre-copying phase, the less data need to be transferred in the final stop-and-copy phase, and the downtime becomes less. However, as Akoush et al. (2010) explained that the short downtime comes at the cost of longer total migration time, which leads to significant influence on service performance. Secondly, the downtime will never be eliminated, because many workloads usually include some memory pages that are updated frequently, named Writable Working Sets (WWS). Clearly, it is wise not to maintain it until the last phase. Moreover, for stop-and-copy, Salfner et al. (2012) noted that the performance of VM live migration is affected by many factors, such as workload type, hypervisor type and so on.

Thus, no visible downtime is only an ideal goal. The number of migrations should be controlled, as it has impact on the performance of the other running VMs in the source and destination hosts, which is mainly proportional to the amount of memory allocated to the VM. In this paper, the VM performance violation happens when the VM experiences the CPU utilization of 100%. And the performance degradation happens during the VM live migration.

Service availability is one of the most important QoS metrics in IT clouds. Machado and Stillerm (2011) outlined that the service availability guaranteed by three large cloud providers (Amazon, Google and Rackspace Cloud) is more than 99.90% in order to obtain good reputation in today’s competitive market. Therefore, in upcoming sections we propose to provide a basic service availability of 99.90% and an advanced availability of 99.99% in the scope of VM live migration. The later one implies that the SP has to pay special attention (e.g., extra resources) on the service in order to avoid SLA violations.

For VM consolidation, Beloglazov et al. (2012), Beloglazov and Rajkumar (2011) and Corradi et al. (2012) mainly discussed:

- Resource management, either in simulation or some prototype implementations within the common cloud middleware. And no SLA management is introduced.
- Using either artificial workloads or partial historical information from the workloads in various projects.
- Using VM live migration to leverage the consolidation and service performance; however, the migration was misused without carefully taking service availability into consideration.

Therefore, in this paper, our goals are the following:
A proof-of-concept prototype implementation of OpenStack SLA management (IaaS), which aims to be connected with PaaS and SaaS layers by providing SLA lifecycle, service customization and automatic scalability.

By using the workloads information at GWDG, our strategies are simulated in CloudSim and compared with others in several aspects.

The influence of SLA chain penalties in multi-domains is introduced into VM consolidation. Also, availability oriented VM allocation strategies that control VM live migration and leverage the objectives between the SP and the customer.

3. **OpenStack SLA Management Framework**

Based on the GSLAM in our previous project SLA@SOI (2009), an OpenStack SLAM is presented. The GSLAM (Figure 1) provides a generic architecture that can be used across different domains and use cases to manage the entire SLA life cycle, including activities such as SLAs modeling, negotiation, provisioning, resources outsourcing, monitoring and adjustment. Gonzalez et al. (2011) argued that the GSLAM is an orchestrator of the generic components. It could provide interoperability and separates the SLAMs from specific representations of an SLA or SLA template. Namely, any domain specific scenario is able to contact with GSLAM to request a new set of generic components, which encapsulate all the basic functionalities for handling SLAs. Moreover, the corresponding planning and optimization component (POC) and provisioning and adjustment component (PAC) will be partially re-implemented so as to enrich the functionalities subject to it own domain. Through OpenStack Nova API Woorea (2012), the GSLAM is able to implement its corresponding Infrastructure Service Manager (ISM) by:

- Querying the status of infrastructure during SLA negotiation, based on which the SP is able to generate the corresponding offer / counter-offer for the customer.
- Providing SLA terms (i.e. pricing, penalty, availability and performance) monitoring mechanism together with OpenStack.
- Deploying the VM allocation strategies within OpenStack Nova so as to maximize the profit and minimize SLA violation as well as energy consumption.
- Creating, customizing and deploying the agreed services.
- Reconfiguring and removing the services on demand.

PaaSage (2012) aims to deliver an open and integrated PaaS for different (e.g., industrial, e-science and public sector) use cases to support model-based lifecycle management of cloud applications.

Using OpenShift (2013), one of the most popular PaaS implementations, it could auto-scale its cloud PaaS framework for Java, Perl, PHP, Python and delivered in a shared-hosting model. PaaS permits many applications offered by multiple development teams to co-exist on the same set of hosts in a safe and reliable fashion. In addition to that, the platform offers a variety of opportunities for multi-tenant deployments. Thus, an application that is intended to work for a single organizational unit can also be deployed in such a manner that many organizations or end-users are able to use it. Therefore, end-users benefit from application management (instead of VM level management) while application providers can bring their applications into the PaaS cloud with minimal effort.

![Figure 1. Integration of the Generic SLA Manager and OpenStack](http://hipore.com/ijcc)
As Figure 2 illustrates, OpenShift can be treated as a customer of the OpenStack-SLAM asking for infrastructure support. Our target is to automatically scale up and down the virtual resources (i.e. VMs) for the PaaS domain as needed. The SLAM not only provides the VMs, it is also able to customize the VMs using pre-defined scripts so as to deliver the “OpenShift-ready” VMs in one click. Specifically, let us suppose a PaaS SP starts SLA negotiation with an IaaS SP. When the IaaS SP has sufficient resources, a counter-offer will be sent back to the PaaS SP with a timeout. Once the offer is accepted within the timeout, then the VM will be created, and the SLAM will automatically log into the VM by matching the public key pair with its private key. Finally, the pre-installed scripts will be executed on the target VM. The execution includes three steps, namely:

- Installing the PaaS broker-specific packages.
- Assigning a public IP for the VM.
- Configuration of Mongo database / ActiveMQ / other components associated with the PaaS broker in OpenShift.

Thus, the VM can be recognized and controlled by the PaaS broker. Similarly, if the host is detected as under-loaded, the infrastructure can be easily scaled-down by removing the VMs. Here, PaaS and IaaS layers are technically interconnected. In Section 4, we will see how are they mutually influenced in term of economical aspect during SLA violation.

![Figure 2. Sequence diagram for the negotiation between PaaS and the OpenStack SLA Manager](image)

4. MODELING OF SLA PRICING AND PENALTY

As Lu et al. (2011) argued that IaaS SPs are able to compute the minimum implementation costs as part of price quotations towards customers, in order to remain competitive. At the same time, profit and SLA violation probability constraints are used to decide whether the problem can be satisfied at all, and what is the decision space based on which implementation costs can be calculated. Furthermore, outsourcing via subcontracts was included as part of the decision process, to achieve additional profit but also to sustain customers when local resources are not sufficient.

Here, we assume that the corresponding planning and optimization strategies implemented by Lu et al. (2011) are fully applied but not explained in detail in order to keep the paper reasonably concise. In brief, the objective of the approach is to minimize implementation and outsourcing costs for reasons of competitiveness, while respecting business policies for profit and risk. A greedy algorithm for outsourcing was implemented, using cost and subcontractor reputation as selection criteria; and local resource configurations as a constraint satisfaction problem for acceptable profit and failure risks. Thus, it becomes possible to provide educated price quotes to customers and establish safe electronic contracts automatically. Discarding either local resource provisioning, or outsourcing, models
efficiently the specialized cases of infrastructure resellers and isolated infrastructure providers respectively.

Therefore, let us assume an IaaS service $i$, and an SLA that governs consumption of this service by a certain customer. We have:

$$ C^i = C^{i\text{imple}} + Pr^i \quad (1) $$

$$ C^{i\text{imple}} = C^i + C^{i\text{E}} \quad (2) $$

$$ C^i = C^{i\text{energy}} + C^{i\text{utility}} \quad (3) $$

where the cost $C^i$ of service $i$ is the sum of internal cost (i.e. internally utilized resources, energy cost) and external cost (i.e. sub-contracted resources) as well as profit.

In PaaS, a container includes a set of resources that allows users to run their applications. By delivering such a computing platform (e.g., operating system, program execution environment), many containers can be run simultaneously on one VM (see Figure 3). We assume on each VM there are $n$ containers.

Therefore, the cost of each PaaS service $p$ is:

$$ C^p = \frac{C^i}{n} + Pr^p \quad (4) $$

SaaS developers can implement and deploy their applications on a cloud platform (e.g., container) without the cost and complexity of buying and managing the underlying hardware and software layers. Similarly, we assume on each container $m$ applications are allocated. Then, the cost of a SaaS service $s$ is defined as following:

$$ C^s = \frac{C^p}{m} + Pr^s \quad (5) $$

$C^i$ and $C^p$ apply only based on the assumption that the payment has no implementation costs other than the infrastructure and platform environment for service execution.

Meanwhile, an SLA should also contain a set of penalty clauses specifying the responsibilities in case the SPs fail to deliver the pre-agreed QoS terms. Thus, we will use a variation of penalty model from Kotsokalis et al. (2011) as outlined in Equation 6: service $s$ is defined as following:

$$ R_s = \sum_k GW_k \cdot VR_k \quad (6) $$

where $R_s$ is the penalty ratio associated with the cost of service $x$, where $GW_k$ is the weight of one specific guarantee being violated, for this specific combination of guarantees. This value may be arbitrarily high. It allows the negotiating customer to express the importance of honoring certain guarantees in this penalty function. $VR_k$ is the violation ratio: the relationship between achieved quality and planned quality. It indicates how far the offered quality has drifted from the agreed quality of a specific service parameter.

Therefore, the penalty of IaaS service $i$ is:

$$ P^i(QoS_1^i,...,QoS_t^i) = C^i \cdot R_i \quad (7) $$

The penalty of all PaaS services $p$ is:

$$ n \cdot P^p(QoS_1^p,...,QoS_t^p) = n \cdot C^p \cdot R_p \quad (8) $$

By applying Equation 4, we have:

$$ n \cdot C^p \cdot R_p = (C^i + n \cdot Pr^p) \cdot R_p \quad (9) $$

The penalty of all SaaS services $s$ is:

$$ m \cdot n \cdot P^s(QoS_1^s,...,QoS_t^s) = m \cdot n \cdot C^s \cdot R_s \quad (10) $$
By applying Equation 5, we have:

\[ m \cdot n \cdot C^s \cdot R_i = (C' + n \cdot Pr^s + n \cdot m \cdot Pr^r) \cdot R_i \]  \hspace{1cm} (11)

The violation of some QoS terms on the IaaS layer will automatically affect the other domains. For example, unavailability of a VM will unquestionably enforce its inner PaaS and SaaS services to be unavailable. For all these QoS terms in the three layers, we have \( R_i = R_p = R_s = R \). Thus, the extra penalties of the PaaS layer comparing with the IaaS layer is:

\[ (9) - (7) = n \cdot Pr^p \cdot r \] \hspace{1cm} (12)

Similarly, the extra penalties of SaaS layer comparing with IaaS layer is:

\[ (11) - (7) = (n \cdot Pr^p + m \cdot n \cdot Pr^s) \cdot r \] \hspace{1cm} (13)

Hence, a slight availability violation in IaaS will lead to exponential influences on its associated domains (PaaS and SaaS). An IaaS SP, in order to compliant with the SLAs, has to make optimal reaction and adjustment while the service is running.

5. AUTONOMOUS SLA VIOLATION MONITORING AND FILTERING

Violation can be seen as one of states during the whole lifecycle of an SLA (as Figure 4 illustrated). As mentioned in Section 3, negotiation is of fundamental interest in service-oriented systems. Two or more parties negotiate SLAs based on their individual goals. Specifically, a user may initiate the SLA negotiation by sending pre-customized SLA template (S0). An SLA template is a document that describes a set of options for each domain specific service. The negotiation may last several rounds (S1) to negotiate offers or counter-offers, and both of them may require certain iterations. At some point, a final agreement is reached (S3). The negotiation could also be terminated (S2) as long as the timeout expires or one of two negotiating parties withdraws. Re-negotiating an SLA could modify its corresponding running service (S4). Furthermore, when an SLA is completely fulfilled, it will be automatically terminated (S6).

Sometimes, a potential SLA violation could be avoided in execution state (S3). Adding extra resources on the running service, indicating the provider’s dynamic policy with regard to the additional measures to take, is able to safeguard the respective guaranteed quality of a certain specific SLA so that a violation (S7) could be resolved in execution state (S8). Thereby, based on the state machine in Figure 4, an autonomous SLA violation-filtering framework is presented. In Figure 5, three modules are classified and defined individually into three layers.

Specifically, a common SLA manager module creates and maintains one or more SLA modules. Similarly, each SLA module creates and maintains one or more QoS property modules. In this framework, each module has exactly one supervisor, which is the module that creates it. If one module does not have the means for dealing with a certain situation, it will send a corresponding event-driven failure message (e.g., exception) to its supervisor, asking for help. The recursive structure then allows handling failure at the right level. Everything in this framework will be designed to work in a complete distributed environment. As thus, all interactions of modules are pure message passing and asynchronous. Therefore, the SLA violation could be efficiently eliminated.
Consequently, each module will strive to resolve the (potential) violation on its own layer. At very beginning, when a certain violation (e.g., service performance issue) is detected by property module, it tries to fix the violation by restarting the service or adding extra resources on the running service and without altering the content of SLA. In this case, customer does not feel any change and the running SLA is still valid.

Otherwise, it will send an exception message to its supervisor (SLA module). In SLA module, the corresponding exception handlers have already been predefined with all possible solutions. Likewise, if SLA module cannot find an alternative solution, it will forward the message to its supervisor (SLA manager module). SLA manager tries to initiate a renegotiation with the customer so as to establish a new SLA without paying the penalty ($\delta$). And there could also be several rounds of renegotiation within a certain time span.

6. SLA-BASED RESOURCE ALLOCATION AND PROVISIONING

Avoiding SLA violation by initializing an SLA renegotiation is mainly dependent on customer’s willing. In this paper, we are more focusing on how to fix the violation on the first two layers. Through VM live migration, both VM consolidation and service performance can be coordinated. Nevertheless, short downtimes of service migration are unavoidable due to the overheads of moving the VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and this could also be the main cause of the chain effect on penalties between domains. Here, the term \textit{downtime} is used to refer to periods when a service is unavailable and fails to provide or perform its primary function to customers. The downtime can be further classified to be planned and unplanned. Since unplanned downtime, e.g., failure of the system, is complicated and uncertain in a simulation environment, in our work, we only consider the planned downtime for evaluating the service availability. The downtime that is introduced by VM live migration is a kind of planned downtime. Thus, the availability is formulated as below:

$$\text{availability} = \frac{T_u}{T_u + T_d} \times 100$$  \hspace{1cm} (14)

where $T_u$ is service uptime and $T_d$ is service downtime.

When a user is consuming a service, the service provider has to ensure that the service availability is in accordance with the one expressed in the SLA, in order to avoid the potential penalties due to SLA violations. In this paper, we are focusing on how to manage the number of live migrations so as to control availability according to the established SLA during resource allocation. The optimization of resource allocation problem in a data center can be executed in two steps: initial selection of VMs that need to be migrated, then the chosen VMs will be placed on the hosts using a VM allocation algorithm.

6.1 VM SELECTION
In Algorithm 1, the input value is the requested availability of customer and the output value is final selected VM that will be migrated. Firstly, all the migratable VMs will be selected by removing the VMs that are already in migration. Thus, the selected VMs will be sorted in descending order of their current service availabilities. Then, the availability of each VM in the sorted VMs list will be re-calculated to check whether the availability is still greater than the requested availability or not, when the VM is migrated. Finally, if such a VM can be found, then we will migrate it and update the downtime record of this VM. Otherwise, no VM will be migrated. The complexity of the selection part of algorithm is $O(n)$, $n$ being the number of migratable VMs.

Algorithm 1 VM Selection-AV

Input: reqAvailability
Output: selectedVM

migratableVms ← getMigratableVms()
totalTime ← 86400 // 24 hours in seconds
vmSize ← migratableVms.getSize()
vms, downtime, totalDowntime, availability ← NULL

sortByAvailability(migratableVms) // descending order
for v ← 1 to vmSize do
  vm ← migratableVms[i]
  downtime ← vm.downtimeEstimator()
  totalDowntime ← vm.totalDowntime
  availability ← 1 - (totalDowntime + downtime) / totalTime
  if availability > reqAvailability then
    selectedVM ← vm
    break
  else
    continue
  end if
end for

selectedVM.updateDowntime()
return selectedVM

6.2 VM Allocation

In Algorithm 2, the inputs are optimal host utilities and selected VMs; the outputs are the hosts to where each selected VM will be migrated. First of all, the over-loaded host(s) and the host(s), which is (are) going to be over-loaded after allocating the migrated VM, will not be considered. Then, a host, the utility of which is the closest to optimalHostUtility, will be selected. Here, the optimalHostUtility is not a fixed value and will be discussed in Section 7. We want to find the relationship between the utilization of target allocation host and the value of QoS terms.

Specifically, by considering the service availability constraint, our goal is to allocate the VM to a host that provides the least increase of power consumption and service performance violation due to this allocation. The complexity of the allocation part of algorithm is $O(n)$, $n$ is the number of hosts.

Algorithm 2 VM Allocation-AVL

Input: optimalHostUtility, selectedVM
Output: allocatedHost

minimalDiff ← Double.MAX_VALUE
hosts ← getHostList(); host, hostUtility, diff ← NULL
for h ← 1 to hosts.size do
  host ← hosts[i]
  if excludedHosts.contains(host) then
    continue
  end if
  hostUtility!=0
  if host.isSuitableForVm(selectedVM) then
    continue
  end if
  if hostUtility > host.getUtilizationOfCpu() then
    if hostUtility > optimalHostUtility + minimalDiff then
      allocatedHost ← host
    end if
  end if
end for

return allocatedHost

6.3 VM Live Migration Downtime Estimator

As discussed in Section 2, the overall duration and short downtime that are introduced by VM live migration are essential properties while implementing service availability in an SLA. In this section, we introduce a VM live migration downtime estimator into CloudSim. As modeled by Akoush et al. (2010) and Salfner et al. (2012), using migration bounds, the downtime of VM live migration is defined in lower and upper bounds as follows:

$$mig\_overhead \leq t_d \leq mig\_overhead + \frac{VMsize}{LinkSpeed}$$

In order to estimate better the downtime value, the authors summarized four main factors that affect the downtime, namely: available link speed, average page dirty rate, VM memory size and migration overheads. The link speed and page dirty rate are proportional to the access traffic of the server applications in a day. The access traffic reaches the highest value at noon and in the morning and late night it will reach the lowest value. Therefore, we assume the probability of determining live migration downtime is consistent with normal distribution as following:

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma^2} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where $0 < x < 24$ (hour), expected value is 12 and variance is 1.9, which means at 12 o’clock the server application usually reaches the highest access traffic. For instance, a VM loading server application workloads has 1024 MB memory and 1 Gbps migration link. As such, the lower and
upper bounds of migration are around 314 ms and 9497.8 ms respectively.

7. CASE STUDIES AND DATA ANALYSIS

We choose CloudSim as our simulation platform in order to validate the approaches in Sections 6.1 and 6.2. As it was explained by Beloglazov et al. (2012), CloudSim is able to model and trace the energy consumption and SLA performance with automatic host over/under-loading detection, which reduce the preparation of our simulation work mainly to focusing on SLA availability based VM selection and allocation strategies.

Based on the cloud infrastructure and workloads at GWDG in Germany, a virtual data center is simulated, including 120 virtual hosts. 81 VMs (2 Euro for each) are created, in which 244 containers (3 containers for each VM)
are generated with the corresponding 732 application workloads (3 workloads for each container). 1 kw/h of electricity costs 0.2 Euro. Each container and application workload will make a 2-Euro profit respectively. The whole simulation time is 24 hours. Once the cloud environment in CloudSim is setup, it will automatically allocate the workloads into the VMs. The interval of utilization measurements is 5 minutes. As it was already defined in CloudSim, SLA violation Time per Active Host (SLATAH) is the percentage of time, during which active hosts have experienced the CPU utilization of 100%. And Performance Degradation due to Migrations (PDM) is the overall performance degradation by VMs due to migrations. SLA performance violation, modeled by Beloglazov and Rajkumar (2011) is as following:

\[ \text{SLAV} = \text{SLATAH} \cdot \text{PDM} \] (17)

By applying the Algorithm 1 into the CloudSim, we want to test how the SLA availability constraint affects energy consumption and SLA performance. Therefore, the difference between 99.90% and 99.99% is equally divided into 40 intervals, and we set each interval as the reqAvailability of Algorithm 1. The results are illustrated in Figure 6 (a-e). In Figure 6 (a), when we set SLA availability constraint as 100%, meaning VM live migration is not applied, 88 of 120 hosts are always in active, thus all VMs have sufficient resources for their applications without SLA performance violation. However, this leads to huge energy consumption (around 85 kw/h, see upper-right corner of the figure). Once VM live migration is applied, VM consolidation is always considered in order to save on energy consumption. Thus, for example, at 99.99% constraint, 50 hosts are turned into energy saving mode and the energy consumption decreases dramatically (around 52 kw/h). On the contrary, when the availability constraint is not strict (e.g., 99.90%), the SLATAH value (Figure 6 (c)) is relatively low, because as long as an over-load situation is detected, it will be resolved by migrating the VM(s) to the other host(s). Nevertheless, the corresponding PDM value (Figure 6 (b)) is very high, because:

- In under-loaded situations, if all the VMs on this host can be migrated to other host(s), the number of VM migrations is increased. However, this leads to "circular flow" for some VMs, meaning they are migrated between hosts back and forth. Hence, although servers shutting down count increases, energy is actually not saved.
- Extra migrations will definitely lead to performance degradation, therefore the PDM value is also increased.

By applying Equation 17, the final SLA performance violation is as illustrated in Figure 6 (d). From Figure 6 (e), it is apparent that the server shutdown time decreases with increases in service availability. Therefore, from the first experiment, we find that VM live migration can efficiently reduce the energy cost of data center. However, the number of migrations should be balanced in order to achieve the desired service availability and performance requirements.

By applying Algorithm 2 into CloudSim, we strive to find which host will be the optimal destination for allocating the migrated VM(s). Similarly, the host utility between 0% and 100% is divided equally into 200 intervals, and we set each interval value as the optimalHostUtility value into Algorithm 2. By default, 80% is the threshold utility. As illustrated in Figure 6 (f-g), migrating a VM to a host whose utility is the closest to the threshold utility, will lead to the least energy consumption and SLA performance violation. Because otherwise, some VMs from the source host will be migrated back and forth until one of them cannot be moved anymore. This not only increases the number of VM migrations (see Figure 6 (h)) unnecessarily and blocks reasonable future migrations due to the availability constraint, but also introduces further energy consumption. From Figure 6 (i), we can see that the server shutdown time also decreases with increases in target host utilization. As such, we can directly replace the optimalHostUtility with thresholdHostUtility in Algorithm 2.

In our final experiment, by taking the results of the experiments above, we set 99.99% as target service availability and 80% as target host CPU threshold utilization during VM live migration. Using our workloads, we compare the VM allocation algorithms (THR, IQR, MAD, LR and LRR) and the VM selection algorithms (MMT, RS and MC) of CloudSim with our approach, namely AVL/AV. Consequently, the results (see Figure 7 (a)) show that on average the current resource allocation algorithms in CloudSim are able to deliver service availability for each VM from around 99.7% to 99.93%. In our approach, the service availability is always kept as 99.99%, the SLA violation is 0.00155% and the energy consumption is 54.22 kw/h. Whereas, the other approaches in CloudSim introduce around 37 to 51 kw/h energy consumption and higher SLA performance violation (Figure 7 (b)). Although AVL/AV introduces slightly more energy than the other approaches, using VM consolidation in general still saves much more energy (85 kw/h in non-consolidation case).

The penalty model in Section 4 applies when SLA violation happen. The VM allocation algorithms, using THR, IQR or MAD as VM selection strategy, will lead to availability lower than 99.90%. In this case, the penalty of all three approaches will be increased up to the full cost. Thereby, we selected all the representative approaches to compare with ours in order to find the penalty chain influences in three cloud domains. In Figure 7 (c), our approach introduces the fewest penalties. Whereas by using other approaches, penalties are increased exponentially in PaaS and Saas layers. Especially, the THR approach will return the full cost of the service due to the availability constraints.
8. CONCLUSIONS

In this paper, we present an OpenStack version of the Generic SLA Manager, which could be further applied into project PaaSage. Based on the proposed autonomous SLA violation-filtering framework, by combining IaaS (OpenStack-SLAM) and PaaS (OpenShift) as a use case, applying a multi-domain SLA pricing & penalty model and introducing a resource allocation strategy, we show experimentally that we can manage VM live migration more efficiently than current state of the art.

In the future, based on the above simulation results, we would like to take this work one step further by using the cloud emulation tool “Emusim” to automatically extract information from various types of workloads (e.g., CPU intensive, memory intensive, network intensive etc.) via emulation and then uses this information to generate the corresponding simulation model. Thus, the results of emulation can be used to prove the correctness and accuracy of simulation.

