2013

Service Isolation vs. Consolidation: Implications for Iaas Cloud Application Deployment

Wes Lloyd
University of Washington Tacoma, wlloyd@uw.edu

Shrideep Pallickara

Olaf David

Jim Lyon

Mazdak Arabi

See next page for additional authors

Authors
Wes Lloyd, Shrideep Pallickara, Olaf David, Jim Lyon, Mazdak Arabi, and Ken Rojas

Service Isolation vs. Consolidation: Implications for IaaS Cloud Application Deployment

Wes Lloyd[1,2], Shrideep Pallickara[1], Olaf David[1,2], Jim Lyon[2], Mazdak Arabi[2]
[1]Department of Computer Science
[2]Department of Civil and Environmental Engineering
Colorado State University, Fort Collins, USA
wes.lloyd, shrideep.pallickara, olaf.david, jim.lyon, mazdak.arabi@colostate.edu

Ken Rojas
USDA-Natural Resource Conservation Service
Fort Collins, Colorado USA
Ken.Rojas@ftc.usda.gov

***Abstract*— Service isolation, achieved by deploying components of multi-tier applications using separate virtual machines (VMs), is a common "best" practice. Various advantages cited include simpler deployment architectures, easier resource scalability for supporting dynamic application throughput requirements, and support for component-level fault tolerance. This paper presents results from an empirical study which investigates the performance implications of component placement for deployments of multi-tier applications to Infrastructure-as-a-Service (IaaS) clouds. Relationships between performance and resource utilization (CPU, disk, network) are investigated to better understand the implications which result from how applications are deployed. All possible deployments for two variants of a multi-tier application were tested, one computationally bound by the model, the other bound by a geospatial database. The best performing deployments required as few as 2 VMs, half the number required for service isolation, demonstrating potential cost savings with service consolidation. Resource use (CPU time, disk I/O, and network I/O) varied based on component placement and VM memory allocation. Using separate VMs to host each application component resulted in performance overhead of ~1-2%. Relationships between resource utilization and performance were harnessed to build a multiple linear regression model to predict performance of component deployments. CPU time, disk sector reads, and disk sector writes are identified as the most powerful performance predictors for component deployments.**

Keywords Service Isolation; Service Composition; Infrastructure-as-a-Service; Provisioning; Virtualization; Multi-Tenancy; Resource Management and Performance;

I. INTRODUCTION

Migration of multi-tier client/server applications to Infrastructure-as-a-Service (IaaS) clouds involves deploying components of application infrastructure to one or more virtual machine (VM) images. Images are used to instantiate VMs to provide the application's cloud-based infrastructure. Application components consist of infrastructure elements such as web/application servers, proxy servers, NO SQL databases, distributed caches, relational databases, file servers and others. *Service isolation* refers to the total separation of application components with deployment using separate VMs. VMs are then hosted by one or more physical machines in an IaaS cloud. The non-determinism of the mapping of VMs to physical hosts is a concept known as *provisioning variation*, which may lead to unpredictable performance behavior [1-2]. Service isolation provides application components with their own explicit sandboxes to operate in with a separate VM and operating system instance. With *hardware virtualization* isolation can be accomplished many times for each component of an application across a cluster of physical servers. Before server virtualization, service isolation of application components using physical machines required significant server capacity.

Methods to assess performance effects from VM interference and investigation of approaches to better manage multiplexing resources of physical hosts are active areas of research [8-10, 26, 33-35]. Physical resources including CPU, disk, and network I/O bandwidth are shared and appropriate quantities must be allocated to meet application service-level agreements (SLAs). With current virtualization technology, only memory isolation is guaranteed. VMs reserve a fixed amount of memory for exclusive use which is not released until shutdown. Processor, network I/O, and disk I/O resources are multiplexed and sharing is coordinated by the virtualization hypervisor. Popular virtualization hypervisors include kernel-based VMs (KVM), Xen, and the VMware ESX hypervisor. Hypervisors vary with respect to methods used to multiplex resources. Some allow pinning VMs to operate using specific CPU cores to guarantee resource availability, but CPU caches must be shared [26].