In order to set performance monitoring policies in an SLA to alert the SLAM when a server nears the threshold of satisfactory performance as agreed upon with the customer, a light-weight hierarchical system will be chosen to represent and monitor the SLAs in a fault-tolerant fashion. As such, the SLA violation can be avoided maximally by resolving the problem in the right component. Since Akka (2013) system is such a framework with features such as message passing, scalability, fault-tolerance and high availability, we plan to apply it into our autonomous SLA violation-filtering framework.

9. ACKNOWLEDGMENT

The research leading to these results is supported by Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) in Germany.

10. REFERENCES


Kuan Lu is working as a scientific researcher and PhD candidate of the GWDG at the University of Göttingen, Germany. He received his Master degree in computer engineering from Duisburg-Essen University, Germany. His expertise lies within planning and optimization for service level agreements of cloud computing, resource scheduling and allocation management, parallel computing, service computing. He has been actively involved in the FP7 IP PaaSage, SLA@SOI projects and the projects of Lotus Mashup and MILA at IBM Research and Development GmbH, Böblingen, Germany.

Prof. Ramin Yahyapour is the managing director of the GWDG at the University of Göttingen, Germany. He is active in research on clouds, grid and service oriented infrastructures for several years. His research interest lies in resource management. He is a steering group member and on the Board of Directors in the Open Grid Forum. He has been active in several national and European research projects. Notably, he was the scientific coordinator of the FP7 IP SLA@SOI and was a steering group member in the CoreGRID Network of Excellence.

Constantinos (Costas) Kotsokalis received his electrical and computer engineering diploma from the National Technical University of Athens, Greece, and his PhD from the Dortmund University of Technology, Germany. His expertise lies within the domain of distributed computing at large, with a focus mostly on service level agreements. Additional scientific topics of interest include distributed algorithms, job / resource scheduling and data management. Costas has worked in multiple international projects and diverse domains such as wide-area networking, grid / cloud computing, SLA management, remote instrumentation and e-Health.

Philipp Wieder is deputy leader of data center of the GWDG, Germany. He received his PhD from the Dortmund University of Technology, Germany. He has been active in research on clouds, grid and service oriented infrastructures for several years. His research interest lies in distributed system, service level agreements and resource scheduling. He has been actively involved in the FP7 IP PaaSage, SLA@SOI and SLA4D-Grid projects.

Edwin Yaqub is working as a Research Assistant and PhD candidate of the GWDG at the University of Göttingen, Germany. He received his Master degree in computer science from RWTH Aachen Technical University, Germany. His research expertise includes SLA Negotiations for cloud and service computing. He has been actively involved in the FP7 IP PaaSage, SLA@SOI projects. Previously, he worked as a software engineer on e-Commerce, web and desktop based business applications. During studies, he worked part time at the University, Philips Research Labs and Siemens Research, Aachen, Germany.

Ali Imran Jehangiri studied in Bergische Universität Wuppertal, where he did Master in computer simulation in the disciplinary science of experimental particle physics. He later joined the chair of service computing at Dortmund University of Technology, Germany. Currently, he is a PhD candidate of the GWDG at the University of Göttingen, Germany. His research interest is in monitoring of service level agreements for virtual platforms.
RAPIDLY ALTERNATING BOTTLENECKS: A STUDY OF TWO CASES IN N-TIER APPLICATIONS

Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, Calton Pu

1College of Computing, Georgia Institute of Technology
2System Software Laboratories, FUJITSU LABORATORIES LTD.
3ICT Systems Laboratories, FUJITSU LABORATORIES LTD.

{qywang, jack.li, calton}@cc.gatech.edu, {kanemasa, shimizut, matz, kawaba}@jp.fujitsu.com

Abstract

Identifying the location of performance bottlenecks is a non-trivial challenge when scaling n-tier applications in computing clouds. Specifically, we observed that an n-tier application may experience significant performance loss when bottlenecks alternate rapidly between component servers. Such rapidly alternating bottlenecks arise naturally and often from resource dependencies in an n-tier system and bursty workloads. These rapidly alternating bottlenecks are difficult to detect because the saturation in each participating server may have a very short lifespan (e.g., milliseconds) compared to current system monitoring tools and practices with sampling at intervals of seconds or minutes. Using passive network tracing at fine-granularity (e.g., aggregate at every 50ms), we are able to correlate throughput (i.e., request service rate) and queue length (i.e., number of concurrent requests) in each server of an n-tier system. Our experimental results show conclusive evidence of rapidly alternating bottlenecks caused by system software (JVM garbage collection) and middleware (VM collocation).

Keywords: bottleneck, n-tier, performance analysis, scalability, web application

1. INTRODUCTION

Web-facing enterprise applications such as electronic commerce are not embarrassingly parallel (e.g., web indexing and data analytics). They are typically implemented using an n-tier architecture with web server, application server, and database server tiers. Such n-tier applications have implicit dependencies among their components, which create alternating bottlenecks (Balbo & Serazzi, 1997; Casale & Serazzi, 2004; Malkowski, Hedwig, & Pu, 2009; Mi, Casale, Cherkasova, & Smirni, 2008). These alternating bottlenecks are both interesting and challenging. They are interesting because they cause the entire n-tier system to reach its performance limit (i.e., flat throughput) even though all system resources are measurably below 100% utilization. They are challenging because classic queuing models that assume independent jobs predict single resource saturation bottlenecks, so they are inapplicable to alternating bottlenecks.

The main hypothesis of this paper is that (contrary to previously common belief) alternating bottlenecks occur naturally in real application scenarios and they can be found by experimental measurements using appropriate tools. Alternating bottlenecks constitute an important problem because there is lingering skepticism about their prevalence (and even existence) in the real world, despite early theoretical predictions (Balbo & Serazzi, 1997; Casale & Serazzi, 2004; Mi et al., 2008). In the past, observed evidence of alternating bottlenecks was rare and it was not easy to reproduce them reliably in experiments. We report consistent experimental results which suggest that alternating bottlenecks may be far more common than previously believed. The perception of rarity is simply due to many alternating bottlenecks being short-lived (on the order of tens of milliseconds). Consequently, these interesting phenomena have been (and still are) completely invisible to normal monitoring tools that sample at time intervals measured in seconds or minutes.

The main contribution of the paper is an unequivocal confirmation of our hypothesis through reproducible experimental observation of two rapidly alternating bottlenecks when running the standard n-tier RUBBoS benchmark ("RUBBoS: Bulletin board benchmark," 2004). Specifically, we found that bottlenecks alternate between the Tomcat tier and the MySQL tier at time interval of tens of milliseconds. Our study further shows that alternating bottlenecks can be caused by factors at the software level (e.g., JVM garbage collection, see Section 4) and middleware level (e.g., VM collocation, see Section 5). Despite its relatively short duration, the impact of this alternating bottleneck becomes significant when the frequency and intensity of the alternating pattern increase. The detection of alternating bottlenecks becomes a problem when the frequency and intensity of the alternating pattern increase.

http://hipore.com/ijcc
doi: 10.29268/stcc.2013.1.1.2
Figure 1: A rapidly alternating bottleneck case of a 4-tier system with Browse-only workload (CPU intensive). The system achieves the highest throughput at WL 14,000 as shown in Figure 1(a) while no hardware resources are fully saturation as shown in Figure 1(b) and Figure 1(c).

Table 1: Average resource utilization in each tier at WL 14,000. Except Tomcat and MySQL CPU, the other system resources are far from saturation.

<table>
<thead>
<tr>
<th>Server/Resource</th>
<th>CPU util (%)</th>
<th>Disk I/O (%)</th>
<th>Network receive/send (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>45.9</td>
<td>0.5</td>
<td>23.8/39.9</td>
</tr>
<tr>
<td>Tomcat</td>
<td><strong>86.9</strong></td>
<td>0.3</td>
<td>7.6/13.1</td>
</tr>
<tr>
<td>CJDBC</td>
<td>36.2</td>
<td>0.2</td>
<td>11.2/14.3</td>
</tr>
<tr>
<td>MySQL</td>
<td><strong>84.3</strong></td>
<td>0.4</td>
<td>0.8/4.6</td>
</tr>
</tbody>
</table>

2.2 MULTI-BOTTLENECKS

Multi-bottlenecks describe a phenomenon where an n-tier system is saturated (i.e., achieves the maximum throughput) while no single hardware resource is fully utilized (Malkowski et al., 2009). We use an example to illustrate this phenomenon. The example was derived from a three-minute experiment of RUBBoS running on a four-tier configuration (1L/2S/1L/2S, see Figure 14(c)). The details of the experimental setup are in Appendix A.

Figure 1(a) shows the system works well from a workload of 1,000 concurrent users to 13,000. At 14,000, the average response time increases significantly and the throughput reaches a maximum. The interesting observation is that the saturated system does not have any single resource bottleneck. Since we use the CPU intensive Browse-only workload of this benchmark, we show the timeline graphs (with one second granularity) of CPU utilization. During the execution of the WL 14,000, both Tomcat (Figure 1(b) and MySQL (Figure 1(c)) show less than full CPU utilization, with an average of 86.9% (Tomcat) and 84.3% (MySQL). We also summarize the average usage of other main hardware resources of each server in Table 1. This table shows that except for Tomcat and MySQL CPU, the other system resources are far from saturation.

This example shows that simply monitoring hardware resource utilization may be unable to identify the system bottleneck, since there is no single saturated resource. Later in Section 4 we explain that this is a rapidly alternating
bottleneck case (a special case of multi-bottleneck) where the bottleneck alternates rapidly between MySQL and Tomcat. During normal processing, MySQL CPU is the primary system bottleneck, being fully utilized for processing requests sent from Tomcat. However, the Tomcat JVM garbage collection process freezes request processing and consumes the server CPU (at the granularity of milliseconds). Thus the Tomcat becomes the bottleneck during garbage collection. In either case, the system throughput is limited.

2.3 RAPIDLY ALTERNATING BOTTLENECKS

Rapidly alternating bottlenecks are a special case of multi-bottlenecks that the bottleneck in an n-tier system alternates rapidly (on the order of tens of milliseconds) among multiple system resources while at any moment one system resource becomes the main bottleneck. Rapidly alternating bottlenecks arise due to the implicit dependencies among servers in an n-tier system. For example, requests that originate from a client arrive at the web server, which distributes them among the application servers, which in turn ask the database servers to carry out the query. The dependencies among the servers are in the long invocation chain (through RPC calls) of transaction processing in the system and maintained by soft resources (e.g., threads and database connections (Wang et al., 2011)). Such dependencies may cause requests to congest in different servers at different time period. For example, Figure 2 demonstrates a rapidly alternating bottleneck case in a 3-tier system. This figure shows that the bottleneck alternates between Tomcat and MySQL at tens of milliseconds level, thus monitoring resource utilization at every second can rarely detect any resource saturation (similar to what Figure 1(b) and Figure 1(c) show).

The identification of rapidly alternating bottlenecks as an important phenomenon is due to its significant impact in cloud computing environments, where hardware resources are supposed to be “infinite” for applications to scale. Detection of rapidly alternating bottlenecks poses significant challenges to current state of the art monitoring tools, which leads to inefficient performance management of applications deployed in cloud.

3. DETECTION OF RAPIDLY ALTERNATING BOTTLENECK

In this section, we briefly explain our fine-grained analysis to detect rapidly alternating bottlenecks. This kind of analysis is necessary to detect a bottleneck alternating on the order of tens of milliseconds among servers. Later we will show two case studies of applying our method to detect rapidly alternating bottlenecks caused by JVM garbage collection (Section 4) and VM collocation (Section 5).

3.1 TRACE MONITORING TOOL

Our fine-grained analysis is based on the tracing of client transaction executions of an n-tier system. Before we start the details of the fine-grained analysis method, we first briefly explain our tool (Fujitsu SysViz ("Fujitsu SysViz: System Visualization," 2010)) for the tracing of transaction executions in an n-tier system. A client transaction service an entire web page requested by a client and may consist of multiple interactions between different tiers. Figure 3 shows an example of such a trace (numbered arrows) of a client transaction execution in a three-tier system. SysViz is able to reconstruct the entire trace of each transaction executed in the system based on the traffic messages (odd-numbered arrows) collected through a network switch which supports passive network tracing. In our experimental environment, all the servers are connected to the network switch, which forwards all the traffic messages to a dedicated SysViz server. Thus, the arrival/departure timestamps of each request (small boxes with even-numbered arrows) for any server can be recorded by the SysViz server. SysViz requires no modification on application source code and has a negligible performance impact on the target n-tier application. We note that since the timestamps of all messages are assigned by one dedicated SysViz server, the precision of the derived processing time of each request in any tier in the system is close to microsecond level. Thus, the influence of clock errors between machines caused by limited accuracy of NTP can be avoided.

In fact the transaction tracing technology has been...
Figure 4: Fine-grained analysis of MySQL when the system is at WL 14,000. Figure 4(a) and Figure 4(b) show the MySQL queue length and throughput measured at the every 50ms time interval. Figure 4(c) is derived from Figure 4(a) and Figure 4(b); each point in Figure 4(c) represents the MySQL queue length and throughput measured at the same 50ms time interval during the 12-second experimental time period.

3.2 FINE-GRAINED QUEUE-LENGTH/THROUGHPUT ANALYSIS

Since each participating server in a rapidly alternating bottleneck case only presents short-term saturations, a key point of detecting the rapidly alternating bottleneck is to find the participating short-term saturated servers. Instead of monitoring hardware resource utilizations, our approach measures a server's queue length and throughput in continuous fine-grained time intervals. The throughput of a server can be calculated by counting the number of completed requests in the server in a fixed time interval, which can be 50ms, 100ms, or 1s. Queue length is the average number of queued requests over the same time interval. Both these two metrics for each server in the system can be easily derived from the trace captured by SysViz.

Figure 4(a) shows the MySQL queue length average at every 50ms time interval over a 12-second time period for the 1L/2S/1L/2S configuration case at WL 14,000 (See the case in Figure 1). This figure shows that a large number of requests are queued in MySQL from time to time, which suggests MySQL frequently presents short-term saturation. Figure 4(b) shows the MySQL throughput average at every 50ms time interval over the same 12-second time period as in Figure 4(a). This figure shows that in some time intervals MySQL even produces zero throughput, which suggests MySQL is frequently under-utilized.

To precisely diagnose in which time intervals a server presents short-term saturation, we need to correlate the server's queue length and throughput as shown in Figure 4(c). This figure is derived from Figure 4(a) and Figure 4(b); each point in Figure 4(c) represents the MySQL queue-length/throughput measured at the same 50ms time interval during the 12-second experimental time period. This figure shows the clear trend of queue-length/throughput correlation (we call the trend as main sequence curve), which is consistent with Denning et al.'s (Denning & Buzen, 1978) operational analysis result for the relationship between a server's queue length and throughput. Specifically, a server's throughput increases as the queue length on the server increases until it reaches the maximum throughput $TP_{\text{max}}$. The saturation point $N^*$ is the minimum queue length beyond which the server starts to saturate.

Once $N^*$ is determined, we can judge in which time intervals a server is saturated based on the measured queue length. For example, Figure 4(c) highlights three points labeled 1, 2, and 3, each of which represents the queue-length/throughput in a time interval that can match back to Figure 4(a) and Figure 4(b). Point 1 shows that MySQL is saturated in the corresponding time interval because the long queue length far exceeds $N^*$. Point 2 shows that MySQL is not saturated due to the zero queue length and throughput. Point 3 also shows that MySQL is not saturated.

---

1. At each time tick, we know how many requests for a server have arrived, but not yet departed. This is the number of concurrent requests being processed by the server. Concurrent requests can also be thought as “queued” requests. More detailed fine-grained queue-length/throughput calculation can be found in (Wang, Kanemasa, Li, Jayasinghe, et al., 2013).

2. The queue length in their definition is the load in a system, which means the number of jobs being processed concurrently.

3. Due to the Utilization Law, the maximum throughput $TP_{\text{max}}$ of a server is fixed by the bottleneck resource in terms of $1/d$, where $d$ is the service demand for the bottleneck resource per job (Denning & Buzen, 1978).
Figure 5: Fine-grained queue-length/throughput analysis for Tomcat and MySQL. Figure 5(c) and Figure 5(f) are derived from Figure 5(b) and Figure 5(e) respectively, with 3-minute experimental data. Figure 5(b) shows that there are some time intervals that Tomcat has high queue length but low throughput (see the point labeled 4), which correspond to the low queue length and throughput of MySQL as shown in Figure 5(e)(see the point labeled 4).

Figure 6: Correlation analysis of the rapidly alternating bottleneck between Tomcat and MySQL at WL 14,000. Figure 6(a) shows that Tomcat and MySQL queue length have strong negative correlation. Figure 6(b) shows that the peaks of Tomcat queue length are due to frequent JVM GCs.

because the corresponding queue length is less than $N^*$ though it generates relatively high throughput.

After we detect all the short-term saturated servers, the next step is to analyze whether the short-term saturation of each participating server occurs in an alternating pattern. We will illustrate this point in the following two case studies.

4. RAPIDLY ALTERNATING BOTTLENECK CAUSED BY JVM GC

In this section we explain the rapidly alternating bottleneck mentioned in Section 2.2. In that example, the poor system performance is caused by the frequent short-term saturations of both Tomcat and MySQL. Our further correlation analysis shows that the frequent JVM GCs in
Figure 7: Collocation strategy between SysA and SysB; SysA-App2 is collocated with SysB-DB.

Table 2: Workload of SysA and SysB in each of the five collocation experiments. SysA is at constant stable WL 14,000 and SysB is at constant workload but with different burstiness levels (from I = 1 to 400).

<table>
<thead>
<tr>
<th>#</th>
<th>WL (users)</th>
<th>Burstiness level</th>
<th>CPU (%)</th>
<th>WL (users)</th>
<th>Burstiness level</th>
<th>CPU (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>14,000</td>
<td>1 - 1</td>
<td>74.0</td>
<td>0</td>
<td>Non-col</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>14,000</td>
<td>1 - 1</td>
<td>74.9</td>
<td>400</td>
<td>1 - 1</td>
<td>10.2</td>
</tr>
<tr>
<td>3</td>
<td>14,000</td>
<td>1 - 1</td>
<td>74.7</td>
<td>400</td>
<td>1 - 100</td>
<td>10.6</td>
</tr>
<tr>
<td>4</td>
<td>14,000</td>
<td>1 - 1</td>
<td>75.5</td>
<td>400</td>
<td>1 - 200</td>
<td>10.5</td>
</tr>
<tr>
<td>5</td>
<td>14,000</td>
<td>1 - 1</td>
<td>75.2</td>
<td>400</td>
<td>1 - 400</td>
<td>10.8</td>
</tr>
</tbody>
</table>

Tomcat cause the bottleneck to alternate between Tomcat and MySQL.

Figure 5 shows the fine-grained load/throughput analysis for Tomcat and MySQL at WL 7,000 and 14,000 with the same system configuration as in Section 2.2. Figure 5(a) and Figure 5(d) show that both Tomcat and MySQL are not saturated at WL 7,000 since the load of each tier is below the corresponding saturation point N*, which is derived from Figure 5(c) and Figure 5(f) respectively.

The interesting figures are Figure 5(b) and Figure 5(e), which show that at WL 14,000 both the Tomcat tier and the MySQL tier frequently present short-term saturations. Specially, Figure 5(b) shows that in some time intervals the Tomcat load is high (e.g., the point labeled 4) but the corresponding throughput is zero, which means that many requests are queued in Tomcat but no output responses (throughput). Figure 5(c), which is derived from Figure 5(b) but based on the 3-minute runtime experiments, shows that there are many time intervals when Tomcat has a high load but low or even zero throughput (POI inside the rectangular area). Since Tomcat is the upstream tier of MySQL, the output responses of Tomcat feeds the input requests of MySQL; thus having fewer output responses from Tomcat means there will be fewer input requests sent to MySQL, which leads to the zero throughput and load of MySQL (see the point labeled 4’ in Figure 5(c)).

To illustrate the rapidly alternating bottleneck between Tomcat and MySQL, Figure 6(a) shows the correlation between the Tomcat load and the MySQL load over the same 12-second time period. This figure shows that these two metrics have a negative correlation (the Pearson correlation is -0.42), which suggests that the short-term saturation alternates between Tomcat and MySQL. Thus, the reason for the limited system throughput is clear: at any moment either Tomcat or MySQL becomes the bottleneck in the system.

Our further analysis shows that the short-term saturations of Tomcat are caused by frequent JVM GC. In this set of experiments, the JVM in Tomcat (with JDK 1.5) uses a synchronuous garbage collector; it waits during the garbage collection period and only starts processing requests after the garbage collection is finished. To confirm that JVM GC causes the bottleneck in Tomcat, Figure 6(b) shows the timeline graph which correlates the Java GC running ratio4 with the Tomcat load (50ms). This figure shows the occurrence of Tomcat JVM GC has a strong positive-correlation with the high load in Tomcat. The high peaks of JVM GC in Figure 6(b) stop Tomcat and make requests queued in Tomcat dramatically. We note that such long freeze times in Tomcat do not happen frequently when the system is under low workload as shown in Figure 5(a).

This is because JVM GC has a non-linear relationship with the amount of workload (Wang, Kanemasa, Kawaba, & Pu, 2012).

5. RAPIDLY ALTERNATING BOTTLENECK CAUSED BY VM COLLOCATION

In this section we show another rapidly alternating bottleneck case due to VM collocation, i.e., collocating multiple under-utilized VMs into the same physical host so that VMs can share hardware resources. Although VM collocation can reduce infrastructure/maintenance

4 Java GC running ratio means the total time spent on Java GC during each monitoring time interval to the total monitoring time interval length. JVM provides a tool recording the starting/ending timestamp of every GC activity.
Figure 9: Fine-grained queue-length/throughput analysis for the Tomcat and MySQL of SysA in the collocation experiments. Figure 9(c) and Figure 9(f) are derived from Figure 9(b) and Figure 9(e) respectively, with 3-minute experimental data.

Figure 9(a) and Figure 9(d) show that the queue length of the SysA Tomcat/MySQL tier is low when SysB workload is stable (I = 1). As the burstiness of SysB workload increases (I = 400), there are some time intervals that SysA-App has high queue length but low throughput (see the point labeled 5 in Figure 9(b)), which correspond to the low queue-length/throughput of SysA-DB (see the point labeled 5’ in Figure 9(e)).

Figure 10: Analysis of rapidly alternating bottleneck between SysA-App and SysA-DB. Figure 10(b) shows that the peaks of SysA-App queue length are caused by the burst of SysB-DB CPU utilization because SysA-App and SysB-DB are competing for CPU resources.

costs (Barham et al., 2003; Govindan, Liu, Kansal, & Sivasubramaniam, 2011), it may significantly hamper the performance of the collocated applications in a non-trivial way, especially when the workload for the collocated applications becomes bursty (Kanemasa, Wang, Li, Matsubara, & Pu, 2013; Malkowski et al., 2012). For example, rapidly alternating bottlenecks may occur in a foreground application if it is collocated with another application facing a bursty workload.

We illustrate this problem by collocating two VMs, each of which is from a separate n-tier application, into the same host and with each VM sharing the same CPU core. Figure 7 shows our collocation strategy of the two applications; SysA with 1L/2S/1L/2S configuration (4-tier) and SysB with...
with 1S/1S/1S configuration (3-tier). SysA keeps the same hardware configuration as in the previous section but with JDK1.6 in Tomcat. The VM of SysA-App2 is collocated with the VM of SysB-DB on the same ESXi host and they share the same CPU core; the VMs of the front tiers (one Apache and one Tomcat) of SysB are deployed in separate ESXi hosts from SysA in order to simplify the analysis. Table 2 shows the workload conditions and average CPU utilization of both SysA and SysB in the collocation experiments. SysA is at a constant stable workload of 14,000 in all five experiments. Except for the first experiment (the non-collocation case), SysB is under constant WL 400 but with varying burstiness levels, which is represented by $I$. This table shows that the average CPU utilization of both the collocated VMs SysA-App2 and SysB-DB are almost constant and the total of the CPU utilization is less than 90%, which justifies the collocation strategy based on traditional bin packing practices.

Figure 8 shows the average response time of SysA in all the five cases. This figure shows that the SysA response time is almost the same when the collocated SysB has the relatively stable workload ($I = 1$), and increases significantly when the burstiness level of the workload for SysB becomes high (e.g., $I = 400$). The significant increase in SysA response time may seem strange since the average CPU utilization remains constant as seen in Table 2.

Figure 9(b) shows a similar interesting phenomenon as in the previous rapidly alternating bottleneck case that in some time intervals (e.g., between 52s and 53s) the SysA-App has a long queue length but low throughput; the low throughput of SysA-App leads to the low queue length and throughput in SysA-DB during the same time period as shown in Figure 9(e). Figure 9(c) (derived from Figure 9(b) but based on 3-minute experimental data) suggests that there are many time intervals when SysA-App has a long queue length but low throughput (points in POI). During these time intervals, SysA-App presents short-term saturations and SysA-DB is under-utilized due to the low number of input requests sent from SysA-App (see Figure 9(f)).

Figure 10(a) shows the correlation of the queue length between SysA-App and SysA-DB over the same 8-second time period. This figure shows that SysA-App queue length has a negative correlation ($\rho = -0.46$) with SysA-DB queue length, which suggests the bottleneck alternates rapidly between SysA-App and SysA-DB.

Our further analysis shows that the short-term saturation of SysA-App is caused by the burst of SysB-DB CPU utilization. Figure 10(b) shows the timeline graph of the CPU utilization of SysB-DB (measured using VMware esxtop with 2s granularity) and the SysA-App queue length (measured at every 50ms time interval). This figure shows that the SysA-App queue length increases significantly when there is a spike in the SysB-DB CPU utilization, which indicates that the Tomcat tier of SysA temporarily becomes the bottleneck due to the interference of SysB-DB. The detailed research about the race of CPU scheduling between collocated VMs has been provided in (Kanemasa et al., 2013).

### 6. Resolving Rapidly Alternating Bottlenecks

Once we detect a rapidly alternating bottleneck, we can resolve the bottleneck through various ways, depending on whether we can find the exact cause for the rapidly alternating bottleneck. Specifically, we can simply scale-out/up the participating servers if we cannot find the exact cause, or we can resolve the bottleneck by addressing the exact cause. For example, we can resolve the rapidly alternating bottleneck caused by frequent JVM GCs in Tomcat through upgrading the JDK version from 1.5 to 1.6, which has more efficient garbage collectors. We can also resolve the rapidly alternating bottleneck caused by VM collocation through VM migration.

In this section we show the performance gain after we resolve the rapidly alternating bottleneck caused by frequent JVM GCs in Tomcat (Section 4). We choose to resolve the bottleneck by upgrading the JDK version from 1.5 to 1.6 in Tomcat since we know the exact cause. The experiments have the same hardware/software configuration as in Section 3.2 except for the Tomcat JDK version.

Figure 11 shows the fine-grained queue-length/throughput analysis for Tomcat and MySQL at WL 14,000 and 16,000. Recall from Section 4 the system throughput reaches the maximum at WL 14,000 due to the rapidly alternating bottleneck between Tomcat and MySQL before the JDK version upgrade. After the JDK version upgrade, Figure 11(a) and Figure 11(d) show that only MySQL presents frequent short-term saturations at WL 14,000; further workload increase to 16,000 leads to the full saturation of MySQL as shown in Figure 11(c) (the queue length is above the N* most of the time). Thus, the rapidly alternating bottleneck is resolved after the JDK upgrade in Tomcat. Specifically, the POIs in Figure 5(c) do not appear in Figure 11(c), which means Tomcat does not have long “freezing” periods. Accordingly, there are only a few time intervals with low queue length and low throughput in MySQL (see Figure 11(e)), which means MySQL is fully utilized.

---

5 The upgrade of JDK version in Tomcat solves the rapidly alternating bottleneck caused by frequent JVM GCs; see Section 6 for more details.

6 Mi et al. (Mi, Casale, Cherkasova, & Smirni, 2009) introduced index of dispersion (abbreviated as $I$) to characterize the intensity of the traffic surges. The larger the $I$ is, the longer the duration of the traffic surge. The burstiness level of the by default RUBBoS workload is $I = 1$.

7 We couldn’t measure the fine-grained CPU usage of SysB-DB because 2s is the finest granularity that the latest esxtop supports.

8 JDK 1.6 uses garbage collection algorithms which support both parallel and concurrent garbage collection.
7. DISCUSSION

So far we illustrated that rapidly alternating bottlenecks can be caused by system software level factors (e.g., JVM GC) and middleware level factors (VM collocation). In fact there are many other factors that can cause rapidly alternating bottlenecks such as Dynamic Voltage and Frequency Scaling (DVFS) technology at the architecture level (Wang, Kanemasa, Li, Lai, et al., 2013). The reason why rapidly alternating bottlenecks can happen frequently is because there are many factors that can cause short-term saturations of component servers in real systems, especially when the system is at high utilization. The short-term saturations of component servers, compounded with dependencies among tiers in an n-tier system, create rapidly alternating bottlenecks in the system.

The short-term saturation of a component server in an n-tier system has a significant impact on the servers in other tiers of the system due to resource dependencies among tiers...
in the system. For example, the short-term saturation (either caused by JVM GC or VM collocation) of a server causes a bursty workload to the downstream tiers, causing downstream tiers to switch between a saturation and non-saturation state. Specifically, during the short-term saturation of a server, many requests start to queue in the server, which causes the downstream tiers to be underutilized (non-saturation state); after the short-term saturation period, the queued requests are pushed to the downstream tiers in a batch mode, which cause the short-term saturation of the downstream tiers; the short-term saturation may present an alternating pattern between the server and the downstream tiers once the frequency of the short-term saturations of the server becomes high.

Figure 13 shows the comparison of MySQL queue length distribution before and after resolving the rapidly alternating bottlenecks caused by the short-term saturations of Tomcat (see Section 4 and Section 5). The system in all these three cases is under the same hardware configuration (1L/2S/1L/2S), at the same WL 14,000, and has the same amount of experimental time (3-minute). However, Figure 13(b) and Figure 13(c) show that MySQL has more frequent long queue length compared to the case after we resolve the alternating bottleneck as shown in Figure 13(a). Specially, Figure 13(e) even shows a clear bi-modal distribution. This is because in a rapidly alternating bottleneck case the frequent short-term saturations of Tomcat augment the burstiness level of the workload to MySQL, which causes more frequent short-term saturations in MySQL.

8. RELATED WORK

Performance debugging and scalability analysis for distributed systems such as n-tier systems has been widely studied in previous research efforts.

Shifting/Alternating bottlenecks have been studied before in either multiclass queueing networks or n-tier enterprise systems. Balbo et al. (Balbo & Serazzi, 1997) and Casale et al. (Casale & Serazzi, 2004) use analytical approaches to illustrate that bottlenecks in a multiclass queueing network with load independent servers can switch to different servers, depending on the current workload mix. Malkowski et al. (Malkowski et al., 2009) showed an alternating bottleneck case where the bottleneck alternates among eight MySQL databases due to the unbalanced query dispatching from the upper tiers. As shown in this paper, alternating bottlenecks can be far more common than previously believed. The reason behind is that the detection of an alternating bottleneck becomes extremely difficult as the frequency of alternating pattern increases; such interesting phenomena are completely invisible to normal monitoring tools that sample at time intervals measured in seconds or minutes.

Analytical models have been proposed for bottleneck detection and performance prediction of n-tier systems. Urgaonkar et al. (Urgaonkar, Pacifici, Shenoy, Spreitzer, & Tantawi, 2005; Urgaonkar, Shenoy, et al., 2005) present a novel dynamic provisioning technique for n-tier systems that employs a flexible queueing model to determine how much resources to allocate to each tier of the application. However, such model is based on Mean Value Analysis (MVA), which has difficulties dealing with alternating bottleneck cases in the system. Mi et al. (Mi et al., 2008) propose a more sophisticated analytical model that considers bottleneck shifting in an n-tier system due to bursty workloads. One challenge of this work is to precisely map the bursty characteristics of a workload to the queuing model with multiple service rates for each queue in the system. As shown in this paper, without fine-grained monitoring (sub-second level) granularity, the bursty characteristics of a workload can be largely masked.

Software mis-configuration and failure detection of distributed system have been studied in (Attariyan & Flinn, 2010; Oliveira, Tjang, Bianchini, Martin, & Nguyen, 2010). Attariyan et al. (Attariyan & Flinn, 2010) present a tool that

http://hipore.com/ijcc 22
locates the root cause of configuration errors by applying dynamic information flow analysis within process (mainly) in the runtime. Oliveira et al. (Oliveira et al., 2010) propose a mistake-aware management framework for protecting n-tier systems against operator mistakes by facilitating the previous correct operations. All these works differ from our work in that they focus more on faults or anomalous behavior of system components rather than the system performance problem.

Perhaps the work closest to ours is Aguilera et al.'s performance debugging based on Black boxes (Aguilera, Mogul, Wiener, Reynolds, & Muthitacharoen, 2003). The authors propose a statistical method that derives causal paths (the trace of a transaction) in a distributed system (not limited to n-tier system) from the communication messages between different nodes. By measuring the delay of request processing in each node, they detect the “bottlenecked” server as the node where a request has the longest delay in the system. Though this approach can be effective to detect the single bottleneck case, it may fail to detect rapidly alternating bottlenecks since requests can wait in multiple servers in an alternating pattern.

9. CONCLUSIONS

We observed that the performance of an n-tier system may degrade significantly due to rapidly alternating bottlenecks between multiple tiers. We found that rapidly alternating bottlenecks can be caused by various factors at different levels of an n-tier application; for instance, JVM GC at the software level (Section 3.2), VM collocation in the middleware level (Section 3.3). Solving those rapidly alternating bottlenecks leads to significant performance improvement (Section 4). We proposed a novel bottleneck detection method to detect these rapidly alternating bottlenecks (Section 5), where the effectiveness of our approach is validated through the two case studies in Section 3. More generally, our work is an important contribution towards scaling complex n-tier applications under elastic workloads in cloud environments.

10. REFERENCES


APPENDIX A: EXPERIMENTAL SETUP

In our experiments we adopt the RUBBoS standard n-tier benchmark (“RUBBoS: Bulletin board benchmark,” 2004), based on bulletin board applications such as Slashdot. RUBBoS has been widely used in numerous research projects due to its design, derived from a real web-facing application. RUBBoS can be configured as a three-tier (web server, application server, and database server) or four-tier (addition of clustering middleware such as C-JDBC(Cecchet, Marguerite, & Zwaenepoel, 2004)) system. The benchmark includes two kinds of workload modes: browse-only and read/write interaction mixes. We use browse-only workload in this paper. The closed-loop workload generator of this benchmark generates a request rate that follows a Poisson distribution parameterized by a number of emulated clients and a fixed user thinking time. Such workload generator has a similar design as other standard n-tier benchmarks such as RUBiS, TPC-W, Cloudstone etc.

We run the RUBBoS benchmark on our virtualized testbed. Figure 14 outlines the software components, ESXi host and virtual machine (VM) configuration, and a sample topology used in the experiments. We use a four-digit notation #W/#A/#B/#D to denote the number of web servers, application servers, clustering middleware servers, and database servers. Each server runs on top of one VM. We have two types of VMs: ‘’L’’ and ‘’S’’, each of which represents a different size of processing power. Figure 14(c) shows a sample 1L/2S/1L/2S topology. Each ESXi host runs the VMs from the same tier of the application. The VMs from the same tier are pinned to separate CPU cores to minimize the interference between VMs. Hardware resource utilization measurements (e.g., CPU) are taken during the runtime period using Sysstat at one second granularity and VMware esxtop at two second granularity (the minimum granularity provided by the tool). The esxtop is used to monitor the CPU utilization of each VM in the VM collocation experiments (see Section 5).

Authors

Qingyang Wang received the BS degree in computer science from Wuhan University, China, in 2004, and the MS degree in computer science from Chinese Academy of Sciences, in 2007. He is currently a computer science PhD candidate in College of Computing, Georgia Institute of Technology, Atlanta, Georgia. His research focuses on performance debugging and scalability analysis for distributed systems, especially multithreaded enterprise systems.
Yasuhiko Kanemasa received the B.Eng. degree in computer engineering from Tokyo Institute of Technology (Tokyo, Japan) in 1996, and the M.S. degree in computer science from Japan Advanced Institute of Science and Technology (Nomi, Japan) in 1998. He has been working for Fujitsu Laboratories Ltd. (Kawasaki, Japan) as a researcher since 1998. His research interests are in the areas of database systems, performance analysis of enterprise computer systems, and cloud computing. He is a member of IPSJ and DBSJ.

Jack Li is a PhD student in the College of Computing at the Georgia Institute of Technology. He received a BSE in electrical and computer engineering from Duke in 2010. His current research interests distributed and cloud computing, hypervisors and virtualization, MapReduce, and n-tier application performance and scalability.

Toshihiro Shimizu received his B.S. and M.S. in Computer Science from the University of Tokyo in 1994 and 1996, respectively. He has been working for Fujitsu Laboratories Ltd. since 1999. His current major research interests are in verification and testing of software systems.

Masazumi Matsubara received the BE, ME and PhD in Engineering from University of Tsukuba in 1996, 1998 and 2001, respectively. He has been working for Fujitsu Laboratories Ltd. (Kawasaki, Japan) as a researcher since 2001. His research interests are in the areas of parallel computing, system management, and cloud computing. He is a member of IPSJ.

Motoyuki Kawaba received the B.E degree in Electronic Engineering from the University of Tokyo in 1989, and the M.E. degree in Information Engineering from the University of Tokyo in 1991. He joined Fujitsu Laboratories Ltd., Kawasaki, Japan in 1991 and has been engaged in research and developments of computer systems and architectures. His research interests include storage systems, data management systems, and performance analysis of commercial workloads.

Calton Pu was born in Taiwan and grew up in Brazil. He received his PhD from University of Washington in 1986 and served on the faculty of Columbia University and Oregon Graduate Institute. Currently, he is holding the position of Professor and John P. Imlay, Jr. Chair in Software in the College of Computing, Georgia Tech. He has worked on several projects in systems and database research. His contributions to systems research include program specialization and software feedback. His contributions to database research include extended transaction models and their implementation. His recent research has focused on automated system management in clouds (Elba project) and document quality, including spam processing. He has collaborated extensively with scientists and industry researchers. He has published more than 70 journal papers and book chapters, 200 conference and refereed workshop papers.
CROSS CLOUD MAPREDUCE: A RESULT INTEGRITY CHECK FRAMEWORK ON HYBRID CLOUDS

Yongzhi Wang*, Jinpeng Wei*, Mudhakar Srivatsa§
* Florida international University, Miami, USA
§ IBM T.J. Watson Research Center, Yorktown Heights, USA

Abstract
Large-scale adoption of MapReduce computations on public clouds is hindered by the lack of trust on the participating virtual machines, because misbehaving worker nodes can compromise the integrity of the computation result. In this paper, we propose a novel MapReduce framework, Cross Cloud MapReduce (CCMR), which overlays the MapReduce computation on top of a hybrid cloud: the master that is in control of the entire computation and guarantees result integrity runs on a private and trusted cloud, while normal workers run on a public cloud. In order to achieve high accuracy, we propose a result integrity check scheme on both the map phase and the reduce phase. On the other hand, we strive to reduce the performance overhead by reducing the cross-cloud communication and merging sub-tasks. We implement CCMR based on Apache Hadoop MapReduce and evaluate it on Amazon EC2. Both theoretical and experimental analysis show that our approach can guarantee high result integrity in a hybrid cloud environment while incurring non-negligible performance overhead (e.g., when 16.7% workers are malicious, CCMR can guarantee at least 99.52% of accuracy with 33.6% of overhead when replication probability is 0.3 and the credit threshold is 50). Keywords: MapReduce, Integrity Assurance, Hybrid Cloud

1. INTRODUCTION
MapReduce (Dean & Ghemawat, 2008) has become the dominant paradigm for large-scale data processing applications such as web indexing, data mining, and scientific simulation. However, MapReduce applications normally are running on a cluster of hundreds or thousands of computation nodes. Most MapReduce customers cannot afford or do not want to invest in computer clusters of such a large scale. The emergence of Cloud Computing provides an economical alternative for getting a large-scale cluster on demand, thus MapReduce in the cloud has been embraced by the market with enthusiasm. For example, various services such as Amazon Elastic MapReduce and Microsoft Daytona are provided to facilitate the transition of MapReduce applications to the cloud.

However, MapReduce applications running on the cloud suffer from the integrity vulnerability problem: malicious participants can render the overall computation result useless. While the cloud vendors can be trusted and the cloud infrastructure (i.e., the virtualization layer) can be assumed to be secure, the virtual machines and the MapReduce applications installed in the virtual machines cannot be trusted to always return correct results. For instance, (Balduzzi, Zaddach, Balzarotti, Kirda, & Loureiro, 2012) and (Bugiel, Nürnberger, Pöppelmann, Sadeghi, & Schneider, 2011) point out a security vulnerability that Amazon EC2 suffers from: some members of the EC2 community can create and upload malicious Amazon Machine Images (AMIs), which, if widely used, could flood the EC2 cloud with virtual machine instances that contain malicious applications, including MapReduce. The above threat puts a MapReduce customer in a dilemma: using public clouds has economic advantage but incurs the risk of getting wrong computation results; on the other hand, avoiding the public cloud completely (i.e., running everything “in house” or in the private cloud) can guarantee result accuracy, but there will be less economic benefit.

In this paper, we propose Cross Cloud MapReduce (CCMR for short) that combines the benefits of private clouds and public clouds. CCMR overlays the MapReduce framework on top of a hybrid cloud which consists of a private cloud and a public cloud. The master that is in control of the entire computation and guarantees result integrity runs on a private and trusted cloud, while normal workers run on the public cloud and are untrusted. We further introduce a special type of workers (called verifiers) on the private cloud to detect collusive malicious workers on the public cloud. The key rationale of our solution is to retain control and trust “at home”, while delegating the more resource-intensive computations to the public cloud.

We explore the design space of result integrity checking in both phases of MapReduce: the map phase and the reduce phase. We extend the capability of the master to propose the result integrity check mechanism, which combines several integrity assurance techniques (replication (Golle & Stubblebine, 2002), verification (Du, Jia, Mangal, & Murugesan, 2004), and credit-based trust management (Zhao, Lo, & Dickey, 2005)). Due to the different properties of map and reduce phases, CCMR uses the result integrity check on different objects. In the map phase, integrity check is performed on map tasks. In the reduce phase, CCMR factors each reduce task into multiple sub-tasks and applies the integrity check on sub-tasks. In order to improve performance of the reduce phase, we propose the request
**bucketing technique** to further reduce the performance overhead.

Our theoretical simulation (in Section 4.3) shows that when credit threshold $T$ is set to 50 (in the map phase), and replication probability $r$ is set to 0.5, CCMR can guarantee a job error rate of less than 1% when less than half of workers on the public cloud are malicious, and a job error rate of less than 9% when all the workers on the public cloud are malicious. When $T$ is set to 600 (in the reduce phase) and $r$ is set to 0.16, CCMR can guarantee a job error rate of 0% when less than half of workers are malicious, and a job error rate of less than 6% when all the workers are malicious.

Our experiment result shows that CCMR introduces 19% to 83% of delay depending on the replication probability $r$ in the map phase. It also shows that CCMR can introduce 29% of delay on average in the reduce phase when applying the request bucketing technique.

We make the following contributions in this paper: 1) we propose a novel cross-cloud MapReduce architecture that combines the benefits of private clouds and public clouds; 2) we propose a result integrity check mechanism that combines several integrity assurance technique to enhance the result integrity of MapReduce on both the map and reduce phases; 3) we analyze the security of CCMR and quantitatively measure its accuracy and overhead; 4) we implement CCMR based on Apache Hadoop MapReduce and run a series of experiments over the commercial public cloud (Amazon EC2). We show that CCMR is an efficient framework to guarantee high computation integrity.

The rest of this paper is organized as follows. Section 2 describes the system assumptions and attacker model. Section 3 presents the system design of CCMR. Section 4 makes the theoretical analysis in terms of security, accuracy, and overhead. Section 5 describes and analyzes the experiment result. Section 6 discusses the related work, and Section 7 concludes the paper.

### 2. System Assumptions and Attacker Model

#### 2.1 System Assumptions

In CCMR, we assume the private cloud is trusted since it is deployed within the user’s organization. Therefore, the master and the verifiers, which are deployed on the private cloud, are trusted. On the public cloud, we assume the infrastructure provided by the cloud provider, such as the virtualized hardware and network, is trusted. However, we assume the virtual image used by the customer is untrusted. That makes the MapReduce entities running on the public cloud untrusted. Since our paper only focuses on the MapReduce computing, we assume Distributed File System (DFS) of MapReduce is trusted. For example, the integrity of DFS can be guaranteed by the techniques proposed in (Popa, Lorch, Molnar, Wang, & Zhuang, 2011) and (Bowers, Juels, & Oprea, 2009).

In CCMR, the master requires that each worker who runs a task/sub-task submit the hash value of its computation results to the master. We assume the hash value to be consistent with the actual task/sub-task output. Such an assumption can be realized by applying the commitment-based protocol proposed in (Wei, Du, Yu, & Gu, 2009) (i.e., the previous worker commits the task output to the master by the hash value. The later worker who takes the previous task’s output as input will return the input hash value to the master. The master compares the hash values to ensure the consistency of the hash value and the corresponding task output). Finally, we assume that the tasks running on each worker are deterministic. This assumption guarantees that multiple executions of the same task/sub-task by honest workers return the same result.

#### 2.2 Attacker Model

We model the attacker as an intelligent adversary that controls the malicious nodes on the public cloud. It receives and correlates information collected by the malicious nodes and coordinates them to cheat at the right time in order to introduce as many errors as possible to the final result without detection. For example, if the master replicates the same task on two malicious workers, the adversary can instruct them to return the same erroneous results (i.e., to collude) so that simply comparing the results cannot detect the error. We call such malicious workers *collusive workers*.

### 3. System Design

#### 3.1 System Overview and Architecture

CCMR overlays MapReduce on a hybrid cloud consisting of one private cloud and one public cloud, which is shown in Figure 1. The master node and a small number of slave nodes (called *verifiers*) are deployed on the trusted private cloud within the customer’s organization. Other slave nodes (called *workers*) and Distributed File System (DFS) are deployed on the public cloud. According to our assumption, the verifiers, the master and the DFS are trusted, yet the workers are untrusted.

In both the map and the reduce phases, CCMR defines three types of tasks: the *original task*, the *replication task*, and the *verification task*. The original and replication tasks are executed by the workers on the public cloud. The verification tasks are executed by the verifier on the private cloud. The replication task repeats the original task’s work to validate the original task result. Since the replication tasks are executed by the untrusted public cloud worker, the verification tasks can check the replication task result by re-executing the task on the verifier. In the map phase, the replication and verification task completely repeat the original map task’s work. While in the reduce phase, the replication and verification reduce task repeats only part of the original reduce task. Each replication/verification reduce task consists of one or several small portions of original reduce task. Each portion of task is called a *sub-task*.

http://hipore.com/ijcc
evade the detection of the two-layer check. In order to achieve high accuracy, credit based trust management is applied on each worker. The master only accepts a worker’s task/sub-task results when it achieves certain credit threshold.

The task/sub-task execution in CCMR differs from the original MapReduce in both the map and reduce phase. Rather than passively waiting for the worker to ask for task/sub-task, the master of CCMR randomly selects the worker to execute a certain task/sub-task. When a task/sub-task is finished, CCMR requires the worker to return the result. In order to reduce the communication cost, the worker only returns the hash value of the result. Since the replication and verification tasks are only used to evaluate the correctness of the original tasks, the actual result of replication and verification task/sub-tasks will not be stored back to the DFS.

Given the different characteristic of map and reduce phases, we propose different integrity check solutions.

3.2 MAP PHASE INTEGRITY CHECK

CCMR applies two-layer check on each returned original map task result. In the first-layer, CCMR creates a replication task and assigns the task to another worker. The replication task assignment is applied with a technique called hold-and-test, which will be introduced later. When the worker returns the replication task result, CCMR compares the original and replication task results. If the results are not consistent, at least one of the workers are cheating, so CCMR will create a verification task and assign it to a verifier to detect the malicious mapper(s). If the original and replication task results are consistent, CCMR launches the second-layer check. In the second-layer check, CCMR creates a verification task and assigns it to a verifier to verify the consistent results. If the consistent results are different from the verification task result, the two mappers providing the results are all determined as malicious. The reason for the second-layer check is to detect collusive workers. To reduce overhead, CCMR creates replication and verification tasks with certain probability. Each original map task is replicated with replication probability, and each pair of consistent results is verified with verification probability.

Since replication or verification is not performed for every task, there is a possibility that some bad results can evade the detection of the two-layer check. In order to overcome this drawback, CCMR performs the credit based trust management to improve the job result accuracy.

Initially, the master sets the credit for each mapper as zero, and maintains a history cache for each mapper to record the id and result (hash value) of original map tasks the mapper has executed. When a mapper passes one two-layer check, the master increments the credit for this mapper and updates the mapper’s history cache. The actual task result is buffered in the mapper’s local storage before it becomes trusted. When a mapper’s credit achieves certain threshold (called credit threshold), the mapper becomes trusted temporarily. The task results buffered in the mapper’s local storage are accepted by the master in a batch. At the same time, the credit and the history cache of this mapper are reset, and this mapper becomes untrusted again.

The mapper has to earn credit again in order to submit the next batch results to the master. If a mapper fails any two-layer check before achieving credit threshold, it is determined to be malicious and is added to a blacklist. The actual results buffered in its local storage are discarded, and the tasks cached in its history cache will be re-executed.