Service isolation has been suggested as a best practice for deploying components of multi-tier applications across VMs An Amazon Web Services best practices white paper suggests decoupling components by "bundling the logical construct of a component into an Amazon Machine Image so that it can be deployed more often" [36]. This loose component coupling which isolates the various layers of an application to enable horizontal scaling (increasing the # of VMs) is encouraged in the white paper to enable easy resource scalability. Service isolation enables scalability and supports fault tolerance at the component level. Isolating components may reduce inter-component interference allowing them to run more efficiently though this isolation is only at the guest operating system level as VMs share physical hardware resources and compete for CPU, disk, and

network bandwidth. Service isolation adds an abstraction layer above the physical hardware which introduces overhead potentially degrading performance. Deploying all application components using separate VMs may increase network traffic, particularly when VMs are hosted by separate physical machines. Component deployment should be done carefully considering both virtualization overhead and the effects of physical placement. Consolidating key components together on a single VM guarantees they will not be physically separated when deployed providing an opportunity for potential performance improvements which may potentially provide performance enhancements. User control of VM placement is not always provided by IaaS clouds resulting in provisioning variation which has been shown to result in inconsistent performance [1-3].

This paper presents results of our investigation on the implications of isolation versus consolidation for deploying components of multi-tier applications to IaaS clouds. The following research questions are investigated:

RQ-1) *What are the impacts on resource utilization and application performance resulting from different component deployments (isolation versus consolidation) for multi-tier applications? How does increasing VM memory allocation impact Performance?*

RQ-2) *How much overhead results from VM service isolation?*

RQ-3) *Can VM resource utilization data be used to build models to predict performance of component deployments?*

This paper presents a thorough and detailed investigation on how the deployment of multi-tier application components impacts application performance and resource consumption (CPU, disk, network). This work extends prior research on provisioning variation and heterogeneity of cloud-based resources. Relationships between component placement, resource utilization and application performance are investigated. Benchmarks are made to measure performance effects of increasing VM memory allocation and to quantify overhead of service isolation. This work leads to the development of a multiple linear regression model using resource utilization statistics to predict performance of application component deployments. Our approach for collecting application resource utilization data discussed in section III to construct performance model(s) uses a simple Linux script and can be generalized to any multi-tier application. Supporting autonomic identification of good component deployments through modeling can help reduce application hosting costs (# of VMs) and improve management and load balancing of physical cloud resources without sacrificing performance goals.

II. Related Work

Rouk first identified the challenge of finding ideal service compositions for creating virtual machine images in cloud environments in [4]. Schad et al. [2] demonstrated the unpredictability of Amazon EC2 VM performance caused by contention for physical machine resources and provisioning variation of VMs. Using a Xen-based private cloud Rehman et al. tested the effects of resource contention on Hadoop-based MapReduce performance by using IaaS-based cloud VMs to host worker nodes [1]. They tested provisioning variation of three different deployment schemes of VM-hosted Hadoop worker nodes and observed performance degradation when too many worker nodes were physically co-located. Their work investigated VM deployments not for multi-tier application(s), but for MapReduce jobs where all VMs were homogeneous in nature. Multi-tier applications with many heterogeneous components present a more complex challenge for resource provisioning than studied by Rehman et al. Zaharia et al. identified that Hadoop's scheduler can cause severe performance degradation from being unaware of resource contention when Hadoop nodes are hosted by Amazon EC2 VMs [3]. They improved upon Hadoop's scheduler with the Longest Approximate Time to End (LATE) scheduling algorithm which better addresses performance variations of heterogeneous Amazon EC2 VMs. Their work also did not consider hosting of heterogeneous components.