Figure 2 presents the control flow of CCMR. In the figure, W1 and W2 are two slave workers randomly chosen from the public cloud. The “Arbitrate/Verify task” step is completed by the verifier on the private cloud, and the remaining components in the figure are all performed on the master. Notice that in the figure, instead of assigning the replication and original task simultaneously, the “replication” decision (step 3) is made after W1 returns the original task result R2 (step 2). We call such a technique hold-and-test, and it makes it harder for malicious workers to collude because the adversary cannot predict whether the replication task will be assigned to another collusive worker. A detailed discussion of the benefit of hold-and-test is deferred to Section 4.1.

If the total number of original map tasks in a job is less than the credit threshold, CCMR directly assigns all tasks to verifiers, since the computing workload is not significant. Therefore, the accuracy in this case is still guaranteed. If the total number of original map tasks is large enough, a higher credit threshold would guarantee a higher accuracy, as our theoretic simulation shows (Section 4.3).
3.3 REDUCE PHASE INTEGRITY CHECK

In the reduce phase, the approach presented in Section 3.2 can be directly applied only if the number of reduce tasks is large enough (i.e., bigger than the credit threshold). However, in some applications, the reduce task number is smaller than the credit threshold, even though the computing workload for each reduce task is significant. For instance, the word count application in Section 5.2 contains only one original reduce task. However, this single task will process 2.7M of records (1.07GB of data) in the input and generate 598K of records in the output, and take 262 seconds to complete. In this case, directly verifying the entire reduce task is expensive in terms of computation and communication cost. Therefore, we propose to break down each original reduce task into many sub-tasks and apply two-layer check on each sub-tasks.

In the original MapReduce, the master breaks the job input into multiple blocks. Each map task processes one block and generates the task output in the format of <key, value> tuples, which are sorted by key. Reduce tasks will process the output of map tasks and also generate <key, value> tuples as reduce task output. The output is also sorted by key. Each reduce task only processes certain map output tuples with specific keys, which are determined by the partition function.

Our reduce phase integrity check design is based on the following intuition. We observe that both the map tasks and the reduce tasks outputs are sorted by key. For each key in the reduce output, if we can precisely pinpoint the map output tuples that are related to that key, we can reproduce the portion of the reduce task that is related to that reduce output key. We call each portion of reduce task as sub-task. Therefore, each original reduce task can be divided into multiple sub-tasks, each of which is related to one key. By applying two-layer check to each sub-task, we can guarantee high accuracy of the original reduce task. Here, we temporarily define each sub-task to cover one key. We will extend this concept later for practicality reason.

Our reduce phase integrity check uses the same high-level ideas as the map phase. Each original sub-task returns its result to the master in the form of a hash value (we call each returned sub-task result as a report). The master applies the first-layer (replication) and second-layer (verification) check on each report with replication probability and verification probability, respectively. The generation of a replication sub-task is decided after the report of an original sub-task is returned to the master (hold-and-test). We regulate that an original reduce task result is accepted by the master only when all its sub-tasks pass the two-layer check, which is essentially a credit-based trust management. The credit threshold is the number of sub-tasks in the original reduce task. When the number of sub-tasks in an original reduce task is smaller than the credit threshold, CCMR directly generates a verification reduce task to verify the entire original reduce task.

The above idea only works conceptually. However, in order to make it practical, we need to extend the concept of sub-tasks and reports, and overcome the following three challenges. 1) Creating a sub-task for each key would incur significant overhead because in many cases, the amount of keys in a reduce task can be huge (e.g., 598K keys in the word count application in Section 5.2). 2) The accuracy only relies on the two-layer checks of the sub-task reports submitted by the reducer. If a malicious reducer cheats on some sub-tasks but does not send these reports to the master, the master would have no way to detect the error. 3) The replication and verification sub-tasks should efficiently locate the portion of map task output with the key they are interested in.

We address the first challenge by extending the concept of report and sub-task to cover a range of consecutive keys, instead of just one. In other words, we require each report to cover a range of (instead of only one) consecutive keys. With this improvement, the number of sub-tasks will be reduced.

For the second challenge, CCMR requires that consecutive reports in the original reduce task must overlap in one key. In addition, the first and last report in each original reduce task should cover the first and last key of the task output, respectively. The master will check those requirements when it receives the reports. Since the reduce task result is sorted by the key, this requirement ensures that no key in the output is skipped in the reports. In case that the master does not know the first and last key in the original reduce task output, the master can insert dummy records in the job input data, which will generate reduce result tuples with predictable smallest and largest keys. For example, when the type of the key is integer, the master can insert records with keys Integer.MIN VALUE and Integer.MAX VALUE. When the reduce task is complete, the output can be sanitized by removing the output records related to the dummy input records.

For the third challenge, each map task in CCMR builds a key table to facilitate the record look up in the map task output, as shown in Figure 3. When a sub-task wants to fetch the map output within a certain key range (e.g., Key2 to Key 9) from the map task output file, it will locate the position through the key table of that task. In the original MapReduce, each map task stores the map output file locally, which consists of key-value pairs sorted by key. Notice that the lengths of keys and values vary and multiple key-value pairs can have the same key (e.g., Key 3). For brevity, we call consecutive key-value pairs in the map output file the same as a block. Each record in the key table corresponds to a block in the map output file. It has three fields, indicating the start position of the block, the length of the block, and the length of the block. Since the lengths of keys and values vary, each access to a key on the map output file needs to go through the key table, which has the fixed record length. The key table records are sorted by key, CCMR can apply binary search to look up the records...
within the request key range. However, each key comparison in the binary search needs to fetch the key in the map output file through the key table. When the binary search finds the keys (e.g., key 3 through key 8) between the request range (e.g., key 2 to key 9), the key position and the record length direct the reducer to fetch the portion of map output. Since it is the mapper who creates the key table, a malicious mapper could manipulate its content to fool CCMR. As a defense, CCMR requires each map task to submit the hash value of its key table to the master along with that of task result. The consistency between the hash value and the actual key table can be guaranteed by the commitment-based protocol (Wei, Du, Yu, & Gu, 2009).

Figure 4 shows how CCMR works in the reduce phase of word count application. The word count application calculates the frequency of each word appeared in a collection of text files. For simplicity, our example only has two map tasks (map 0 and map 1) and one original reduce task (reduce 0). As Figure 4 shows, each map task creates a key table (Step 1). When the original reduce task (reduce 0) starts to output (step 3), sub-task reports (e.g., report 1 and 2) are sent to the master sequentially. The report format is <start key, end key, hash value of the output records covered in the key range> (Step 4). Since consecutive reports must overlap in one key (According to the solution of challenge 2), the key “Driver” appears in both report 1 and report 2. When the master receives report 1 (Step 5), it launches the first-layer check by initiating a replication sub-task (with replication probability). The replication sub-task fetches input with a key range of (Apple, Driver) from each map task (map 0 and map 1) through key tables (Step 7). When it completes reducing (Step 8), the replication sub-task sends a report to the master (Step 9), and the master compares the report with the original sub-task report (Step 10). If they are consistent, a second-layer check is performed to verify the consistent results. The verification sub-task will be created if necessary (Step 11).

### 3.4 Request Bucketing

One draw back of reduce phase design in Section 3.3 is that in order to achieve high accuracy, CCMR will generate a large number of sub-tasks. Each sub-task will be executed as an individual reduce task. Such a reduce task needs to connect to each map tasks, locate the map output position, and fetch only a small portion of data. In addition, its setup and tear down will consume a certain amount of resource.

As a result, a big number of sub-tasks can introduce a big performance delay. For example, according to the word count experiment in Section 5.2, the reduce phase will generate 88 replication sub-tasks and six verification sub-tasks. The execution delay is 43%. We hope to merge multiple sub-tasks into one reduce task to improve performance, while without sacrificing the accuracy. Therefore, we propose the request bucketing technique to achieve such goal.
is still performed on each original reduce sub-task. The analysis in Section 4.2 and experiment in Section 5.2 will show that request bucketing technique does not undermine the accuracy of CCMR, while it reduces the execution delay.

4. SYSTEM ANALYSIS

4.1 QUANTITATIVE ANALYSIS ON MAP PHASE

Since the major differences of map and reduce phase in CCMR is the object used to perform two-layer check (task in map phase and sub-tasks in reduce phase), we could use similar model to analyze both phases. In this section we give quantitative analysis on the map phase. In the next section, we will discuss how to adapt the theorem in this section to the reduce phase.

We first analyze the adversary strategy of malicious worker. Based on that, we will perform quantitative analysis on accuracy and overhead.

Adversary Strategy: We denote the malicious worker fraction on the public cloud as $m$. We assume that the adversary controls all malicious workers. In other words, all malicious workers are collusive, and there exists only one collusive group. Assuming that the goal of the adversary is to inject as many errors as possible and yet not reveal the malicious workers, we analyze the strategy of the adversary under CCMR as follows. Suppose a task is assigned to a malicious worker, two cases are possible for the adversary.

Case 1. If the adversary has not seen a similar task (i.e., the one with the same input) before, it has to make a decision on whether to cheat, and remembers the decision, the current task and the returned result. Due to the existence of hold-and-test (Section 3.2), the adversary is not allowed to defer the decision to the time that it sees the replica of the current task. If the decision is not to cheat, the worker is obviously safe (i.e., not to be caught). If the decision is to cheat, the malicious worker can survive the first-layer check only when either the current task is not replicated, or the replica of the current task is assigned to another malicious worker.

Case 2. If the adversary has seen a similar task before, it is assured that the current task is a replication task. It can simply ask the worker to take the same action for the current task as the one it has seen before. In this case, it is guaranteed that the malicious worker will survive the first-layer check.

Since in case 2 the adversary just follows its decision made previously in case 1, the risk of revealing a malicious worker is essentially determined by the adversary’s decision in case 1. Because the master controls task assignment and replication in a randomized manner, the adversary in case 1 cannot predict whether cheating at the current task is safe or not. On the other hand, since the master constantly applies the two-layer check on tasks in a randomized manner, the adversary cannot tell whether cheating at the current task has a smaller chance of detection than cheating at other tasks. Therefore, the only thing the adversary can do in case 1 is to make a random guess/predict in terms of whether cheat can be detected. We model the adversary’s decision making behavior in case 1 as a random variable, cheat probability $c$. Note that adversaries who cheat rarely (e.g., only cheat once in hundreds of tasks) can still fit in our model because we can set $c$ as a small value close to 0.

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m$</td>
<td>malicious worker fraction</td>
<td>The fraction of malicious workers on the public cloud.</td>
</tr>
<tr>
<td>$c$</td>
<td>cheat probability</td>
<td>The probability that the adversary decides to cheat in Case 1 of the Adversary Strategy.</td>
</tr>
<tr>
<td>$r$</td>
<td>replication probability</td>
<td>The probability that an original task/sub-task is replicated.</td>
</tr>
<tr>
<td>$v$</td>
<td>verification probability</td>
<td>The probability that consistent task/sub-task results are verified.</td>
</tr>
<tr>
<td>$T$</td>
<td>credit threshold</td>
<td>The credit a mapper/reducer has to achieve to make its batch of results to be accepted by the master.</td>
</tr>
<tr>
<td>$L$</td>
<td>survival length</td>
<td>The expected number of batches a malicious worker can submit to the master before it is detected.</td>
</tr>
<tr>
<td>$E$</td>
<td>batch error number</td>
<td>The expected number of incorrect task/sub-task results in one accepted batch.</td>
</tr>
<tr>
<td>$e$</td>
<td>batch error rate</td>
<td>The fraction of incorrect results in one batch of results.</td>
</tr>
<tr>
<td>$J$</td>
<td>job error rate</td>
<td>The ratio of incorrect results number to the total results number in one job.</td>
</tr>
<tr>
<td>$O$</td>
<td>overhead</td>
<td>The expected number of extra executions for each task/sub-task performed on the public cloud.</td>
</tr>
<tr>
<td>$V$</td>
<td>verifier overhead</td>
<td>The expected number of extra executions for each task/sub-task performed on the private cloud.</td>
</tr>
<tr>
<td>$K$</td>
<td>task key number*</td>
<td>The number of keys (records) generated by an original reduce task.</td>
</tr>
<tr>
<td>$S$</td>
<td>sub-task key number*</td>
<td>The number of keys covered by a sub-task.</td>
</tr>
<tr>
<td>$R$</td>
<td>report number*</td>
<td>The number of reports an original reduce task sends to the master.</td>
</tr>
<tr>
<td>$B$</td>
<td>bucket size*</td>
<td>The maximum number of sub-task requests contained in a bucket.</td>
</tr>
</tbody>
</table>

* is only applicable to reduce phase

Table 1 CCMR System Setting Parameters
We define several metrics to measure the accuracy and overhead of CCMR for both the map and reduce phases, and summarize the parameters in Table 1. Notice that the first eleven items are applicable to both phases. The metrics in the map phase is with regard to the tasks. When we adapt such metrics into reduce phase analysis, we only need to substitute the “task” with “sub-task”. The last four items are only applicable to the reduce phase. We will discuss them in Section 4.2. We perform a series of probabilistic analysis and present our analysis result in Theorem 1. This theorem can also be applied to the reduce phase by replacing the “task” with “sub-tasks”. The argument of such substitution is provided in Section 4.2.

Theorem 1: Assuming that the assignment of tasks/sub-tasks is uniformly distributed across all workers on the public cloud, and the detected malicious workers are not added to the black list, the probability for a malicious mapper/reducer to survive after executing n original tasks is

\[ S_n = \sum_{i=0}^{n} \binom{n}{i} (1-c)(1-r)^{i} (crm(1-v))^i \]  

The survival length of a malicious mapper/reducer is

\[ L = S_n / (1 - S_n) \]

The batch error number is

\[ E = \sum_{k=1}^{r} \binom{k}{i} T^{i} (1-c)^{i} (crm(1-v))^{i} \]  

The batch error rate is

\[ e = E / T \]

The job error rate is

\[ J = me \]

Let

\[ U = 1 - (1 - rc - rv + rcv + rcm - rcmv)rc(1 - m + mv)m \sum_{j=1}^{r} S_j \]

The overhead for each task/sub-task is

\[ O = (m + r - mr - m(l - r)S_{r-1} + mr(1-v)(1-c + cm)(rc - rcm + rcmv)\sum_{j=1}^{r} S_j) / U \]

The verifier overhead for each task/sub-task is

\[ V = ((1-m)(crm+v-vcm)S_{r-1} + mr(c+v-vc - cm + cmv)S_{r-1} + rcm(1-m+mv)(1 + rv - rvc)\sum_{j=1}^{r} S_j) / U \]

The derivation of \( S_n \) is the foundation of Theorem 1. \( S_n \) is the probability summation of all permutations on \( n \) independent events. Each event falls into one of three cases. For each original task/sub-task, if executed on a malicious worker, it can pass the two-layer check in one of the following three cases: 1) the worker does not cheat; 2) the worker cheats, but the task/sub-task is not replicated; 3) the worker cheats, the task/sub-task is replicated, but the replication task/sub-task is assigned to another malicious worker and their results are not verified. The probabilities of the above cases are \( (1-c) \), \( c(1-r) \), and \( crm(1-v) \), respectively. By summing up the probabilities of different permutations of above three cases, we get (1). The complete proof of Theorem 1 can be found in the Appendix.

4.2 Analysis on Reduce Phase

The difference between map phase and reduce phase is that the reduce phase performs the two-layer check to the original reduce sub-tasks instead of tasks. Although request bucketing causes multiple sub-tasks to be replicated or verified in a batch, we can still adapt the model of map phase (Section 4.1) for the following reasons.

Firstly, the way that the master performs two-layer checks to each original reduce sub-task is the same as it does to the tasks in the map phase. Second, even though sub-tasks are merged together to execute, the assignment of sub-task requests to the buckets and the assignment of buckets to the workers are all randomized, which is equivalent to the effect of assigning each sub-task request to workers randomly. Third, the adversary strategy in Section 4.1 is also applicable to the reduce phase. If the reducer executing the original reduce task is malicious, when it sends an original sub-task report to the master, it cannot predict whether the replication sub-task will be assigned to its colluder. Therefore, the malicious reducer still needs to make a random guess on each original sub-tasks.

The reduce phase design in Section 3.3 (without request bucketing) is only a special case where each bucket has a capacity of one. Once a sub-task request is assigned to the bucket, a replication/verification reduce task is created immediately. Hence, the above arguments are also applicable to the design without request bucketing technique.

Therefore, to adapt the theoretical analysis result to the reduce phase, we only need to change the definition in Table 1 by replacing the “tasks” with “sub-tasks”. Theorem 1 can be also adapted by substituting the “tasks” with “sub-tasks”.

We introduce four parameters specific for the reduce phase, shown in Table 1 (The last four items). The total number of reports a master receives from one original reduce task is \( R = \left\lfloor K / S \right\rfloor \). In our design, an original reduce task result is accepted by the master when its credit achieves credit threshold. In other words, we can adjust the parameters to have credit threshold \( T \) to be equal to the report number (i.e., \( T = R \)). When \( K \) is big enough, we can set \( T \) to a big value by adjusting \( S \) to ensure high accuracy. For example, in the word count application in Section 5.2, the single original reduce task output contains 598,000 keys. In this case, in order to set \( T \) to 600 to ensure high accuracy, we can set \( S \) as \( \left\lfloor K / R \right\rfloor = 997 \).

When request bucketing is applied, the two-layer check is still applied to each sub-task. Also, the assignment of sub-tasks is still random. Therefore, request bucketing will not undermine the CCMR accuracy. In addition, the number of sub-tasks is not reduced with the introduction of request bucketing. The only thing that is affected is the number of

http://hipore.com/ijcc 32
replication/verification reduce tasks. Suppose the number of replication/verification reduce sub-tasks generated in a job is $R$, without request bucketing, the replication/verification reduce task number is $\bar{R}$, since each sub-task will be executed in an independent task. However, with request bucketing, the number of replication/verification reduce task will be $\lceil \bar{R} / B \rceil$.

4.3 Simulation Result

We present several simulation results based on Theorem 1 to analyze the relationships among accuracy, overhead and other system parameters.

We first simulate the job error rate under different system parameters in Figure 5(a). The four curves show that when other parameters ($c$, $v$, $r$ and $m$) are fixed, increasing credit threshold $T$ would reduce the job error rate $J$, and when $T$ is greater than 200, $J$ is close to 0 for any parameter combinations in the figure. Moreover, when $T$ and other parameters are fixed, $J$ will be increased if malicious worker fraction $m$ is increased or the replication probability $r$ is decreased. For example, when $T$ is 50 and $r$ is 0.5, $J$ increases from near 0 to 0.06 when $m$ increases from 0.5 to 1.0; when $T$ is 50 and $m$ is 1.0, $J$ increases from 0.06 to 0.15 when $r$ drops from 0.5 to 0.3.

Figure 5(b) shows the relationship between cheat probability $c$ and job error rate $J$ with fixed $T$ and $v$. According to the simulation, when $T$ is 50, $r$ is 0.5, and $m$ is 0.5, the maximum $J$ an adversary can achieve is less than 0.01. When $m$ is 1.0, setting $r$ as 0.5 can limit $J$ to less than 0.09.

The simulation also shows an interesting tradeoff between $c$ and $J$: if $c$ is too big, the malicious worker would be detected easily and thus its injected errors are rejected, resulting in a smaller $J$; if $c$ is too small, the number of injected errors is reduced, also resulting in a smaller $J$.

Figure 5(c) shows the relationship between $c$ and $J$ when $T$ is 600, $v$ is 0.07, and $r$ is 0.16. With this configuration, even under the most extreme case where $m$ is 1.0, the maximum $J$ the adversary can achieve is less than 0.06; when $m$ is no larger than 0.5, the maximum $J$ is close to 0.

Figure 5(d) shows the relationship between cheat probability $c$ and survival length $L$ when $T$ is 50 and $v$ is 0.15. We can see that $L$ is generally very small when $c$ is bigger than 0.02, which means that a malicious worker cannot survive CCMR checks for a very long time. Our experiment in Section 5.1 and Figure 6 confirms this observation. However, $L$ increases exponentially when $c$ decreases from 0.02 to 0, which suggests that CCMR cannot remove very low-profile malicious workers (those that rarely cheat) quickly, but since such workers inject very few errors at the same time, CCMR can still guarantee very low job error rate in that case.

Figure 5(e) shows the tradeoff between job error rate $J$ and overhead $O$, and Figure 5(f) shows the tradeoff between job error rate $J$ and verifier overhead $V$, given different credit threshold $T$. For each curve in the figures, the top-left most point corresponds to the setting where $T$ is 0, and the bottom-right most point corresponds to the case where $T$ is 600. The difference of $T$ values between adjacent points on each curve is 50. The figures show that when $T$ is small (e.g., 50), a higher value of $r$ results in a lower job error rate and higher overhead and verifier overhead. When $T$ is big

![Figure 5. Simulation of CCMR Analysis Model](http://hipore.com/ijcc)
enough (e.g., greater than 200), different values of \( r \) do not make much difference in job error rate. However, a smaller value of \( r \) would bring a smaller overhead and verifier overhead limit. We find that on each curve, the points become denser with the increase of \( T \) and eventually concentrate to their outmost limits. This suggests that when \( T \) is big enough (e.g., bigger than 200), increasing \( T \) further would bring neither additional accuracy benefit, nor additional overhead or verifier overhead cost.

We should point out that Theorem 1 assumes that malicious worker fraction \( m \) is constant, i.e., detected malicious workers are not eliminated. However, in our real implementation, detected malicious workers are eliminated, which will cause fewer errors. As a result, task/sub-task reschedule will be reduced, and eventually the overhead and verifier overhead should be lower than the simulation result.

4.4 Communication Cost Analysis

In order to reduce the cross-cloud communication cost, we only deploy the cross-cloud nodes on the public cloud. Such a deployment avoids DFS data synchronization across clouds. In addition, it reduces the cross-cloud communication incurred by MapReduce tasks. Since each mapper fetches input from DFS and stores task output to its local storage, the only major cross-cloud communication in the map phase happens when a verification task fetches input data from the DFS. Since each reducer fetches input from the mappers’ local storage, and only the original reduce task writes its output to the DFS (According to Section 3.1), the only major cross-cloud communication in the reduce phase happens when a verification reduce task fetches input data from mappers on the public cloud. Since the number of map/reduce verification tasks is usually very small compared to the number of original and replication task, such cross-cloud communication is not significant.

Other sources of cross-cloud communication includes task scheduling instructions from the master to workers and the task results (hash value) returned from workers to the master. However, network traffic caused by such communication messages is not significant due to their small sizes (e.g., a hash value of a task result contains only a few bytes).

5. Experimental Result

We implement a prototype system based on Hadoop MapReduce and deploy it across our private cloud and Amazon EC2. The experiment environment consists of the following entities: a Linux server (2.93 GHz, 8-core Intel Xeon CPU and 16 GB of RAM) is deployed on a private cloud, running both the master and the verifier. Twelve Amazon EC2 micro instances are running as slave workers (Amazon Linux AMI 32-bit, 613 MB memory, Shared ECU, Low I/O performance).

We perform experiments on map and reduce phase separately to measure the job error rate, overhead, verifier overhead, and performance overhead. To compare the performance overhead, we set the baseline as a standard MapReduce cluster consisting of thirteen nodes deployed on Amazon EC2. Each node is a micro instance. Out of the 13 nodes, one is running as the master, and the other 12 nodes running as workers.

5.1 Map Phase

We measure job error rate, overhead and verifier overhead of CCMR by running a word count MapReduce job (Section 3.3) in an environment with malicious Map Reduce workers. We simulate such malicious workers by implementing the adversary’s strategy described in Section 4.1. The word count job consists of 800 map tasks and one reduce task. We fix \( T \) and \( v \) and vary other parameters with different value combinations.

The experiment result in Table 2 indicates that in all parameter combinations, CCMR can keep a very low job error rate. Overall, the maximum job error rate is 2.25% and the minimum is 0. The changing trend of experiment result is consistent with the simulation result in Section 4.3. For example, when \( m \) and \( c \) are fixed (\( m \) is 0.167 and \( c \) is 0.1), job error rate drops from 0.48% to 0 when \( r \) increases from 0.3 to 1.0. When \( m \) and \( r \) are fixed (\( m \) is 0.5 and \( r \) is 1.0), the job error rate drops from 0.14% to 0 when \( c \) increases from 0.1 to 1.0. On the other hand, a higher value of \( r \) incurs a higher overhead and verifier overhead. For example, when \( r \) is 1.0, the average overhead is 112%, and when \( r \) is 0.3, the average overhead is 41%. We note that the experiment overhead and verifier overhead is lower than the simulation result in Figure 5(c) and (f), respectively. This fact confirms our prediction in Section 4.3: Since Theorem 1 assumes \( m \) as a constant value, its estimation of overhead and verifier overhead should be higher than the experiment result.

We observe that in each of the 18 parameter combinations, CCMR is able to eliminate all malicious workers during the execution of the word count job. In Figure 6, we show three representative combinations in terms of how soon each malicious worker is detected and thus removed. We could see that under the first two combinations, CCMR can remove all malicious workers very quickly (within less than 15% of the total job execution time). Under the third combination, the malicious workers are very stealthy (cheat with a probability of 10%) and the replication frequency is low (30%), but CCMR can still remove all six malicious workers before 50% of the job has finished. Such observations suggest that CCMR is effective.
Table 2  Accuracy and Overhead of Wordcount Application with Map Phase Integrity Check

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Baseline</th>
<th>v=0.15, T=50</th>
<th>v=0.15, T=50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>r=0.3</td>
<td>r=0.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Running time(s)</td>
<td>Extra running time (%)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1728</td>
<td>2069</td>
</tr>
</tbody>
</table>

Table 3  Performance of Map Phase Integrity Check

<table>
<thead>
<tr>
<th>Application</th>
<th>Job No.</th>
<th>Baseline</th>
<th>CCMR without request bucketing</th>
<th>CCMR with request bucketing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word Count</td>
<td>1</td>
<td>979</td>
<td>1398</td>
<td>43%</td>
</tr>
<tr>
<td>20 News Letters</td>
<td>1</td>
<td>517</td>
<td>983</td>
<td>90%</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>331</td>
<td>482</td>
<td>45%</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>210</td>
<td>221</td>
<td>5%</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>85</td>
<td>63</td>
<td>-25%</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>161</td>
<td>143</td>
<td>-11%</td>
</tr>
</tbody>
</table>

Ave. Delay (%)* | --- | 46% | 29% |

* Average delay does not count the direct verification job (i.e., job 4 and 5 in the 20 news letters application).

Table 4  Performance of Reduce Phase Integrity Check

5.2 REDUCE PHASE

To measure the reduce phase, we set the replication probability r as 0.16, the verification probability v as 0.07, and the credit threshold T as 600. We set both the replication and verification bucket size B as 10 when applying request bucketing technique.

Our accuracy test shows that such a configuration guarantees 0 job error rate when m is 0.5 and c changes among 0.1, 0.5 and 1.0, which is consistent with our simulation in Figure 5 (c). Given the space limit, we only present the performance experiment result in this section. We use two applications to measure the performance overhead of reduce phase integrity check: word count and mahout twenty newsgroups classification (The Apache Software Foundation, n.d.). For a similar reason as the map phase, we introduce neither map integrity check nor malicious nodes in this performance test.

To measure the performance gain from the request bucketing technique, we perform two sets of experiments: the one without using the request bucketing technique and the one using such technique. The experiment result is shown in Table 4. The word count job is the same job described in Section 5.1, which consists of 800 map tasks and one reduce task. We compare the running time of CCMR with baseline. On average, CCMR without request bucketing takes 1,398 seconds to finish the job. It produces 88 replication reduce tasks and six verification reduce tasks. Compared with the standard MapReduce, which takes 979 seconds to finish the same job (we enable the combine phase to accelerate the execution), the execution delay is 43%. When we apply the request bucketing to the reduce phase, the number of replication and verification tasks drops
to nine and one, respectively; the execution time is reduced to 1,263 seconds and the execution delay is reduced to 29%. We attribute the reduced execution delay to the reduction of number of replication/verification tasks.

We also run the Mahout twenty newsgroups classification example on CCMR. This application consists of five jobs. Each of the first three jobs produces more than 100,000 keys. Hence, CCMR sets the credit threshold $T$ to 600. The last two jobs produce less than 600 keys, so CCMR directly generates a verification reduce task for each job. The total execution time under CCMR without request bucketing is 1,892 seconds. Compared to 1,304 seconds on the baseline, the execution delay is 45%. When applying request bucketing, the execution time is reduced to 1,556 seconds. The execution delay is reduced to 19%. It is interesting to notice that the CCMR execution times of the last two jobs are shorter than the baseline. This is because the master in CCMR is executed on the private cloud, which has more computation power than a micro instance on the public cloud. When evaluating the average slow down incurred by CCMR, we exclude these two special jobs.

Overall, request bucketing plays an important role in boosting the performance. Based on the execution time of word count and the first three jobs of 20 newsletters applications, the average execution delay of CCMR without request bucketing is 46%. When request bucketing is applied, the average execution delay is reduced to 29%.

### 6. RELATED WORK

Several existing solutions have been proposed using replication sampling, and verification techniques to address result integrity problems in other distributed environments such as P2P Systems and Grid Computing. Golle et al. (Golle & Stubblebine, 2002) propose to guarantee correctness of the distributed computation result by duplicating computations. Zhao et al. (Zhao, Lo, & Dickey, 2005) proposed a sampling based idea of inserting indistinguishable Quizzes to the task package, which is going to be executed by the untrusted worker and verify the returned result for those Quizzes. Their simulation result shows by combining reputation system, Quiz approach gains higher accuracy and lower overhead than replication-based approach. However, suggested by their simulation, the reputation accumulation is a long-term process so that in order to accumulate reliable reputation, it takes as many as $10^3$ tasks. Du et al. (Du, Jia, Mangal, & Murugesan, 2004) proposed to insert several sampled tasks to the task package, and check the sampled task returns using Merkle-tree based commitment technique. The analysis in the paper showed that in order to detect error from a malicious worker who cheats with low probability such as 0.1, it takes more than 75 samples to be inserted to each worker.

For MapReduce, Wei et al. (Wei, Du, Yu, & Gu, 2009) proposed an integrity assurance framework SecureMR to enforce the commitment protocol and the verification protocol. SecureMR employs task duplication to defeat collusive workers. The design difference from our paper is that the number of duplication task for each original task is non-deterministic. Such an approach guarantees 90% of detection rate in defeating periodical collusive attacker with 40% of duplication rate when the malicious worker fraction is below 0.15 and malicious cheat probability is 0.5. (According to (2) in (Wei, Du, Yu, & Gu, 2009)) However, (2) in (Wei, Du, Yu, & Gu, 2009) also shows that when malicious worker fraction is 0.5, malicious cheat probability is 0.1, 40% of duplication rate can achieve only 25% of detection rate. The maximum detection rate SecureMR can achieve under this environment setting is 80%, with a duplication rate more than 500%. Wang et al. (Wang & Wei, 2011) proposed the VIAF framework that uses full replication and non-deterministic verification. Such approach eliminates all non-collusive workers and removes collusive worker with certain probability. However, their work does not consider practical factors when deployed on a real cloud, such as cross-cloud communication. In addition, both above works cannot handle the case where the reduce task number is very small (e.g. only one reduce task). This paper is extended from the conference paper (Wang, Wei, & Srivatsa, 2013). Based on the original CCMR design, we improve the reduce phase performance by proposing request bucketing techniques. Our experiment result shows that the request bucketing technique reduces the average execution delay from 46% to 29%.

### 7. CONCLUSION

We propose a novel framework, CCMR, which overlays MapReduce on top of a hybrid cloud to offer high result integrity. Based on such framework, we propose the result integrity check scheme in order to boost the accuracy meanwhile to reduce the delay. Our theoretical analysis and experimental result suggests that CCMR can achieve low job error rate while introducing non-negligible performance overhead.

### 8. ACKNOWLEDGEMENT

This material is based upon work supported by the U.S. Department of Homeland Security under grant Award Number 2010-ST-062-000039. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the U.S. Department of Homeland Security.

### 9. APPENDIX

**Proof of Theorem 1:**

$S_n$, the probability of a malicious mapper/reducer to survive after executing $n$ original tasks/sub-tasks is the probability summation of all permutations on $n$ independent events. Each event should fall into one of the below three cases.
1). The worker does not cheat. The probability in this case is \((1-c)\);
2). The worker cheats, but the task/sub-task is not replicated. The probability in this case is \(c(1-r)\);
3). The worker cheats, and the task is replicated. However, the other worker executing the replication task/sub-task can collude with it. When the consistent results are returned to master, the verification task is not invoked. The probability in this case is \(rcm(1-v)\).

<table>
<thead>
<tr>
<th>Condition 1</th>
<th>Condition 2, given condition 1 is satisfied</th>
<th>Condition 3, given condition 2 is satisfied</th>
<th>Category Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>The task/sub-task (t) is assigned to a malicious worker (z), which survived in the current batch</td>
<td>(t) is not replicated.</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>(z) does not cheat.</td>
<td>(t) is replicated, but (z) does not cheat.</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>(t) is replicated but not verified. (z) cheats. But the replication task/sub-task is executed by another malicious worker.</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>The task/sub-task (t) is assigned to a malicious worker (z), which does not survive in the current batch</td>
<td>(t) is not the last task/sub-task in the current batch (i.e., (t) is the one being detected)</td>
<td>(t) is not replicated.</td>
<td>4</td>
</tr>
<tr>
<td>(z) cheats. The results are not detected.</td>
<td>(t) is replicated. (z) does not cheat. The results are not verified.</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>(t) is replicated. (z) does not cheat. The results are verified.</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(t) is the last task/sub-task in the current batch (i.e., the one being detected).</td>
<td>--</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>The task/sub-task (t) is assigned to a benign worker (z).</td>
<td>(t) is replicated. The corresponding replication task is assigned to a malicious worker. The malicious worker cheats.</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>(t) is not replicated.</td>
<td>(t) is replicated. The corresponding replication task returns the same result as the original one.</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td>11</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 5 Categories of Tasks based on Assignment Conditions.**

<table>
<thead>
<tr>
<th>Category Number</th>
<th>Probability</th>
<th>Workload (W)</th>
<th>Verifier overhead (V)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(P_1 = n \sum_{j=1}^{1} \frac{1}{T} S_{j,c}(1-r))</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>(P_2 = n \sum_{j=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>(P_3 = n \sum_{j=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>(P_4 = m \sum_{j=1}^{1} \sum_{i=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>1+W</td>
<td>V</td>
</tr>
<tr>
<td>5</td>
<td>(P_5 = m \sum_{j=1}^{1} \sum_{i=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>2+W</td>
<td>V</td>
</tr>
<tr>
<td>6</td>
<td>(P_6 = m \sum_{j=1}^{1} \sum_{i=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>2+W</td>
<td>V</td>
</tr>
<tr>
<td>7</td>
<td>(P_7 = m \sum_{j=1}^{1} \sum_{i=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>(P_8 = m \sum_{j=1}^{1} \sum_{i=1}^{1} \frac{1}{T} S_{j,c}(1-c))</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>(P_9 = (1-m)rmc)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>(P_{10} = (1-m)rmc)</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>(P_{11} = (1-m))</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 6 Probability, Workload and Verifier Overhead of Each Category.**
By summing up the probability of different permutations of above three cases on n independent events, we have

\[ S_r = \sum_{j=0}^{\infty} \sum_{i=0}^{n-j} \binom{n-j}{i} (1-c)^i (c(1-r))^{j-i} (crm(1-v))^{j-i} \]  

Setting \( n=T \), we have \( S_T \), the probability that a malicious mapper/reducer submit a batch of task/sub-task to a master. The probability of each malicious worker can submit exactly k batches (i.e., not detected in first k batches but detected on the (k+1)th batch) is

\[ (S_r)^k \cdot (1-S_r) \]

The expected number of batches a malicious worker can submit (batch error number) is therefore.

\[ L = \sum_{i=0}^{\infty} i \cdot (S_r)^i (1-S_r) = \frac{S_r}{1-S_r} \]

In a batch with credit threshold T, The probability that exactly k out of T tasks return erroneous results but are not detected is as follows. It consists of (T-k) events that exactly k out of T tasks return erroneous results but are not detected as follows. It consists of (T-k) events that exactly k out of T tasks return erroneous results but are not detected and k events that the worker cheats but without detection.

\[ \Delta_k = \sum_{i=0}^{\infty} \binom{T}{k} i (1-c)^i (c(1-r))^{j-i} (crm(1-v))^{j-i} \]

The expected number of tasks returning erroneous results in a batch (batch error number) is therefore

\[ E = \sum_{k=1}^{T} (k \times \Delta_k) \]

\[ = \sum_{k=1}^{T} \sum_{i=0}^{\infty} \binom{T}{k} i (1-c)^i (c(1-r))^{j-i} (crm(1-v))^{j-i} \]

The batch error rate by definition is therefore

\[ e = \frac{E}{T} \]

Since the assignment of tasks/sub-tasks on the public cloud is uniformly distributed on all workers. And the detected malicious workers are not added to the blacklist, we can assume ratio m of workers on the public cloud submit erroneous results to the master, the batch error rate for those erroneous results is e. We have total error rate in a job (job error rate) is

\[ J = m \times e + (1-m) \times 0 = me \]

In order to calculate the overhead and the verifier overhead, we divide the original tasks/sub-tasks into 11 different categories based on different conditions the task/sub-task may encounter, as shown in Table 5. We summarize the probability, workload and verifier overhead of each category in Table 6. Here we define workload as the number of executions each task/sub-task has to run on the public cloud, marked as \( W \). It includes the original task/sub-task execution and the task/sub-task overhead. Therefore, we have \( W = 1 + O \), where 1 corresponds to the original task/sub-task and O corresponds to the overhead.

Since the categories summarized in Table 5 are mutual exclusive and exhaustive, we have \( \sum P_i = 1 \).

We calculate the expected workload by combining the probability and workload under each category.

\[ W = P_1 + (1 + W) P_2 + (2 + W) (P_3 + P_4) + 2(P_5 + P_6 + P_7 + P_8 + P_9) \]

We calculate the expected verifier overhead by combining the probability and verifier overhead under each category.

\[ V = vP_1 + vP_2 + vP_3 + vP_4 + P_5 + P_6 + P_7 + vP_9 \]

By reorganizing the formula and replacing \( P_i \) with the value in Table 6, we have,

\[ O = (m+r-mr-m(1-r))S_r + \]  

\[ mr(1-v)(1-c+cm)(rc-rc+rcm)) \sum_{i=0}^{\infty} S_i + U \]

\[ V = (1-m)cm + v - vmc + mr(c+v-cm+cmv)S_{i-1} + 
+ rcm(1-m+mv)(1+rv-rvc) \sum_{i=0}^{\infty} S_i + U \]

, where

\[ U = 1 - (1 - rc + rvc + rcm - rcmv)(rc(1-m+mv)m \sum_{i=0}^{\infty} S_i) \]

10. References


Authors

Yongzhi Wang received his BS and MS degree in School of Computer Science and Technology from Xidian University, China in 2004 and 2007, respectively. He is currently a PhD student in School of Computing and Information Sciences from Florida International University, USA. His research interests include big data, cloud security and outsourced computing security. Before starting his PhD program, he worked for SPSS and IBM as a software engineer.

Jinpeng Wei received a PhD in Computer Science from Georgia Institute of Technology, Atlanta, GA in 2009. He is currently an assistant professor at the School of Computing and Information Sciences, Florida International University, Miami, FL. His research interests include malware detection and analysis, information flow security in distributed systems, cloud computing security, and file-based race condition vulnerabilities. He is a member of the IEEE and the ACM.

Mudhakar Srivatsa is a research scientist in the Network Technologies Department at the IBM T. J. Watson Research Center. He received his Ph.D. degree in computer science from Georgia Tech. His research interests primarily include network analytics and secure information flow. He serves as a technical area leader for Secure Hybrid Networks research in International Technology Alliance in Network and Information Sciences, a research consortium formed from twenty three US and UK industrial and academic members. He also serves as a principal investigator for modeling and analyzing co-evolving networks in the Network Science Collaborative Technology Alliance.

http://hipore.com/ijcc
Implementación y Evaluación Empírica de un Herramienta de Depósito de Aplicaciones de Nube de Infraestructura (IaaS)

Américo Sampaio, Roberto Costa Jr, Nabor C. Mendonça, Ricardo Hollanda Filho

1Universidade de Fortaleza (UNIFOR), 2E-NOVAR Soluções Tecnológicas

{americo.sampaio,nabor}@unifor.br, {roberto,ricardo.luiz}@enovar.com.br

Abstract

Cloud computing is becoming mainstream in software development, with many organizations now considering migrating their applications to the cloud. However, the task of deploying applications in infrastructure-as-a-service (IaaS) clouds is generally daunting due to a number of manual steps necessary to configure all application components as a set of virtual machine images. This work presents a tool, called TREX CLOUD, which greatly simplifies the task of deploying fully functional Web applications in IaaS clouds. With TREX CLOUD, users can easily select a set of files that contain the application components (e.g., .war files, database backup files, context files), which are then used by the tool to automatically deploy and configure the whole application stack in a supported IaaS cloud provider. An empirical study, in which nine users were asked to deploy two different Java web applications in the Amazon cloud using three different deployment tools, shows how TREX CLOUD enables a significant reduction in deployment effort (over 90% in the best case) when compared to existing state-of-the-practice approaches.

Keywords: cloud computing, infrastructure-as-a-service, deployment tool, empirical assessment

1. Introduction

Cloud computing is a recent computing paradigm that is transforming the way we build, deploy and manage software. The benefits of being able to easily provision and re-size the infrastructure used by an application, leveraged by a pay-per-use resource consumption model, are motivating many organizations to adopt infrastructure-as-a-service (IaaS) clouds. 

One popular use for IaaS clouds is to deploy modern web applications such as Java Enterprise Edition (EE) applications (Hajjat et al., 2010) (Zhang, Cheng & Boutaba, 2010). To be able to deploy this type of application in the cloud, one has to pack all application components (e.g., load balancer, web server, application server, database server, and business data) as a collection of virtual machines (VMs) images. In order to get the complete application “up and running”, a set of manual steps is necessary, such as creation of VM images, installation of the required software components in each VM as well as configuration of the components’ inter-dependencies (e.g., to configure the application server with the IP address of the database server). Besides, each IaaS cloud provider usually offers its own (often proprietary) deployment solutions, which tend to make application deployment in different clouds a hard and labor-intensive task. For example, Amazon offers a proprietary image format for packaging application components called Amazon Machine Image (AMI) (AMI, 2013). In case the organization decides to move its applications from Amazon to another IaaS provider (say, Rackspace), little reuse can be achieved and the deployment process has to be mostly re-done from scratch.

To tackle some of the difficulties discussed above, over the last few years some tools have been proposed specifically aimed at facilitating the IaaS cloud deployment task. Most of those tools, e.g., Chef (2013), Heroku (2013) and Rightscale (2013), have a broad scope and allow the deployment of any kind of application in the cloud. In general, those tools are difficult to use as they are based on “low-level” scripting languages and therefore are more recommended for experienced system administrators. On the other hand, there are tools with a narrower scope, such as Amazon Elastic Beanstalk (Beanstalk, 2013), which focuses on providing support for the deployment of web-based applications in the Amazon EC2 cloud. However, that tool still suffers from two fundamental limitations: (i) it offers no support for automatic database configuration, which has to be done manually by the tool user; and (ii) it is a provider-specific solution and thus cannot be used to deploy applications in different IaaS clouds.

This paper presents the TREX CLOUD tool, which allows the deployment and configuration of complete Web applications (including all components, from presentation logic to database) in different IaaS clouds in a simplified manner, without the need to execute manual and complex procedures. The tool currently supports two IaaS cloud providers (Amazon and Rackspace) and offers extensibility mechanisms to programmatically introduce new ones. An empirical evaluation of the tool, conducted in an industrial setting, shows promising results, indicating that the tool enables users with varied IT skills and backgrounds to quickly (in a matter of a few minutes) deploy and run different Web applications in the Amazon cloud. This represents a significant reduction in deployment effort (over 90% in the best case) when compared to Amazon’s own deployment tools (i.e., Amazon EC2 Dashboard and Amazon Elastic Beanstalk).

The remainder of the paper is structured as follows. Section 2 provides the required background on IaaS cloud deployment and explains the motivation for our work.
Section 3 analyzes related deployment approaches. Section 4 describes the TREXCLOUD tool in more details. Section 5 reports on the method and results of the empirical study conducted to evaluate the tool. Section 6 provides a discussion and, finally, Section 7 offers some conclusions and directions for future work.

2. Problem and Motivation

Cloud computing services are generally categorized in three types (Armbrust et al., 2010) (Zhang, Cheng & Boutaba, 2010). Infrastructure-as-a-Service (IaaS) clouds, such as Amazon EC2 and Rackspace, offer computational (e.g., virtual machines), network and storage resources to customers. Other types are Software-as-a-Service (SaaS) and Platform-as-a-Service (PaaS) clouds, which offer applications (e.g., Salesforce.com) or development and service platforms (e.g., Google App Engine), respectively. This work focuses on offering a tool to support the deployment of web applications in IaaS clouds.

Typically, the process of deploying a web application in an IaaS cloud can be organized in two main steps: (i) creating images; and (ii) installing and configuring software components. The following subsections describe each of those steps in more details, along with a discussion of the need for a new tool to support them.

2.1 Creating Images.

This step corresponds to creating the virtual machine images that will be used to host the application components in the cloud. For example, in the case of a typical Java EE application, a minimal configuration would require two images: one for the application server and one for the database.

The application server image would host the Java Container (e.g., Spring, JBoss, Tomcat, etc.) necessary for the software being delivered. When considering a specific IaaS cloud one would have to either build a new VM image from scratch (a cumbersome manual process), or reuse one of the several pre-built images usually offered by the IaaS provider. For example, Amazon offers many public AMIs for several known Java Containers (in several operating system distributions) that users can customize according to their technical needs and preferences (AMI, 2013). Even though reusing an existing image is easier than building one from scratch, in both cases the user still needs to follow a set of specific configuration steps, such as registering the image and configuring it with the provider's security certificates.

The database image in turn would host the specific database. In the database image, the configuration would be similar. The user would need to create a new database from scratch, or upload an existing one. Moreover, this is also the step where some tuning and configuration settings would be applied to the database.

Besides configuring each image, the user also has to configure the dependencies between each component. In the example above, a common step would be to configure the application server with the IP address of the database server virtual machine. It is worth mentioning that this step usually can only be performed after the database server has been deployed and initialized, as most IaaS virtual machines have dynamic IP addresses. Therefore, in every new deployment the database server would have a different IP address, which would have to be properly configured (either manually or automatically) as part of the application server configuration.

2.3 Motivation.

It is clear from the above description that the process of deploying applications in IaaS clouds includes a series of non-trivial steps that can be very time-consuming if done manually. Nevertheless, its importance cannot be ignored, as application deployment is one of the most basic tasks when it comes to using an IaaS cloud (Hajat et al., 2010) (Rodero-Merino et al., 2010) (Binz et al., 2012). For this reason, there are a number of tools currently available for deploying applications in IaaS clouds, which vary in functionality and deployment style. Some tools (e.g., Appzero (2013), Chef (2013), CloudFoundry (2013), Heroku (2013) and Rightscale (2013)) are not only deployment tools as they also combine other functionality such as being able to monitor and automatically scale the application, which is outside the scope of this work. Regarding deployment styles, while there are tools focused at using more powerful and complex mechanisms to be able to deploy several kinds of applications (e.g., scripting languages), there are other tools which have a narrower scope and use simpler mechanisms to deploy specific applications (e.g., Amazon Elastic Beanstalk (Beanstalk, 2013)).

To explain the reasons why we have decided to build another deployment tool (i.e., TREXCLOUD), we will first
provide an overview of the industrial setting where the tool was originally implemented and developed. E-NOVAR is a small software company located in Fortaleza, Brazil, whose main business focuses on building customized solutions for different industries. Each software product developed by E-NOVAR aims at offering a solution that is targeted for a specific problem that the customer faces. Since its inception, the company has implemented more than 50 products for different industries (e.g., financial, healthcare, media, gaming, retail etc.) and using several different technologies (e.g., Java, PHP, .NET, Objective C, Android, etc.).

Some of E-NOVAR customers are companies that operate with a very lean IT team and are not interested in investing on a dedicated datacenter infrastructure. For these companies, E-NOVAR was contracted to manage the computing infrastructure required for their applications. In order to host its customers' applications, E-NOVAR started using some local hosting companies in Fortaleza, which offer a reliable yet expensive hosting service. In an attempt to reduce its hosting costs, more recently the company has started to consider using cloud services.

Since E-NOVAR works with many different technologies and solutions, the best option was to use IaaS clouds due to their flexibility when compared to other cloud models. Initially, the company migrated a few applications to the Amazon EC2 cloud and experienced several benefits. Besides a significant reduction in operational costs, E-NOVAR also perceived other clear benefits, such as flexibility in pricing options, easiness to scale the resources up and down, as well as the good performance obtained from the cloud environment.

However, the migration process itself was cumbersome and mostly manual. As mentioned previously, many virtual machine images had to be created and configured with the required software components. Moreover, whole databases had to be moved from the previous hosting service to the Amazon cloud. This situation soon led to the conclusion that a customized software deployment tool was required to facilitate and expedite the cloud migration task. After experimenting with several automated and semi-automated deployment technologies, the company decided to build its own tool, as the technologies available were not able to fulfill all the company’s cloud deployment requirements, described as follows.