Camargos et al. investigated performance of virtualization hypervisors for virtualizing Linux servers with numerous performance benchmarks for CPU, file and network I/O [5]. Hypervisors tested included Xen, KVM, VirtualBox, and two container based virtualization approaches OpenVZ and Linux V-Server. Their benchmarks targeted different parts of the system using kernel compilation, file transfers, and file compression. Armstrong and Djemame investigated performance of VM image propagation using Nimbus and OpenNebula, two IaaS cloud infrastructure managers [6]. Additionally they benchmarked Xen and KVM paravirtualized I/O. Jayasinghe et al. investigated performance implications of deploying the RUBBoS n-tier e-commerce system using three different IaaS clouds: Amazon EC2, Emulab, and Open Cirrus [7]. They tested horizontal scaling, changing the number of VMs for each component, and vertical scaling, varying the resource allocations of VMs. They deployed components using separate VMs for full service isolation and did not investigate consolidation of components. Matthews et al. developed a VM isolation benchmark which quantified the isolation level when co-located VMs ran several conflicting tasks [8]. They tested VMWare, Xen, and OpenVZ hypervisors to quantify isolation. Somani and Chaudhary benchmarked Xen VM performance with two and four co-located VMs running CPU, disk, or network intensive tasks on a single physical host [9]. They benchmarked the Simple Earliest Deadline First (SEDF) I/O credit scheduler vs. the default Xen credit scheduler. Physical resource contention was investigated when running different co-located tasks, a scenario which may occur for component deployments of multi-tier applications. Raj et al. improved hardware level cache management of the Hyper-V hypervisor introducing VM core assignment and cache portioning to reduce inter-VM conflicts from sharing the same hardware caches. These improvements were shown to improve VM isolation [10].

Niehörster et al. developed an autonomic system using support vector machines (SVM) where service specific agents were used to provide horizontal and vertical scaling of virtualization resources hosted by an IaaS Eucalyptus cloud [11]. Their agents scaled # of VMs, memory, and virtual core allocations to meet quality-of-service goals using their SVM modeling approach. They applied their approach to scale the number of modeling engines for GROMACS, a molecular dynamics simulation and also to scale Apache web application servers to meet QoS goals. Sharma et al. investigated implications of physical placement of non-parallel tasks and their resource requirements to build performance model(s) to aid task scheduling and distribution on compute clusters [32]. Like Sharma we are interested in understanding implications of resource requirements and physical host location but for application components, not non-parallel sequential tasks. RQ-3 specifically investigates building performance models which could aid component placement.

Previous studies have investigated virtualization performance issues and autonomic scaling of VMs, but none have investigated implications of component placement (isolation vs. consolidation) relative to application performance and physical resource load balancing (CPU, disk, network) for IaaS cloud hosting of multi-tier applications.

III. Experimental Investigation

A. Test Application

For our investigation we utilized two variants of a popular soil erosion model known as RUSLE2 (Revised Universal Soil Loss Equation – Version 2) [12]. RUSLE2 contains both empirical and process-based science that predicts rill and interrill soil erosion by rainfall and runoff. RUSLE2 was developed to guide conservation planning, inventory erosion rates, and estimate sediment delivery. RUSLE2 is the US Department of Agriculture Natural Resources Conservation Service (USDA-NRCS) agency standard model for sheet and rill erosion modeling used by over 3,000 field offices across the United States. RUSLE2 consists of four tiers including an application server, a geospatial relational database, a file server, and a logging server utilizing a non-network file-based relational database. RUSLE2 is a good multi-component application for our investigation because with four components (Table I) and 15 possible deployments (Table II), it is both complex enough to be interesting, yet simple enough that brute force testing is reasonable to accomplish. RUSLE2's architecture is a surrogate for traditional client/server architectures having both an application and relational database.

RUSLE2 was originally developed as a Windows-based Microsoft Visual C++ desktop application and has been extended to provide soil erosion modeling as a JAX-RS RESTful webservice hosted by Apache Tomcat [16] using JSON as the transport protocol for data objects. To facilitate functioning as a web service a command line console was added. The Object Modeling System 3.0 (OMS3) framework [13-14] using WINE [15] provides middleware to facilitate interacting with the console. OMS3 was developed by the USDA–ARS in cooperation with Colorado State University and supports component-oriented simulation model development in Java, C/C++ and FORTRAN.