REQ01: User-friendliness. The tool shall provide a user-friendly way to perform the deployment. By this we mean that the tool shall not require any advanced infrastructure knowledge in order to be used (such as that required to write “low-level” configuration scripts), which could prevent less technical users from successfully performing the deployments.

REQ02: Fully automated support. The tool shall provide fully automated support for all steps involved in deploying the application. This means that all application components shall be deployed and completely configured by the tool. As explained previously, this requirement involves the execution of a series of steps such creating, deploying and configuring all the application virtual machines and their software components inter-dependencies (e.g., configuring the application server with the IP address of the database server virtual machine).

REQ03: Database support. As most industrial applications require a relational database, it is important that the tool provides the necessary support to install and configure whole databases as part of the cloud deployment process. The tool shall enable the user to reuse a database backup file from which the database will be created and populated by the tool. The tool also shall configure the database virtual machine and make it ready to be used by other components (e.g., application servers).

REQ04: Multicloud support. The tool shall provide deployment support for multiple IaaS providers. The users shall be required only to provide the tool with their IaaS cloud credentials and access information in order to be able to perform a deployment with that provider. The users shall be able to choose from different IaaS providers when performing a deployment. This is an important requirement since it avoids the users from being locked in to a specific provider.

The next section discusses whether (and how) several existing deployment approaches fulfill the set of requirements described above.

3. RELATED APPROACHES

This section provides a detailed analysis of several cloud deployment tools with respect to the four requirements described in the previous section. As there are many deployment tools currently available, we focus on five of those tools, namely Appzero (2013), Amazon Elastic Beanstalk (2013), Rightscale (2013), Heroku (2013), and Claudia (2013). These five tools were selected based on their popularity and similarity to our work. Other related approaches are also briefly discussed in the end of the section.

3.1 Appzero.

Appzero (2013) is a tool that enables the creation and management of Virtual Application Appliances (VAAs). Application components (e.g., application logic, application server, and database server) are packed inside a VVA, and can be moved to either a virtual or a physical machine. After building a VAA, deploying it in an IaaS cloud is a simple process that is managed by the tool. Another characteristic of Appzero is that all application components in a VVA are completely isolated from the operating system. The main point is to encapsulate all the changes of state of the application inside the VAA. This allows moving a VAA from one cloud provider to another without losing the application state.

REQ01: User-friendliness. Appzero offers a flexible and customizable deployment solution. It enables to deploy
any kind of application, but requires a certain level of expertise to use the tool because users need to understand about the configuration details of each of the application components contained inside a VAA.

REQ02: Fully automated support. Appzero automates the whole deployment and configuration process. After the required VAAs are created, the application components can be easily deployed and configured using the tool in a fully automated fashion.

REQ03: Database support. Appzero supports the deployment and configuration of databases while creating their VAAs. To this end, the tool requires from its users the necessary technical knowledge to install and configure a database system inside the VAA. This means that for users with less technical knowledge this step could be very complicated to achieve.

REQ04: Multicloud support. Appzero offers support for multiple cloud providers, enabling users to easily move VAAs to public cloud providers as well as to private clouds.

3.2 AMAZON ELASTIC BEANSTALK.

Amazon Elastic Beanstalk (EB) (Beanstalk, 2013) is an easy to use deployment tool that was built to facilitate the deployment of web applications in the Amazon EC2 cloud. The tool provides support for several popular web platforms (e.g., Java, .NET, PHP, etc.) and facilitates both the deployment and scaling of applications in the cloud. EB requires as input file the application logic code (e.g., .WAR files of a Java web application) that is uploaded to the cloud and configured within the context of an EC2 virtual machine instance. Regarding database support, EB is easily integrated with Amazon's own database solution, called Amazon Relational Database Service (RDS). Integration with a different database system is not as straightforward, as can be seen in the tool's online documentation.

REQ01: User-friendliness. EB offers a very easy deployment approach, in which users just need to follow a set of deployment Wizards and provide the tool with the application logic code.

REQ02: Fully automated support. The tool enables automatic deployment and configuration of the application logic but lacks in automated support for database services other than RDS (see below).

REQ03: Database support. The tool supports RDS as well as other relational databases, such as MySQL. However, in the latter case the tool requires a set of manual procedures in order to configure the database.

REQ04: Multicloud support. EB only supports application deployment in the Amazon EC2 cloud.

3.3 RIGHTSCALE.

Rightscale (2013) is an advanced tool that supports deployment and auto-scaling of applications in multiple cloud providers (e.g., Amazon, Rackspace, Google, etc.). The tool requires the users to produce a set of configuration scripts, called Rightscripts, which are executed during the deployment and startup of the application instances in the cloud. Typically, those scripts encapsulate a series of configuration steps that have to be executed in order to configure the whole application in the cloud. The tool offers several Rightscript templates (developed by the Rightscale users community) that can be used to configure popular application development and execution environments (e.g., a LAMP stack, and .NET or Java EE applications). Even though those templates can be easily reused and modified by the users, they still require a reasonable level of expertise in order to be properly understood and customized with an appropriate set of configuration parameters (e.g., location of the application code repository, database configuration parameters, etc.).

REQ01: User-friendliness. The power and flexibility gained by writing and reusing configuration scripts is a clear advantage of Rightscale, allowing users to have a tight control over the whole deployment process. However, in order to build and reuse those scripts users need to have a good level of technical knowledge.

REQ02: Fully automated support. Once the necessary scripts are built, Rightscale is able to automatically deploy and configure the whole application stack.

REQ03: Database support. Users are able to deploy any type of database with Rightscale by building the required configuration scripts. Therefore, database support also depends on advanced technical knowledge from users.

REQ04: Multicloud support. Rightscale offers support for several public cloud providers (e.g., Amazon, Rackspace, etc.) as well as open source private cloud solutions (e.g., OpenStack).

3.4 HEROKU.

Heroku (2013) is a PaaS provider that offers support for applications developed in several programming languages (e.g., Java, Scala, Node.js, Python, etc.). Heroku targets developers that want to run and scale their applications in its own cloud infrastructure, which in turn is build on top of Amazon EC2. For experienced developers this can be an interesting solution as it supports several popular development tools, such as application frameworks (e.g., Spring MVC and Hibernate) and source code versioning tools (e.g., Git). Using Heroku to deploy and manage applications requires advanced technical skills from its users, who must be able to write and understand shell scripts to deploy and configure their applications.

REQ01: User-friendliness. Heroku focuses on providing support for software developers interested in deploying and managing their applications in the Heroku cloud. Users with less expertise in source code versioning tools and scripting languages might have difficulty in using Heroku.

REQ02: Fully automated support. Heroku fully automates the deployment process for several kinds of applications and development environments.
REQ03: Database support. Heroku supports the Postgres and MySQL relational databases. The tool requires the use of a series of scripts in order to migrate an existing database to the Heroku cloud.

REQ04: Multicloud support. Heroku operates on its own cloud, which is build on top of Amazon EC2. Users are not able to deploy applications in other cloud providers with Heroku.

3.5 CLAUDIA.

Claudia (2013) (Rodero-Merino et al., 2010) is a platform that supports application deployment and auto-scaling in IaaS clouds. Claudia users can specify a set of elasticity rules with triggers and actions that enable the platform to manage different types of applications. Claudia works with different IaaS providers and uses the Open Virtual Format (OVF) standard (OVF, 2013) to package the application components, configuration parameters and scalability rules.

REQ01: User-friendliness. Claudia users need to have a deep knowledge of OVF in order to provide the virtual application specification and the scalability rules.

REQ02: Fully automated support. Claudia fully automates the deployment process given an OVF specification provided as input.

REQ03: Database support. Users can deploy any type of application with Claudia as long as they provide the proper OVF specification and database images, which requires a good technical knowledge.

REQ04: Multicloud support. Claudia supports multiple cloud providers.

3.6 SUMMARY AND OTHER APPROACHES.

There are a number of other approaches that, like most of the tools investigated in the previous section, also support the deployment of any kind of application by resorting to more general deployment mechanisms. TOSCA (Binz, 2012), for example, works in a similar fashion as Claudia. The applications that will be deployed in the cloud are described in an XML-like template where the user describes the software components to be used, their relationships and configuration properties. Moreover, the user also specifies workflows (called Pans) that describe the management flow of the application. A similar scripting-based approach is offered by CloudFoundry (2013) and Chef (2013). The main shortcoming of these approaches is that they require a deeper knowledge of system administration and scripting languages, thus restricting their use to more advanced technical users.

The approach described in (Wettinger, 2013) defines a methodology that focuses on deploying the application and the middleware (e.g., application servers, database, Enterprise Service Bus) based on a set of script-based deployment plans. There is no tool support yet for the approach and its applicability in practice was demonstrated using Chef. Besides, the approach currently does not support dynamic wiring between components which is crucial for automating cloud applications as some properties are only known during deployment (e.g., IP address).

<table>
<thead>
<tr>
<th>Approach</th>
<th>REQ01</th>
<th>REQ02</th>
<th>REQ03</th>
<th>REQ04</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appzero</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>EB</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>Rightscale</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td></td>
</tr>
<tr>
<td>Heroku</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
<tr>
<td>Claudia</td>
<td>★</td>
<td>★</td>
<td>★</td>
<td>★</td>
</tr>
</tbody>
</table>

Support: Full ★ Partial ★ Weak ○

Table 1. Analysis of five deployment approaches

The work by (Kang, 2011) focuses on redeploying cloud applications based on the user experienced performance (measured by latency). The idea is to dynamically redeploy the application in machines that provide optimal performance to the users. This work was further improved in (Kang, 2012) to consider not only distribution of users but also how services affect the performance of others. To attack this challenge, they employ cross service information as well as user locations to build a new model based on integer programming. The works (Fan, 2011) (Fan, 2012) also focus on selecting the best deployment strategy from the performance perspective. They propose a clustering based method for deploying communication intensive applications to the cloud environment. All these works focus on a different aspect (performance optimization) of deployment and are complementary to ours.

There are also some other interesting works that focus on different problems related to deployment. (Ruehl, 2013) and (Ruehl, 2012) describe an approach for deploying application components (services) in SaaS that allows the customer to select (based on a set of requirements) with whom they want to share infrastructure. The idea is to avoid sharing a multitenant infrastructure with possible competitors or untrusted partners. Another work (Chen, 2012) offers a framework for deploying virtual machines in a way that maximizes resource utilization and minimizes the costs for provisioning the VMs in a services provider infrastructure.

Neither the approaches previously described nor the five deployment tools shown in Table 1 fully satisfy all our requirements.

4. TREX CLOUD

TREX CLOUD aims at providing easy to use support for deploying web applications in IaaS clouds. The tool currently supports the deployment of Java EE applications in two well-known commercial cloud providers: Amazon and Rackspace. The goal of the tool is to free the user from having any specific knowledge about proprietary
technologies of IaaS clouds such as those required for creating and configuring virtual machine images. Moreover, the user does not need to have any knowledge about “low-level” scripting languages, so that s/he can deploy a complete application in the cloud with a few mouse clicks. Figure 1 shows the tool’s architecture, which contains three main components, briefly described below.

Figure 1. TREXCLOUD architecture

User Interface (UI). This component implements the tool’s web-based graphical user interface. It is responsible for reading in the user-provided input files and configuration parameters necessary for setting up and deploying the application in the cloud.

Communication. This component is responsible for invoking the specific deployment operations provided by each IaaS cloud provider supported by the tool. It currently offers support for the Amazon and Rackspace clouds, and can be easily extended to support other providers, as we will explain later on in the paper.

Pre-built Images. This component provides a set of pre-configured images (e.g., application server, database server, etc.) for each IaaS cloud supported. The tool is able to automatically launch and configure those images with the appropriate set of configuration parameters during the deployment process.

The following subsections provide more details on the design and implementation of each of these three components.

4.1 USER INTERFACE.

One pre-requisite for using the TREXCLOUD is that before performing any new deployment the user needs to have created an account with one of the supported IaaS providers. Once the user has created such an account, s/he has to register her/his cloud credentials with the tool, as shown in Figure 2.

After this step, the user is apt to perform a deployment with the registered clouds. To do this, the user must provide the tool with following information (see Figure 3):

- **Cloud Provider**: the name of one of the supported IaaS cloud providers.
- **Name**: the name given for the current deployment.
- **File**: the location of the .WAR file that contains the packed application logic that will be uploaded to the cloud.
- **Context**: the context file of the Java EE application.
- **Database**: the name of one of the supported database servers, along with the username, password, name and backup file created for the application database.

Figure 2. Registering cloud credentials

Figure 3. Deploying a web application

4.2 COMMUNICATION.

This is the core component of the tool. It offers a generic interface, named TRexCloudService (see Figure 4), through which the UI component invokes the services of different cloud providers. In this way, the UI component is completely decoupled from any particular IaaS provider API. The TRexCloudService interface defines a set of deployment operations that need to be implemented for each specific provider. In the case of the Amazon EC2 cloud, for instance, the implementation for that interface (AWSServiceImpl) was based on the EC2 Java development API. Similarly, in the case of the Rackspace cloud, the implementation (RakspaceServiceImpl) was based on that provider’s RESTful API. In order to extend the TREXCLOUD for another cloud provider, one has to
implement the operations defined by the TRexCloudService interface, along other requirements, as we explain in Section 4.4.

Figure 4. Communication component class diagram

4.3 PRE-BUILT IMAGES.

In TREXCLOUD, it is up to the tool developer (instead of the tool user, as with other deployment approaches) to create all the images and configuration scripts required to automate the deployment and configuration process. The way this is supported in the current version of the tool is as follows. The TREXCLOUD developer creates virtual machine images containing the required components for each IaaS deployment, as described in Section 2.1. For example, the tool currently has pre-built images for the application server (Tomcat) and the database server (Postgres) supported by both Amazon and Rackspace. Each component image contains an embedded configuration script that runs automatically once the virtual machine is booted and is responsible for setting up the configuration properties passed by the user in the UI component.

4.4 EXTENDING TREXCLOUD.

In order to provide support for a specific cloud provider, the developer of TREXCLOUD has to extend the tool in basically two steps. The first step would be to provide a concrete implementation for the TRexCloudService interface implementing all its methods based on the APIs of the target cloud provider. The second step would be to create the specific images that will contain the required components for each IaaS deployment, as described in Section 2.1. For example, the tool currently has pre-built images for the application server (Tomcat) and the database server (Postgres) supported by both Amazon and Rackspace. Each component image contains an embedded configuration script that runs automatically once the virtual machine is booted and is responsible for setting up the configuration properties passed by the user in the UI component.

4.5 AN ILLUSTRATIVE EXAMPLE.

To illustrate how the whole deployment process works, we will use a simple application as an example. Suppose the developer of a Business Intelligence (BI) application, called MyDashBoard, wants to deploy it in the Amazon cloud. If s/he decides to use the TREXCLOUD tool (from now on we call this person the user) the user needs first to create an account in Amazon (is s/he does not have one already) and then to register her/his Amazon credentials with the tool (Figure 2).

The user then needs to provide a set of configuration properties and some input files, as shown in Figure 3. At this point, when the user hits the “send” button what happens “behind the scenes” is that all those properties are saved into a file that will be uploaded and stored in the Amazon simple storage service (S3). Moreover, the files containing the application logic (.WAR) and the database backup (.TAR) are also uploaded from the user’s computer into S3.

Once the user hits the “send” button the whole application is configured and started up in the cloud. The Amazon-specific implementation for the tool’s communication interface (Figure 5) will call the required APIs for starting up the VM instances of all application components in the correct order.
For example, it will start the database instance first and then, when that instance is booting, the script embedded in it will download the properties file (containing the database username, password and name) and database backup (.TAR) file and perform the complete configuration of the database.

After the database is configured, the tool starts up the VM instance (based on the Tomcat pre-built image) of the application server and its embedded script downloads the application logic files (.WAR) and the correct configuration of the database IP address. What is important to highlight is that the user of the tool is completely unaware of all these configuration steps and only has to provide a bunch of configuration properties and files. This is one of the main contributions of our approach since it makes deployment of a complete application a matter of a few mouse clicks.

5. EMPIRICAL ASSESSMENT

To evaluate the effectiveness of TREXCLOUD, we conducted an empirical study where we asked nine different users to use and assess three different approaches to deploy two different web applications in the Amazon cloud. Our goal was to investigate how TREXCLOUD compares, in terms of user effort, with other approaches commonly used for deploying web applications in the Amazon cloud. More specifically, we wanted to investigate the following research questions:

(RQ1) How does our approach compares, in terms of effort (measured as the time spent to complete a deployment), with other approaches commonly used by Amazon customers?

(RQ2) Which approach is more suitable for which category of user (considering users with varying levels of expertise in Java and cloud related technologies)?

5.1 SETUP AND PLANNING.

To carry out the evaluation, we selected nine users with the following profiles (three users from each profile):

Naive. Users that have never performed a deployment of a Java EE application before and that have never used an IaaS cloud.

http://hipore.com/ijcc
Intermediate. Users that have some previous experience with the deployment of a complete Java EE application (including application server and database), but no experience with IaaS clouds.

Advanced. Users that have the same skills as the users of the Intermediate profile plus some previous experience with IaaS cloud deployment.

Each user was asked to perform six deployments in the Amazon cloud, three deployments per application. In each deployment, the user selected one out of the three following approaches to deploy the application:

Amazon Elastic Compute Cloud (EC2): In this approach, the user performs the deployment using only the Amazon APIs and Dashboard tool, which provides a set of basic operations for deploying and managing VM instances in the Amazon EC2 cloud. Therefore, with this approach most of the deployment steps described in Section 2 are performed manually by the cloud user. This is the approach most commonly used by current IaaS cloud users.

Amazon Elastic Beanstalk (EB): In this approach, the user performs the deployment using Amazon Elastic Beanstalk, a web-based tool offered by Amazon to deploy web applications in its EC2 cloud (Beanstalk, 2013). One important limitation of this tool is that it does not support deploying the database only the application logic.

TREX CLOUD (TREX): In this approach, the user performs the deployment using TREX CLOUD, the tool described in this paper.

The two applications selected for the study were Calendar and MyDashBoard. Calendar is a simple web-based calendar application written in Java and obtained from Google. One interesting characteristic of this application is that it does not have a database, only application logic deployed as a .WAR file. MyDashBoard is a web-based Java Business Intelligence tool developed by E-NOVAR, the company behind TREX CLOUD. MyDashBoard contains both application logic and a database. The application logic is packed in a .WAR and context file while the database is packed in a backup file (.TAR) for the Postgres database.

During the study, the nine users were instructed to perform the deployments in random order, to avoid biasing one approach in favor of others. Moreover, they were required to complete all six deployments. A deployment was considered complete when the user: (i) had the application up and running in the Amazon cloud, which was verified by successfully accessing its URL from a web browser; or (ii) explicitly assumed that s/he had failed to perform the deployment.

Before conducting the experiments, the users were given an overview of the three deployment approaches investigated. They also received a package containing all files required for the deployment of the two applications (e.g., .WAR, .TAR, context file) as well as the necessary credentials to access the Amazon cloud.

To provide a familiar computing environment for the experiments, the users were allowed to perform the deployments using their own computers. However, to make it easier to compare the users’ deployment efforts, we configured all computers used in the study to record a video of their screen during each deployment session. These videos were then subsequently analyzed to calculate precisely the time taken by each user to complete each deployment.

In addition to the quantitative analysis of the users’ deployment effort, we also conducted a qualitative analysis to investigate the users’ impression with each of the three approaches. For the qualitative analysis, we asked the users to fill in a simple questionnaire to provide us with some qualitative data regarding each deployment. The questionnaire contained the following items:

- Assign a grade to each deployment in a scale from 0 to 5, where 0 represents that you failed to perform the deployment and \{1, 2, 3, 4, 5\} represent, respectively, that the deployment was \{very difficult, difficult, average, easy, very easy\} to perform.
- Suggest some improvements on the functionality of each of the tools used.
- Comment about the advantages of using each tool in the context of each deployment.

5.2 Quantitative Results

Table 2 shows the quantitative results obtained representing the time taken by each user to deploy the Calendar application. As we can see, for all users (except for User 2 by a few seconds) TREX outperformed the other two approaches by a large margin. We can also see that none of the naïve users were able to deploy the application with EC2. In contrast, all intermediate and advanced users were able to successfully complete the deployment with that approach; however, those deployments all took a considerable amount of time – approximately 28 minutes in the fastest case (for User 8). The greater effort required by EC2 can be explained by the fact that this is a mostly manual approach.

http://hipore.com/ijcc 48
Another interesting result is the deployment times obtained with EB. Since the Calendar application does not include a database, technically deploying it with this approach should be of a similar level of difficulty of doing it with TREX. This seemed to be true for Users 2, 5 and 9, who were able to complete the task with EB in less than 6 minutes. However, we can see that most users took much longer to deploy the application with EB than with TREX.

In fact, deploying the application with TREX was always very fast, taking from less than two minutes (for Users 4, 6 and 8) up to a maximum of four minutes and twenty-nine seconds (for User 3). Most of this time is spent in the upload of the configuration files (mentioned in Section 4) and virtual machine start up at Amazon rather than with user interaction with TREX.

Table 3 shows the results representing the time taken by each user to deploy the MyDashboard application. Again, only the intermediate and advanced users were able to complete the task with EC2, although it took them a very long time to do it (approximately two hours on average). In the case of EB, this time only one user (User 5) was able to successfully perform the deployment, taking her/him approximately one hour to complete the task. The reason for the high failure rate observed with this approach was the difficulty to deploy the application database. When analyzing the videos recorded during the deployment sessions performed with EB, we noticed that the users somehow got confused about the proper way to configure the application since that tool does not explicitly support database deployment.

Table 4 shows a consolidate view of the effort reduction achieved by TREX when compared to EC2 and EB, with respect to the three user profiles and the two web applications. For Calendar, considering only successful deployments, TREX achieved a reduction of 94% and 83%, on average, in the application deployment time when compared with EC2 and EB, respectively. For MyDashboard, in turn, the results show an average reduction of 94% and 91% in the deployment time when compared to EC2 and EB, respectively (again, considering only successful deployments).

### 5.3 Qualitative Results

Figure 6 shows the results gathered in the questionnaires with the grades assigned by all users for the Calendar application. Users 1, 2 and 3 (naïve users) could not complete the task with EC2 (grade = 0) and gave either grade average (Users 1 and 3) or very easy (User 2) to EB. All naïve users found TREX very easy (grade = 5) to use. For intermediate and advanced users (except for User 6) they found that EC2 is the most difficult to use followed by EB and then TREX, which almost all of them considered very easy. This feedback helps to back up our claim that independently of user profile TREX provides an easy solution for Java Enterprise deployment.

Table 6 shows the grades for MyDashboard. Again, we can see again that all naïve users were unable to complete the task with EC2, which was best evaluated as being of average difficulty (Users 6 and 9). The interesting results here are the grades assigned to EB. Only User 5 was able to complete the deployment with EB and recognized it as
being very difficult (grade = 1). As explained before, this was due to the inclusion of the database as part of the deployment, with most users being unable to complete the process with the EB tool. Regarding TREX, all users found it either easy or very easy to deploy the MyDashBoard application, and had no problems in configuring the database.

Some of the answers given by the users to the open questions included in the questionnaire provide further evidence of the benefits of the TREX CLOUD tool. For example, the following answer was given by one of the naïve users:

“TREX CLOUD is much faster and practical. I just need to provide the input files and hit a few clicks to get everything done. The tool does the entire job for me.”

Another user complained about one particular deployment with Amazon Elastic Beanstalk:

“After I deployed the application logic the web server was not running. The tool only informed that there was a problem with the service but I could not find what exactly the problem was. I tried to restart Tomcat and even restart the instance but could not fix it and complete the task.”

Another common complaint among several EC2 and Elastic Beanstalk users was with respect to the lack of adequate support (both in terms of functionality and documentation) for deploying and configuring a database.

5.4 Revisiting the Research Questions

Let us now revisit our research questions in order to summarize our findings from this study.

(RQ1) How does our approach compares, in terms of effort (measured as the time spent to complete a deployment), with other approaches commonly used by Amazon customers?

As shown from the quantitative results presented in Tables 2 and 3, deploying a Java application in the Amazon cloud with our approach can be much faster (sometimes up to ten times faster) than with other common approaches, such as Elastic Beanstalk and EC2.

(RQ2) Which approach is more suitable for which category of user (considering users with varying levels of expertise in Java and cloud related technologies)?

For all three user profiles investigated, deploying the two applications with TREX CLOUD was much easier and faster than with other common approaches. In the case of naïve users, the tool proved essential as they failed to perform the deployments using the other approaches. This was especially true when a database deployment was required.

5.5 Limitations

The study reported in this paper shares some of the limitations commonly found in any experimental studies involving software (Easterbrook, 2007), such as to depend on people with varied level of skills to perform the required tasks (i.e., cloud deployments). For our study purposes, it was better to have users with a more diverse background since we wanted to investigate how the different categories of users influenced the results.