The RUSLE2 web service supports ensemble runs which are groups of individual model requests bundled together. To invoke the RUSLE2 web service a client sends a JSON object with parameters describing land management practices, slope length, steepness, latitude, and longitude. Model results are returned as JSON objects. Ensemble runs are processed by dividing sets of modeling requests into individual requests which are resent to the web service, similar to the "map" function of MapReduce. These requests are distributed to worker nodes using a round robin proxy server. Upon completion individual runs of the ensemble are "reduced" into a single JSON response object. A simple test generation program was used to create randomized ensemble tests. Latitude and longitude coordinates were randomly selected within a bounding box from the state of Tennessee. Slope length, steepness, and land management practice parameters were randomized. Randomization of latitude and longitude coordinates led to variable geospatial query execution times because the polygons intersected with varied in complexity. To verify that our test generation technique produced test sets with variable complexity, 20 randomly generated 100-model run ensemble tests were run using the 15 RUSLE2 component deployments twice and average execution times were calculated. Execution speed (slow/medium/fast) of ensemble tests was preserved across subsequent runs indicating that individual ensembles exhibited a complexity-like characteristic ($R^2=.914$, $df=18$, $p=5 \cdot 10^{-11}$).

TABLE I. RUSLE2 Application Components

Component		Description
$\mathcal{M}$	Model	Apache Tomcat 6.0.20, Wine 1.0.1, RUSLE2, Object Modeling System (OMS 3.0)
$\mathcal{D}$	Database	Postgresql-8.4, PostGIS 1.4.0-2 Geospatial database consists of soil data (1.7 million shapes, 167 million points), management data (98 shapes, 489k points), and climate data (31k shapes, 3 million points), totaling 4.6 GB for the state of TN.
$\mathcal{F}$	File server	nginx 0.7.62 Serves XML files which parameterize the RUSLE2 model. 57,185 XML files consisting of 305MB.
$\mathcal{L}$	Logger	Codebeamer 5.5 w/ Derby DB, Tomcat (32-bit) Custom RESTful JSON-based logging wrapper web service. IA-32libs support operation in 64-bit environment.

Our investigation utilized two variants of RUSLE2 which are referred to as "d-bound" for the database bound variant and "m-bound" for the model bound variant. The variants are named based on the predominant component requiring the largest quantity of execution time. By testing two variants of RUSLE2 the same test harness can be used to investigate implications of component deployments for both applications. These application variants provide surrogates for two potentially common scenarios in practice: an application bound by the database tier, and an application bound by the middleware (model) tier. For the "d-bound"

version of RUSLE2 two primary geospatial queries were modified to perform a join on a nested query. The "m-bound" variant was unmodified. The "d-bound" application exhibits a different application profile than the "m-bound" RUSLE2. On average the "d-bound" application requires ~2.45x more CPU time than the "m-bound" variant.

B. Application Services

Table I describes the application components of RUSLE2's application stack. The $\mathcal{M}$ component provides model computation and web services using Apache Tomcat. The $\mathcal{D}$ component implements the geospatial database which resolves latitude and longitude coordinates to assist in providing climate, soil, and management data for RUSLE2 model runs. Postgresql with PostGIS extensions were used to support geospatial functionality [17-18]. The file server $\mathcal{F}$ component provides static XML files to RUSLE2 to parameterize model runs. NGINX [19], a lightweight high performance web server hosted over 57,000 static XML files on average ~5KB each. The logging $\mathcal{L}$ component provided historical tracking of modeling activity. The Codebeamer tracking facility which provides an extensive customizable GUI and reporting facility was used to log model activity [20]. A simple JAX-RS RESTful JSON-based web service decoupled logging functions from RUSLE2 by providing a logging queue to prevent delays from interfering with model execution. Codebeamer was hosted by the Apache Tomcat web application server and used the Derby file-based relational database. Codebeamer, a 32-bit web application, required the Linux 32-bit compatibility libraries (ia32-libs) to run on 64-bit VMs. A physical server running the HAProxy load balancer provided a public proxy server which redirected modeling requests to the VM hosting the modeling engine. HAProxy is a dynamically configurable very fast load balancer which supports proxying both TCP and HTTP socket-based network traffic [21].