As mentioned before, we tried to make sure users performed deployments in different orders (varying the tool and application) so that this could avoid putting one tool in favor of another as the same user normally learns between deployments. Moreover, during the studies each user performed the tasks on their own and did not communicate among them.

The first author of this paper was also available during the experiments to respond to any doubts about the task to be done but not about how to do it or how to use a specific tool. In addition, to avoid any bias in the results, all the videos for each deployment session were analyzed to make sure that only the time devoted to the deployment being performed (e.g., the user could potentially start browsing the web for another purpose) was actually computed as part of the overall deployment time associated with a given deployment task.

6. Discussion

After presenting the TREX CLOUD tool implementation and its evaluation, we now revisit our requirements defined in Section 2 and discuss if (and how) the tool satisfies then.

REQ01: User-friendliness. The users of TREX CLOUD have a very easy to use way of deploying their applications, as shown in Section 5. All they are required is to provide a set of input files in a web-based user interface without any need to write any scripts or XML specifications.

REQ02: Fully automated support. TREX CLOUD fully automates the deployment and configuration of the application in the cloud once the required input files are provided. The user does not need to perform any manual configuration step.

REQ03: Database support. TREX CLOUD supports automated creation of relational databases in the cloud, enabling users also to move an existing database to the
cloud by providing a backup file and some configuration parameters (e.g., database name, password, etc.). The application is automatically configured to connect to the provided database without any action required by the user. This was a major facilitator for users as pointed out by the study presented in Section 5.

REQ04: Multicloud support. TREXCLOUD currently supports deployments on Amazon and Rackspace and can be extended for other providers, as explained in Section 4.4.

To sum up, we can conclude that TREXCLOUD is the only approach that completely fulfills our requirements when compared to other existing deployment approaches (see Table 1).

7. CONCLUSIONS AND FUTURE WORK

One of the fundamental aspects of deploying or moving applications to the cloud is to be able accomplish this task in the easiest and effortless manner. In this work, we presented TREXCLOUD, a tool created for easy deployment of web applications in IaaS clouds. TREXCLOUD enables users to quickly deploy a web application requiring, as input, a small set of files (e.g., WAR, context and database backup files) that are automatically uploaded to the cloud and used by TREXCLOUD to set up and configure all the virtual machines that represent the complete application stack.

Our empirical results show that users of varying levels of expertise in cloud technologies (from naïve to more experienced users) were able to deploy a Java web application in the Amazon cloud much faster with TREXCLOUD than with the two most common approaches used for this same purpose. TREXCLOUD outperformed the other deployment approaches with a reduction in deployment effort of up to 94%.

Our future work will focus on supporting other development platforms (e.g., .NET, PHP, Ruby on Rails) as well other database systems (e.g., MySQL, NOSQL). We also plan to incorporate self-scaling mechanisms and provide support for other IaaS clouds.

8. ACKNOWLEDGMENT

The Authors wish to thank Financiadora de Estudos e Projetos (FINEP) for supporting this research under the umbrella of the Subvenção Econômica 2009 Program. Nabor C. Mendonça is partially supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), under grants nos. 311617/2011-5 and 487174/2012-7.

9. REFERENCES


Authors

Américo Sampaio holds a B.Sc. in Computer Science from Federal University of Ceará, Brazil; an M.Sc. in Computer Science from Federal University of Pernambuco, Brazil; a Ph.D. in Computing from Lancaster University, U.K and a Post-Doc from SAP Research United Kingdom. Currently he works as Assistant Professor at University of Fortaleza and collaborates in several cloud computing projects with companies and other universities.

Roberto Costa Jr holds a B.Sc. in Computer Science from Federal University of Ceará, Brazil; and an M.Sc. in Computer Science from University of Fortaleza, Brazil. Roberto has over 10 years of experience as a software engineer and is the director of E-Novar.

Nabor Mendonça holds a B.Sc. in Data Processing from Federal University of Amazonas, Brazil; an M.Sc. in Computer Science from State University of Campinas, Brazil; and a Ph.D. in Computing from Imperial College London, U.K. Currently, he's a Titular Professor at University of Fortaleza, Brazil, where he conducts several research projects on the use and evaluation of cloud computing technologies. Prof. Mendonça is a recipient of a Microsoft Research 2013 Software Engineering Innovation Foundation (SEIF) Award.

Ricardo Hollanda Filho holds a B.Sc. in Computer Science from University of Fortaleza and has more than 5 years of experience as a software engineer at E-Novar.
A QUEUING MODEL TO ACHIEVE PROPER ELASTICITY FOR CLOUD CLUSTER JOBS

Khaled Salah
Electrical and Computer Engineering Department
Khalifa University of Science, Technology and Research (KUSTAR), UAE
khaled.salah@kustar.ac.ae

Abstract
Achieving proper elasticity for cloud jobs is a challenging research problem and far from being solved. In this extended and revised paper, we investigate how to achieve proper elasticity for highly parallelized jobs which run on cloud clusters. In particular, we present an analytical model based on finite queuing systems that can be used to determine at any given instance of time and under current workload conditions the minimal number of cloud resources needed to satisfy the SLO requirements such as response time and request loss probability. We give numerical examples to demonstrate the applicability and usefulness of our model. In particular, we study the performance of a cloud-based clustering system in terms of queue utilization, request loss, and response service time. We also give an example of fluctuating workload to show how our model can be used to achieve proper elasticity. Discrete Event Simulation has been used to verify the correctness of our proposed model.

Keywords: Cloud clusters; HPC clouds; Cluster Jobs; Distributed Systems; Modeling and Analysis; Queuing Theory; Performance Evaluation.

1. INTRODUCTION
Because of the lower cost, reduced investment in IT infrastructure, and increased resource availability, cloud computing is becoming an increasingly attractive option to run cluster jobs and applications (He, Zhou, Kobler, Duff, & McGlynn, 2010; Carlyle, Harell, & Smith, 2010; Napper & Bientinesi, 2009; Evangelinos, 2008; Juve et al., 2009; Vecchiola, Pandey, & Buyya, 2009). Many popular cloud providers have started to offer on-demand HPC (High Performance Computing) services for running cluster jobs and HPC applications (He et al., 2010; Carlyle et al., 2010; Napper & Bientinesi, 2009). These cluster and HPC jobs are characterized to be highly parallelized. Examples of these jobs include industry and business applications such as pharmaceuticals, oil and gas, healthcare, financial services and manufacturing. In addition, academic researchers are leveraging cloud clusters to perform research in physics, chemistry, biology, computer science, and materials science.

Figure 1 shows a typical cloud-based distributed infrastructure for executing cluster jobs. The infrastructure is composed of a number of key elements and nodes. The Job Scheduler (JS) is the key coordinator and scheduler of cloud cluster jobs. The JS is aware of the IaaS cloud resources. The JS services the jobs from the Job Queue one at a time in a FCFS (First Come First Serve) scheduling discipline. The JS examines the type, size, and SLO (Service Level Objective) performance measures of the job. All of this information is commonly included in the SLA (Service Level Agreement). The JS splits the job into multiple equal compute units to be run on worker VMs. For faster execution, the VM must be already provisioned and ready to accept compute workload from the JS. Otherwise, the JS has to provision the worker VMs. The JS also coordinates and schedules the Result Aggregator (RA) which is a node configured to aggregate and collect the computed results from the worker VMs. In MapReduce terminology, the JS is known as the Master or Job Tracker, and the RA is known as the Reducer (Dean & Ghemawat, 2008). Typically, the JS, worker VMs, and RA are interconnected with a high-speed 1 or 10 Gbps Ethernet LAN, with a high speed access to a distributed database or file system such as Hadoop or Lustre (Morengovozmediano, Monero, & Llorente, 2009; Fransham et al., 2010).

The JS acts as a load balancer as well as an elastic controller. One of the primary challenges of the JS is to achieve adequate elasticity which is a key feature of cloud computing paradigm. Elasticity is defined as the ability to use minimal cloud resources at any given time in order to satisfy the SLO performance requirements. As the workload changes over time, the JS has to dynamically provision the minimal number of worker VMs to satisfy the given SLO requirements. Allocating more VM nodes than required to satisfy the SLO will result in over-provisioning and higher cost to the user. On the other hand, allocating fewer resources than required will lead to under-provisioning whereby the SLO performance requirements are violated. Therefore, it is critical for the JS to determine the correct number of worker VMs needed to execute a cluster job taken into account the current mean workload and the SLO requirement of the job.
related work found in the literature on elasticity for cloud-applicability and usefulness of our proposed model, (2) we have included additional materials related to: (1) the published work in (Salah, 2013). In this extended version, throughputs, blocking probability, etc. utilization, network bandwidth, overall service time, request. The metrics may include cloud resource can be deciding factors for the CCA to grant or deny a instance management, migration, etc. Second, for multi-tier applications, our model can be used to compute the expected delay at each tier in which a LB and multiple compute instances are involved. The aggregated delay at each tier can be used to compute the end-to-end response time. Third, the model can compute the required cloud instances and estimate the service response time and the required network bandwidth when deploying VDC (Virtual Data Center) or Amazon EC2 Fleet in which multiple VM instances (or EC2) are provisioned based on the given SLA requirements. Fourth, the model can be used in Cloud Call Admission (CCA) to accept or deny user requests to provision a single or multiple VM instances. As VM instances continuously get provisioned and de-provisioned (due to auto-scaling, migration and termination), the mean resident time for an instance can be estimated. Our model can then be used to determine key performance metrics that can be deciding factors for the CCA to grant or deny a request. The metrics may include cloud resource utilization, network bandwidth, overall service time, throughput, blocking probability, etc.

This paper is an extended and revised version of the published work in (Salah, 2013). In this extended version, we have included additional materials related to: (1) the applicability and usefulness of our proposed model, (2) related work found in the literature on elasticity for cloud-based cluster jobs, (3) more details and expansion on the mathematical derivations and consideration of special cases, and (4) derivation of new set of equations and formulas for key performance measures. More importantly, in this paper, we extended the section on numerical results to include new figures on queue utilization, request loss probability, and response time curves. Detailed interpretations and discussions were given for these new figures.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 presents our analytical model to capture the dynamism of executing cluster jobs on the cloud infrastructure. Section 4 presents numerical examples to show how the analytical model can be used in determining the response time given the current workload conditions. The section also gives numerical examples to illustrate how our analytical model can be used to achieve proper elasticity for cloud cluster jobs. Finally, Section 5 concludes the study and identifies future work.

2. 3. RELATED WORK
In the literature, a framework for elastic execution of real-time cluster jobs on cloud platforms had been proposed and studied in (Zhang, Shu, Chong, Lu, & Yang, 2013). The authors proposed an elastic approach for elastic execution and processing of streaming data on a cloud-based cluster. To achieve elasticity, the authors used a heuristic approach based on a temporal distance cost among data and cluster nodes to allocate compute resources given a fluctuating workload. In (Morengo-Vozmediano et al., 2009), an elastic management approach of cluster-based services hosted in the cloud has been proposed. The authors used a VM manager based on the OpenNebula engine to allocate EC2 compute node resources from Amazon AWS. The authors studied experimentally the computing cluster and a web server cluster. Communication overhead has been identified as a major concern in the deployment of such an approach. Commercially, Amazon Elastic MapReduce (Amazon EMR) is a web service being offered by AWS Cloud to run MapReduce jobs (“Amazon Elastic MapReduce (Amazon EMR),” n.d.). However, the number of slave nodes is specified manually by the user and remains constant during the execution of the job, with no guarantees of satisfying SLO requirements such as response time or request loss. For adequate elasticity and autoscaling, other cloud monitoring and management solutions should be integrated with AWS EMR and Simple Queueing Service (SQS). These solutions are publically and commercially available such as Scalr, CHEFF, PUPPET, NetScler, and RightScale. Even with these integrated solutions, determining the number of slave nodes for mappers and reducers is almost done arbitrarily and is in fact a trial and error exercise that will yield to multiple SLO violations in the course of adjusting the number of mappers and reducers.

http://hipore.com/ijcc
Fadika & Govindaraju (2011) proposed and implemented a dynamically elastic MapReduce framework in which the number of slave nodes is dynamically adjusted during the job execution. Authors had shown such a scheme can boost the performance of job execution by 50%. In (Fadika & Govindaraju, 2011), the performance was characterized for one single job with no consideration given to the impact of workload conditions of arriving jobs on the overall job execution. Other work in the literature has focused on boosting the performance of MapReduce jobs by improving the job scheduling as in (Cheng, Zhang, & Boutaba, 2011; Tan, Meng, & Zhang, 2012) and by tuning various system parameters as in (Wang, Butt, Pandey, & Gupta, 2009; Babu, 2010).

In the literature, queuing models have been used to study the performance of cloud computing systems (Ali-Eldin, Tordsson, & Elmroth, 2012; Wang et al., 2008; Urgaonkar Shenoy, Chandra, Goyal, & Wood, 2008; Khazaee, J. Misic, & V. Misic, 2012; Kikuchi & Matsumoto, 2011; Firdhous, Ghazali, & Hassan, 2011; Salah & Boutaba, 2012), but no research work has been reported on achieving elasticity for cloud cluster jobs using queuing modeling. In addition, and in general, achieving proper elasticity is still a challenging problem and far from being solved (Islam, Lee, Fekete, & Liu, 2012; Ali-Eldin, Kihl, Tordsson, & Elmroth, 2012a; Ali-Eldin, Tordsson, & Elmroth, 2012b). To address this, this paper presents an analytical model with closed-form formulas that can be extremely useful in achieving proper elasticity for cloud cluster jobs, or jobs that require parallel execution on a distributed infrastructure. In particular, our model computes the minimal number of VM workers needed to satisfy the SLO response time of a cloud cluster job given the current mean workload conditions. We give numerical examples and show how this model can be used to achieve proper elasticity.

4. Analytical Model

In this section, we present a finite queuing model for executing cluster jobs on the cloud infrastructure with the aim of deriving closed-form solution for the mean response time (or the mean service time). Figure 2 shows the queuing model which is a representative of the cloud cluster shown in Figure 1. As shown in Figure 2, an arriving job gets first queued in a buffer of size K-1 and then gets serviced sequentially in three stages by: (1) the JS node, (2) the worker VMs which work in parallel, and (3) the RA node. We assume independent exponentially distributed service times for all nodes and VMs. These service times include computation as well as access latencies associated with database access and network communications. Such assumptions are commonly used in analyzing cloud computing systems (Ali-Eldin et al., 2012a; Wang et al., 2008; Urgaonkar et al., 2008; Urgaonkar, Pacifici, Shenoy, Spreitzer, & Tantawi, 2005; Khazaee et al., 2012; Kikuchi & Matsumoto, 2011; Firdhous, Ghazali, & Hassan, 2011). We also assume a Poisson arrival for incoming jobs. This assumption is also in agreement with analysis and measurements taken from Google cluster traces. The Google traces were for general-purpose compute cluster recorded over 29 days (Reiss, Tumanov, Ganger, Katz, & Kozuch, 2012; Sharma, Chudnovsky, Hellerstein, Rifaat, & Das, 2011; Zhang, Hellerstein, & Boutaba, 2011). In addition, we assume the JS handles homogenous jobs (i.e., of the same type and the same SLO performance requirements). This is a realistic assumption in which an HPC customer executes similar jobs but with varying workload over time.

Let \( \lambda \) denote the workload which is the mean arrival rate of the cluster jobs, \( \mu \) denote the mean service rate of the JS node, \( \beta \) denote the mean service rate of a worker VM, \( r \) denote the mean service rate for the RA node, \( N \) denote the number of worker VMs, \( K \) denote the capacity of the queuing system inclusive of the one in service, and \( \gamma \) denote the throughput or the job departure rate.

The queuing system shown in Figure 2 is basically an M/G/1/K queuing system, with a Poisson arrival \( \lambda \), generally distributed service times, and a system capacity \( K \). The challenge in analyzing such a queuing system is to compute \( b(x) \) which is the PDF of the generally distributed random variable \( X \) representing the service times, and subsequently deriving a closed-form solution to \( \alpha_k \) which is the probability of having \( k \) job arrivals during this service time. \( \alpha_k \) is expressed in terms of \( b(x) \) as

\[
\alpha_k = \sum_{x=0}^{\infty} \frac{(\lambda x)^k}{k!} e^{-\lambda x} b(x)dx
\] (1)

Figure 2. Finite queuing model for servicing highly parallelized jobs on a cloud cluster

http://hipore.com/ijcc
Effectively, \( b(x) \) is the PDF of a random variable \( X \) which is the sum of three independent random variables representing the service times at the three service stages; namely, the JS node, the worker VMs, and the RA node. The first stage random service time has an exponential distribution with a mean \( 1/\mu \) and a PDF \( f(t) = \mu e^{-\mu t} \). The third stage random service time has also exponential distribution with a mean \( 1/r \) and a PDF \( g(t) = re^{-rt} \).

For the second stage, we assume the service times for each worker VM are independent and exponentially distributed with a mean \( 1/\beta \). Then, the second stage random service time \( B \) for these \( N \) parallel workers can be expressed as \( B = \max\{B_1, B_2, ..., B_N\} \). In a way, the second stage service time depends on the slowest of these \( N \) parallel workers, whereby the RA will start executing after the slowest worker VM has finished. Therefore, the CDF of the random variable \( B \) can be expressed as

\[
F_B(t) := P(B \leq t) = \prod_{i=1}^{N} P(B_i \leq t) = (1 - e^{-\beta t})^N
\]

(2)

The PDF of \( B \) can be obtained by differentiating the above equation with respect to \( t \) as

\[
f_B(t) = N\beta(1-e^{-\beta t})^{N-1}e^{-\beta t}
\]

(3)

Then, \( b(x) \) is the convolution of the three PDFs: \( f(t) \), \( f_B(t) \), and \( g(t) \). Since the convolution is a linear operator, we compute first the convolution of \( f(t) \) and \( g(t) \), and then compute the convolution of the resulting function \( h(t) \) with \( f_B(t) \).

In (Salah, 2011), it has been shown that the convolution of two density functions with means of \( 1/r \) and \( 1/\mu \), can be expressed as

\[
h(t) = \begin{cases} \frac{\mu r}{\mu - r} e^{-rt} - e^{-\mu t} & \mu \neq r \\ \frac{\mu}{\mu - r} t & \mu = r \end{cases}
\]

(4)

Let us consider first the case where \( \mu \neq r \). \( b(x) \) is the convolution of \( h(t) \) and \( f_B(t) \) which can be expressed as

\[
b(x) = \frac{N\beta r}{\mu - r} \left[ (1-e^{-R})^{N-1} e^{-R(t)} dt - (1-e^{-\beta})^{N-1} e^{-\beta(t)} dt \right]
\]

(5)

where

\[
I_1 = \int_0^x (1-e^{-R})^{N-1} e^{-\beta(t)} e^{-\beta(t)} dt, \quad \text{and} \\
I_2 = \int_0^x (1-e^{-\beta})^{N-1} e^{-\beta(t)} e^{-\beta(t)} dt.
\]

Integrating by parts both \( I_1 \) and \( I_2 \), we get

\[
I_1 = e^{N\beta} \sum_{i=1}^{N} C_{i,N} e^{-(N-i)\beta} \left( \frac{(N-1)! r \beta^{N-1}}{(r - i \beta)} \right),
\]

\[
I_2 = e^{N\beta} \sum_{i=1}^{N} C_{i,N} e^{-(N-i)\beta} \left( \frac{(N-1)! r \beta^{N-1}}{(\mu - i \beta)} \right),
\]

where \( C_{i,N} = (-1)^i \binom{n-1}{i-1} \).

From Equation (1), \( \alpha_k = \frac{2^k}{k!} \int_0^x e^{-xk} b(x) dx \), and by substitution, we get

\[
\frac{\mu - r}{N\beta \mu} \alpha_k = \frac{2^k}{k!} \sum_{i=1}^{N} C_{i,N} \left( \frac{k!}{(r - i \beta)} \right) \int_0^x e^{-xk} e^{-N\beta} e^{-(N-x)\beta} dx
\]

\[
- (N-1)! \beta^{N-1} \frac{2^k}{k!} \sum_{i=1}^{N} C_{i,N} \left( \frac{k!}{(\mu - i \beta)} \right) \int_0^x e^{-xk} e^{-\beta} dx
\]

\[
+ (N-1)! \beta^{N-1} \frac{2^k}{k!} \sum_{i=1}^{N} C_{i,N} \left( \frac{k!}{(\mu - i \beta)} \right) \int_0^x e^{-xk} e^{-\mu} dx
\]

Integrating by parts, we get

\[
\int_0^x e^{-xk} e^{-N\beta} e^{-(N-x)\beta} dx = k!(\lambda + i\beta)^{-k-1}
\]

\[
\int_0^x e^{-xk} e^{-x \beta} dx = \frac{k!(\lambda + \mu)^{-k}}{\lambda + \mu}
\]

Therefore, we have

\[
\frac{\mu - r}{N\beta \mu} \alpha_k = \sum_{i=1}^{N} \frac{C_{i,N} \lambda^k (\lambda + i\beta)^{-k-1}}{(r - i \beta)}
\]

\[
- \sum_{i=1}^{N} (N-1)! \lambda \beta^{N-1} (\lambda + i\beta)^{-k-1}
\]

\[
+ \sum_{i=1}^{N} C_{i,N} \lambda^k (\lambda + i\beta)^{-k-1}
\]

\[
- \sum_{i=1}^{N} (N-1)! \beta^{N-1} \lambda^k (\lambda + \mu)^{-k-1}
\]

\[
+ \sum_{i=1}^{N} (N-1)! \lambda \beta^{N-1} \lambda^k (\lambda + \mu)^{-k-1}
\]
Or, equivalently,
\[
\alpha_k = \sum_{i=1}^{N} \frac{n_{r|i} \beta^{i+1} (\lambda + i \beta)^{k-1}}{(\mu - r)(r - i \beta)} - \sum_{i=1}^{N} \frac{n_{r|-1} \beta^i (\lambda + r)^{k-1}}{(\mu - r) \prod_{i=1}^{N} (r - i \beta)}
\]
\[
- \sum_{i=1}^{N} \frac{n_{r|i} \beta^{i+1} (\lambda + i \beta)^{k-1}}{(\mu - r)(r - i \beta)} + \sum_{i=1}^{N} \frac{n_{r|-1} \beta^i (\lambda + r)^{k-1}}{(\mu - r) \prod_{i=1}^{N} (r - i \beta)}
\]

Let us consider now the case where \( \mu = r \). Then \( b(x) \) can be expressed as
\[
b(x) = \mu^2 \int_{0}^{x} N\beta(1 - e^{-r \gamma}) N^{-1} e^{-r \gamma} (x - t)e^{-r(t-\gamma)} dt
\]
\[
= I_1 - I_2,
\]
where
\[
I_1 = N\beta \mu^2 e^{-\mu x} \int_{0}^{x} (1 - e^{-r \gamma}) N^{-1} e^{-r \gamma} (x - t)e^{-r(t-\gamma)} dt, \quad \text{and}
\]
\[
I_2 = N\beta \mu^2 e^{-\mu x} \int_{0}^{x} (1 - e^{-r \gamma}) N^{-1} e^{-r \gamma} (x - t)e^{-r(t-\gamma)} dt.
\]

Integrating by parts, we get
\[
I_1 = N\beta \mu^2 x e^{-\mu x} \sum_{i=1}^{N} \frac{C_{N|j}(1-e^{-x(i\beta - \mu)}) (i\beta - \mu)}{(i\beta - \mu)^2},
\]
\[
I_2 = N\beta \mu^2 e^{-\mu x} \sum_{i=1}^{N} \frac{C_{N|j} (-1+(i\beta - x\mu) e^{-x(i\beta - \mu)}) (i\beta - \mu)}{(i\beta - \mu)^2}.
\]

From Equation (1), we get
\[
\alpha_k = N\beta \mu^2 \sum_{i=1}^{N} \frac{(k+2)C_{N|j} (\lambda + i \beta)^{k+2} - (i \beta - \mu)^{k+2}}{(i \beta - \mu)^2}
\]
\[
- N\beta \mu^2 \sum_{i=1}^{N} \frac{C_{N|j} (\lambda + i \beta)^{k+2} + (i \beta - \mu)^{k+2} (k+1) \mu - i(k+2) \beta)}{(i \beta - \mu)^2}
\]

To summarize:
\[
\begin{align*}
\frac{d}{dt} \sum_{i=1}^{N} \frac{n_{r|i} \beta^{i+1} (\lambda + i \beta)^{k-1}}{(\mu - r)(r - i \beta)} - \sum_{i=1}^{N} \frac{n_{r|-1} \beta^i (\lambda + r)^{k-1}}{(\mu - r) \prod_{i=1}^{N} (r - i \beta)} & \quad \mu \neq r \\
- \sum_{i=1}^{N} \frac{n_{r|i} \beta^{i+1} (\lambda + i \beta)^{k-1}}{(\mu - r)(r - i \beta)} + \sum_{i=1}^{N} \frac{n_{r|-1} \beta^i (\lambda + r)^{k-1}}{(\mu - r) \prod_{i=1}^{N} (r - i \beta)}
\end{align*}
\]
\[
\sum_{i=1}^{N} \frac{(k+2)C_{N|j} (\lambda + i \beta)^{k+2} - (i \beta - \mu)^{k+2}}{(i \beta - \mu)^2} & \quad \mu = r
\]
\[
- N\beta \mu^2 \sum_{i=1}^{N} \frac{C_{N|j} (\lambda + i \beta)^{k+2} + (i \beta - \mu)^{k+2} (k+1) \mu - i(k+2) \beta)}{(i \beta - \mu)^2}
\]

Following the approach in (Takagi, 1993), we now use \( \alpha_k \) to compute the steady-state probabilities at the departure instants of jobs from the queue. To find the steady-state probabilities immediately after service completion, we need the transition probabilities of the imbedded Markov chain at steady state. We define the system state \( n_i \) as the number of jobs left behind in the system when the \( j \)th jobs departs. Note that \( n_i \) will range between 0 and K-1 since the departure of the jobs cannot leave the system completely full, i.e. with system state K.

The transition probabilities \( p_{jk} \) of the imbedded Markov chain at equilibrium can be found separately using \( \alpha_k \) for the two cases \( j = 0 \) and \( 1 \leq j \leq K - 1 \) as follows
\[
\begin{cases}
\alpha_k & 0 \leq k \leq K - 2 \\
\sum_{i=1}^{K-1} \alpha_i & k = K - 1
\end{cases}
\]

http://hipore.com/ijcc
Using the state transition probabilities of Equations (10) and (11), the steady state probabilities \( \pi_k \) at the departure instants can be computed by solving \( K-1 \) balance equations along with the normalization condition as follows

\[
\pi_k = \sum_{j=0}^{K-1} \pi_j P_{jk} \quad 0 \leq k \leq K-1
\]

\[
\sum_{k=0}^{K-1} \pi_k = 1 \quad \text{(Normalization Condition)}.
\]

Substituting Equations (10) and (11) into (12), we get

\[
\pi_k = \pi_0 \alpha_k + \sum_{j=0}^{K-1} \pi_j \alpha_{k-j+1} \quad 0 \leq k \leq K-2
\]

\[
\sum_{k=0}^{K-1} \pi_k = 1
\]

Note with the normalized condition, only \( K-2 \) equations are needed from Equation (12) to solve the set of linear equations to get the corresponding state probabilities. To solve the state probabilities \( \pi_k \), we solve first for the normalized variables \( \frac{\pi_k}{\pi_0} \) using

\[
\frac{\pi_{k+1}}{\pi_0} = \frac{1}{\alpha_0} \left[ \frac{\pi_k}{\pi_0} - \sum_{j=0}^{K-1} \pi_j \frac{\alpha_{k-j+1}}{\alpha_0} - \alpha_k \right] \quad 0 \leq k \leq K-2
\]

Equation (13) can be solved recursively. In other words, we can successively determine \( \pi_1/\pi_0, \pi_2/\pi_0, \ldots, \pi_{K-1}/\pi_0 \).

Subsequently, \( \pi_0 \) can be solved using the normalization condition as follows

\[
\pi_0 = \frac{1}{\sum_{k=0}^{K-1} \pi_k / \pi_0} = \frac{1}{1 + \sum_{k=1}^{K-1} \pi_k / \pi_0}
\]

Determining \( \pi_0 \) would allow us to obtain the actual state probabilities \( \{\pi_k; 0 \leq k \leq K-1\} \). Let \( p_k \) be the probability that there are \( k \) jobs present in the system at an arbitrary time, where \( k = 0, 1, 2, \ldots, K \). Using \( P_{loss} \) as the equilibrium probability that an arrival is lost (or blocked) because the queue is full, i.e., in state \( K \) where \( P_{loss} = p_K \), we can express

\[
p_k = (1 - P_{loss}) \pi_k \quad 0 \leq k \leq K-1.
\]

\( \overline{X} \) is the mean service time which is basically the sum of the mean service time of all stages, and can be expressed as

\[
\overline{X} = 1/\mu + E[B] + 1/\gamma,
\]

where \( E[B] \) is the mean service time of the second stage and expressed in the derived Equation (23).

We can express \( P_{loss} \) from (8) as

\[
P_{loss} = p_K = 1 - \frac{1 - p_0}{\rho} = \frac{p_0 + \rho - 1}{\rho}
\]

where \( \rho = \lambda \overline{X} \).

The departure rate \( \gamma \) also be expressed as the effective arrival rate \( \lambda' \) which is \( \lambda (1 - P_{loss}) \). Therefore,

\[
\gamma = \lambda' (1 - P_{loss})
\]

Using the values of \( \pi_k \) obtained from Equation (13) and the results of (15) and (17), the equilibrium state distribution \( \{\pi_k; 0 \leq k \leq K-1\} \) can be expressed as

\[
p_k = \frac{\pi_k}{\pi_0 + \rho} \quad 0 \leq k \leq K-1.
\]

From Equation (19), the probability \( p_0 \) of finding the system empty can be expressed as

\[
p_0 = \frac{\pi_0}{\pi_0 + \rho}.
\]

The mean number \( \overline{K} \) in the system can be derived as

\[
\overline{K} = \sum_{k=0}^{K} kp_k = \sum_{k=0}^{K-1} kp_k + Kp_K
\]

\[
= \sum_{k=0}^{K-1} kp_k + K(1 - p_0)
\]

\[
\overline{K} = \frac{1}{\pi_0 + \rho} \sum_{k=0}^{K-1} k\pi_k + K\left(1 - \frac{1}{\pi_0 + \rho}\right).
\]

The mean number \( \overline{Q} \) in the queue can be expressed as

\[
\overline{Q} = \sum_{k=0}^{K-1} kp_k = \frac{1}{\pi_0 + \rho} \sum_{k=0}^{K-1} k\pi_k + (K-1)\left(1 - \frac{1}{\pi_0 + \rho}\right)
\]

Using Equation (22), the utilization of the queue can be expressed as

\[
\overline{Q}_{\text{utilization}} = \frac{\overline{Q}}{\overline{K}}.
\]
Using Little's result, the mean time spent in the system by a jobs succeeding in entering the queue can be expressed as
\[ W = \frac{K}{\lambda} = \frac{1}{\lambda} \sum_{k=1}^{K} k \pi_k + \frac{K}{\lambda} (\pi_0 + \rho - 1). \] (24)

To derive the formula for the mean service time \( E[B] \) of the second stage, we use Equation (3) to express \( E[B] \) as
\[
E[B] = \int_0^\infty f_B(t) dt = \int_0^\infty N \beta (1 - e^{-\beta})^{N-1} e^{-\beta t} dt
\]

Using integration by parts with \( u = (1 - e^{-\beta})^{N-1} \) and \( dv = \lambda e^{-\beta t} dt \), then
\[
du = N(1 - e^{-\beta})^{N-1} dt + N(N-1) \lambda t (1 - e^{-\beta})^{N-2} e^{-\beta t} dt
\]
and \( v = -e^{-\beta t} \). Since \( N(1 - e^{-\beta})^{N-1} t e^{-\beta t} \mid_0^\infty = 0 \) and
\[
\int_0^\infty N(1 - e^{-\beta})^{N-1} e^{-\beta t} dt = \frac{1}{\beta},
\]
the main integral can be written as
\[
E[B] = \frac{1}{\lambda} + \int_0^\infty N(N-1) \beta t (1 - e^{-\beta})^{N-2} e^{-2\beta t} dt
\]
\[
= \frac{1}{\lambda} + I_1,
\]
where \( I_1 = \int_0^\infty N(N-1) \beta t (1 - e^{-\beta})^{N-2} e^{-2\beta t} dt \).

Similarly, using integration by parts to compute \( I_1 \), we get
\[
I_1 = \frac{1}{2\beta} + \frac{1}{2} \int_0^\infty N(N-1)(N-2) \beta t (1 - e^{-\beta})^{N-3} e^{-3\beta t} dt
\]
\[
= \frac{1}{2\beta} + I_2,
\]
where
\[
I_2 = \int_0^\infty N(N-1)(N-2) \beta t (1 - e^{-\beta})^{N-3} e^{-3\beta t} dt
\]
We continue this process of integration by parts \( k \) times, where \( k = 1, 2, \ldots, N-1 \) to get
\[
E[B] = \frac{1}{\beta} + \frac{1}{2\beta} + \frac{1}{3\beta} + \ldots + \frac{1}{k\beta}
\]
\[
+ \int_0^\infty \frac{N!}{k!(N-k-1)!} \beta t (1 - e^{-\beta})^{N-k-1} e^{-(k+1)\beta t} dt
\]
Letting \( k = N - 1 \), we get
\[
E[B] = \sum_{i=1}^{N-1} \frac{1}{i\beta} + \int_0^\infty N \beta e^{-N\beta t} dt
\]
Now, using the integration by parts technique, it is easy to verify that the integral \( \int_0^\infty N \beta e^{-N\beta t} dt = \frac{1}{N\beta} \).

Therefore,
\[
E[B] = \frac{1}{\beta} \sum_{i=1}^{N} \frac{1}{i}.
\] (25)

5. NUMERICAL RESULTS

In this section, we report numerical results to illustrate how our analytical model can be used in predicting the minimal number of worker VMs needed to satisfy a given SLO response time. We also give examples to show how our model can aid in achieving proper elasticity. We also give examples that will aid in better understanding of the overall system so that sound engineering decisions and choices can be made. Some key engineering decisions may include the size of the queue and the compute capacity for the JS, workers, and RA.

For our numerical examples, we choose \( K=100, \mu = 50 \text{ ms} \) and \( r = 20 \text{ ms} \). The mean service times for the JS and RA nodes are realistic and consistent with the reported experimental measurements for cloud servers and nodes reported in (Bi, Zhu, Tian, & Wang, 2010; Dejun, Pierre, & Chi, 2009; Islam et al., 2012). The service times include processing and network communication and database access delays. The mean service time for a worker VM (i.e. \( 1/\beta \)) depends on two factors: (1) \( N \) which is the number of worker VMs, and (2) the execution speed of each worker VM. If we assume a cluster job takes 250 ms to be executed on a single worker VM, and with homogenous splitting, \( 1/\beta \) for a single worker VM can be computed as \( 1/\beta = 250/N \text{ ms} \).

Results are shown from both analysis and simulation. It is clear from Figures 3-6 that the simulation results are very much in line with those of analysis. This implies that our analytical model is verifiable and correct. Our simulation is a DES (discrete-event simulation) written in C language. The DES captures the system dynamism and behavior of a finite queuing system wherein job arrivals are queued and then serviced sequentially in three stages by: the JS, worker VMs, and the RA. The servicing in the three stages is carried out in a sequential manner such that only one stage is active at a time. To verify the correctness of our analytical model, we considered the same assumptions as those of analysis. In our simulation, we followed the guidelines given in (Law & Kelton, 1991).
We first demonstrate how our model can be used in determining the proper size of the queue in relation to incoming workload. The queue size for incoming requests is a key engineering design decision. For the queue size, engineers aim at ensuring that the queue is not fully utilized at high workload so that incoming requests are not dropped when a spike of the workload is encountered. Typically, an overall queue utilization below 70% is desirable to compensate for a sudden surge in the workload. Figure 3 plots the queue utilization of a queue size of 100 (K=100) in relation to incoming workload. We plot the queue utilization from Equation (23) for different N in relation to the average workload $\lambda$. We select the number of VM workers N to have values of 1, 2, 3, and 5, and we vary the workload $\lambda$ from 0 to 5 Jobs/s. A number of observations can be made from Figure 3. It is clear that when N is small the queue gets highly utilized when subjected to relatively low workload. For example at a workload $\lambda$ of 3.5 Jobs/s, using a single VM worker will result in almost 90% utilization when compared to 10% utilization when using N=2. For high workload $\lambda$ of 4.5 Jobs/s, the VM workers have to increase to almost 4 and 5 in order to keep the queue utilization below the desirable level of 70%.

An important SLO parameter that is required to ensure proper cloud service is the request loss probability or $P_{loss}$.
From the curves exhibited in Figures 3-5, we can identify clearly the saturation points in the system. The saturation is the point where the workload approaches approximately the processing capacity in the system, i.e. \( \rho = \lambda X \) as derived in Equation (17). For example, for a single worker VM where \( N = 1 \), the saturation point can be identifying when approximately \( \lambda = 3 \). For two worker VMs, where \( N = 2 \), the saturation point is reached when approximately \( \lambda = 3.75 \). For engineers and designers, it is a good design practice to keep the system below the situation point in order to achieve a system good performance. If not, excessive latency and request loss will be encountered, as shown in Figures 4 and 5.

Figure 6 is a zoom-in version of Figure 5, showing SLO response times for additional VM workers with \( N = 10 \) and \( N = 20 \). Overall, as shown in the figure, as the workload increases the response time increases sharply with a small number of worker VMs. This figure can be used to determine the minimal required number \( N \) of worker VMs to guarantee a given SLO response time. For example, to sustain an average SLO response time of 300 ms, at a light workload (\( \lambda < 1.5 \)), we require \( N \) to be smaller than 5. However at high workload (\( \lambda > 2.5 \)), \( N \) has to be greater than 5. It is clearly shown that at any given time the required \( N \) is dependent on the SLO response time as well as the current mean workload conditions.

Figure 7 illustrates how our analytical model can be used to achieve agile elasticity whereby an SLO response time (of 300 ms) is satisfied at any given time while minimizing the number of cloud resources, i.e. worker VMs. The top subfigure shows a highly fluctuating workload with a pattern of increase and decline in demand. The middle subfigure exhibits the dynamically changing and minimum number of worker VMs (labeled as Elastic N) computed by our analytical model every 0.25 seconds to ascertain an SLO response time below 300 ms. The bottom subfigure shows the corresponding mean SLO response time when using Elastic N and constant N. With a fixed number of VM workers (i.e. constant N=10), it is clearly observed that improper elasticity is achieved in which the SLO response time is violated in addition to undesirable resource utilization in which over-provisioning and under-provisioning are exhibited. Overall, we feel that our model can be extremely useful in achieving proper elasticity. In the given numerical examples, we have selected an interval of 0.25 seconds to estimate N. However, further investigation is needed to test this parameter (Ali-Eldin et al., 2012a; Ali-Eldin et al., 2012b). In addition, further investigation is needed to predict and estimate the mean workload with diverse patterns of slow and sharp fluctuations, growth, and decline. Prior work in the following studies (Ali-Eldin et al., 2012a; Salah & Haidari, 2010; Salah, Haidari, Bahjat, & Manaan, 2009) has been reported on estimating a proper mean \( \lambda \) of highly fluctuating loads.

derived in Equation (17). For the cloud customer, this SLO parameter would ensure that the cloud provider would be able to service incoming requests by ensuring no request is being dropped, or if there is a dropping, it will be dropped by a very small probability. Figure 4 illustrates how our model can predict on average this request loss probability given the current workload and the number of provisioned VMs and their processing capacity. The figure plots the loss probability for different N of worker VMs. As shown in the figure, almost no request gets dropped at light workload for different N worker VMs. For example, when the workload \( \lambda \) is below 2.5 Jobs/s, the \( P_{\text{loss}} \) stays very close to zero. As the workload \( \lambda \) increases beyond 2.5 Jobs/s, we start noticing that the \( P_{\text{loss}} \) starts increasing. The severity of this increase is primarily impacted by the number of worker VMs. As it can be observed from the figure, with a single worker VM, the request loss probability is the highest beyond 2.5 Jobs/s. In fact, with a single worker VM, at a workload of 2.75 Jobs/s, the probability of request loss starts to increase sharply, and therefore resulting in an unacceptable performance leading to clear violation of the SLO request loss criterion. The figure clearly shows at high workload, the worker VMs have to be increased in order to satisfy an acceptable level of the SLO request loss.

Another key SLO criterion of a paramount importance is the response time. This SLO performance criterion usually tops the list of the SLO requirements set by the cloud users for the cloud providers to guarantee. To better understand this performance metric, we plot the overall system response time \( W \) derived in Equation (24) in relation to incoming workload \( \lambda \). As the case for Figures 3 and 4, we use the same workload range and we select the number of VM workers N to have values of 1, 2, 3, and 5. As, expected, it is observed from Figure 5 that as the workload increases, the average response time increases for all values of N with a single VM worker exhibiting the largest response time. As shown, the curves take an “S” shape wherein at a light workload, the smallest response times are exhibited for all values of N. As the workload increases, the response time starts increases sharply and then it flattens off reaching a plateau where the response time stays almost constant despite the increase in the incoming workload. The sharp increase is due to reaching near the saturation point in the system where the queue utilization as well as the request loss starts to increase as previously demonstrated in Figure 3 and Figure 4, respectively. The plateau state of the response time, despite the increase of the workload, is attributed to having the system operating at saturation point, where the response time is counted only for the successful packets entering the queue. As was exhibited in Figure 3, when the workloads go beyond the saturation points, the queue utilization curves reach almost 100%.
6. CONCLUSIONS

We have presented an analytical model based on queuing theory to determine at any given workload conditions the minimum number of compute resources needed for executing highly parallelized jobs on a cloud cluster. We have derived formulas for key SLO performance metrics such as response time, request loss/blocking probability, queue utilization, and throughput. We have shown that our analytical results are in good agreement with results obtained by simulation. As has been demonstrated by the given numerical examples, our analytical model can be extremely useful in achieving proper elasticity for cloud-based clustering systems. As a future work, plans are underway to build a cloud cluster with a Job Scheduler (JS) utilizing our derived analytical formulas to estimate the minimal number of VM resources needed to satisfy the SLO response time. The elasticity and the performance of such a cloud cluster will be evaluated. Moreover, we plan to devise novel schemes to best estimate a suitable interval to compute the Elastic N (representing the minimum number of required cloud compute resources) as well as to best estimate the mean workload characterized by considerably high degree of fluctuation.

7. REFERENCES


**Author**

Khaled Salah is an associate professor in the Department of Electrical and Computer Engineering at Khalifa University of Science, Technology and Research (KUSRAR), UAE. He received the B.S. degree in Computer Engineering with a minor in Computer Science from Iowa State University, USA, in 1990, the M.S. degree in Computer Systems Engineering from Illinois Institute of Technology, USA, in 1994, and the Ph.D. degree in Computer Science from the same institution in 2000. Prior to joining KUSTAR, he was for 10 years with the department of Information and Computer Science, King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia. Khaled has over 90 publications (out of which 40 WOS-listed journal articles) in the areas of network performance modeling and simulation, network security, computer networks, and operating systems. He was the recipient of KFUPM University Excellence in Research Award of 2008/09, the recipient of KFUPM Best Research Project Award of 2009/10. Khaled is a senior member of IEEE, and has been serving on the editorial boards of a number of WOS-listed journals including IET Networks, Wiley’s SCN, Wiley’s IJNM, Elsevier’s JNCA, J.UCS, and AJSE.
IEEE 7th International Conference on Cloud Computing (CLOUD 2014)


IEEE 11th International Conference on Services Computing (SCC 2014)

SCC 2014 will focus on services innovation lifecycle e.g., enterprise modeling, business consulting, solution creation, services orchestration, optimization, management, and BPM. Visit http://conferences.computer.org/scc/.

IEEE 21th International Conference on Web Services (ICWS 2014)


IEEE 3rd International Conference on Mobile Services (MS 2014)

MS 2014 will feature Wearable technology and applications. Topics cover but not limited to all innovative aspects of wearable devices, programming models, integration, and domain specific solutions. Visit http://themobileservices.org/2014.

IEEE 10th World Congress on Services (SERVICES 2014)

June 27—July 2, 2014, Anchorage, Alaska, USA

Federation of 5 Service-Centric Conferences from Different Angles (http://www.servicescongress.org)

International Congress on Big Data (BigData 2014)

BigData 2014 aims to explore various aspects of Big Data including modeling, storage architecture (NOSQL), enterprise transformation, text mining, social networks, applied analytics and various applications, Big Data As A Service, Visit http://ieeebigdata.org/2014.


Conference proceedings are EI indexed. Extended versions of invited ICWS/SCC/CLOUD/MS/BigData papers will be published in IEEE Transactions on Services Computing (TSC, SCI & EI indexed), International Journal of Web Services Research (JWSR, SCI & EI indexed), International Journal of Business Process Integration and Management (IJBPIM), IEEE IT Pro (SCI & EI Indexed).

Contact: Liang-Jie Zhang (LJ) zhanglj@ieee.org
(Steering Committee Chair)

Submission Deadlines
ICWS 2014: 1/15/2014
CLOUD 2014: 1/15/2014
SCC 2014: 1/31/2014
MS 2014: 1/31/2014
BigData 2014: 2/15/2014
SERVICES 2014: 2/15/2014
Call for Articles
International Journal of Services Computing

Mission
The International Journal of Services Computing (IJSC) aims to be a reputable resource providing leading technologies, development, ideas, and trends to an international readership of researchers and engineers in the field of Services Computing. To ensure quality, IJSC only considers extended versions of papers published at reputable international conferences such as IEEE ICWS.

From the technology foundation perspective, Services Computing covers the science and technology needed for bridging the gap between Business Services and IT Services, theory and development and deployment. All topics regarding Web-based services lifecycle study and management align with the theme of IJSC. Specially, we focus on:
1) Web-based services, featuring Web services modeling, development, publishing, discovery, composition, testing, adaptation, and delivery, and Web services technologies as well as standards; 2) services innovation lifecycle that includes enterprise modeling, business consulting, solution creation, services orchestration, services optimization, services management, services marketing, business process integration and management; 3) cloud services featuring modeling, developing, publishing, monitoring, managing, delivering XaaS (everything as a service) in the context of various types of cloud environments; and 4) mobile services featuring development, publication, discovery, orchestration, invocation, testing, delivery, and certification of mobile applications and services.

Topics
The International Journal of Services Computing (IJSC) covers state-of-the-art technologies and best practices of Services Computing, as well as emerging standards and research topics which would define the future of Services Computing. Topics of interest include, but are not limited to, the following:

- Services Engineering
- XaaS (everything as a service)
- Cloud Computing for Internet-based services
- Big Data services
- Internet of Things (IoT) services
- Pervasive and Mobile services
- Social Networks and Services
- Wearable services
- Web 2.0 and Web X.0 in Web services
- Service-Oriented Architecture (SOA)
- RESTful Web Services
- Service modeling and publishing
- Service discovery, composition, and recommendation
- Service operations, management, and governance
- Services validation and testing
- Service privacy, security, and trust
- Service deployment and evolution
- Semantic Web services
- Scientific workflows
- Business Process Integration and management
- Service applications and implementations
- Business intelligence, analytics and economics for Services
Call for Articles
International Journal of Big Data

Mission
Big Data has become a valuable resource and mechanism for the practitioners and researchers to explore the value of data sets in all kinds of business scenarios and scientific investigations. New computing platforms such as cloud computing, mobile Internet, social network are driving the innovations of big data. From government initiative perspective, Obama Administration in United States launched "Big Data" initiative that announces $200 Million in new R&D investments on March 29, 2012. European Union also announced "Big Data at your service" on July 25, 2012. From industry perspective, IBM, SAP, Oracle, Google, Microsoft, Yahoo, and other leading software and internet service companies have also launched their own innovation initiatives around big data.

The International Journal of Big Data (IJBD) aims to provide the first Open Access publication channel for all authors working in the field of all aspects of Big Data. Big Data is a dynamic discipline. One of the objectives of IJBD is to promote research accomplishments and new directions. Therefore, IJBD welcomes special issues in any emerging areas of big data.

Topics
IJBD includes topics related to the advancements in the state of the art standards and practices of Big Data, as well as emerging research topics which are going to define the future of Big Data. Topics of interest to include, but are not limited to, the following:

Big Data Models and Algorithms (Foundational Models for Big Data, Algorithms and Programming Techniques for Big Data Processing, Big Data Analytics and Metrics, Representation Formats for Multimedia Big Data)

Big Data Architectures (Cloud Computing Techniques for Big Data, Big Data as a Service, Big Data Open Platforms, Big Data in Mobile and Pervasive Computing)

Big Data Management (Big Data Persistence and Preservation, Big Data Quality and Provenance Control, Management Issues of Social Network enabled Big Data)

Big Data Protection, Integrity and Privacy (Models and Languages for Big Data Protection, Privacy Preserving Big Data Analytics Big Data Encryption)

Security Applications of Big Data (Anomaly Detection in Very Large Scale Systems, Collaborative Threat Detection using Big Data Analytics)

Big Data Search and Mining (Algorithms and Systems for Big Data Search, Distributed, and Peer-to-peer Search, Machine learning based on Big Data, Visualization Analytics for Big Data)