C. Service Configurations

RUSLE2's infrastructure components can be deployed 15 possible ways using from 1-4 VMs. Table II shows the tested service configurations labeled as SC1-SC15. To create the compositions a composite VM image with all (4) application components was used. An automated test script enabled/disabled application components as needed to achieve the configurations. This method allowed automatic configuration of all component deployments using a single VM image. Tradeoffs for this approach were that the composite image had to be large enough to contain all components, and that VMs had installed but non-running components.

VMs were deployed in physical isolation with each physical machine hosting only one VM. This simplified the experimental setup and isolated VMs supported by homogeneous hardware which provided a controlled environment to support experimentation without interference from external non-application VMs. For RQ-2 physical machines hosted multiple VMs to test the effect of VM service isolation.

For the deployment configurations tests, all VMs were initially configured to have 8 virtual CPUs, 4 GB memory and 10GB of disk space regardless of the number of components hosted by each VM.

Table III describes component deployment configurations tested for RQ-2 service isolation testing. VMs are indicated with [ ]'s. These tests measured application performance variation resulting from using or not using separate VMs to isolate application components. The three top performing component deployments identified in investigation of RQ-1 were used.

TABLE II. TESTED COMPONENT DEPLOYMENTS

	VM 1	*VM 2*	*VM 3*	*VM 4*
SC1	$\mathcal{MDFL}$			
SC2	$\mathcal{MDF}$	$\mathcal{L}$		
SC3	$\mathcal{MD}$	$\mathcal{FL}$		
SC4	$\mathcal{MD}$	$\mathcal{F}$	$\mathcal{L}$	
SC5	$\mathcal{M}$	$\mathcal{DFL}$		
SC6	$\mathcal{M}$	$\mathcal{DF}$	$\mathcal{L}$	
SC7	$\mathcal{M}$	$\mathcal{D}$	$\mathcal{F}$	$\mathcal{L}$
SC8	$\mathcal{M}$	$\mathcal{D}$	$\mathcal{FL}$	
SC9	$\mathcal{M}$	$\mathcal{DL}$	$\mathcal{F}$	
SC10	$\mathcal{MF}$	$\mathcal{DL}$		
SC11	$\mathcal{MF}$	$\mathcal{D}$	$\mathcal{L}$	
SC12	$\mathcal{ML}$	$\mathcal{DF}$		
SC13	$\mathcal{ML}$	$\mathcal{D}$	$\mathcal{F}$	
SC14	$\mathcal{MDL}$	$\mathcal{F}$		
SC15	$\mathcal{MLF}$	$\mathcal{D}$		

TABLE III. SERVICE ISOLATION TESTS

NC	NODE 1	NODE 2	NODE 3
SC11-SI	[$\mathcal{M}$] [$\mathcal{F}$]	[$\mathcal{D}$]	[$\mathcal{L}$]
SC11	[$\mathcal{M}$ $\mathcal{F}$]	[$\mathcal{D}$]	[$\mathcal{L}$]
SC2-SI	[$\mathcal{M}$] [$\mathcal{D}$] [$\mathcal{F}$]	[$\mathcal{L}$]	
SC2	[$\mathcal{M}$ $\mathcal{D}$ $\mathcal{F}$]	[$\mathcal{L}$]	
SC6	[$\mathcal{M}$]	[$\mathcal{D}$] [$\mathcal{F}$]	[$\mathcal{L}$]
SC6-SI	[$\mathcal{M}$]	[$\mathcal{D}$ $\mathcal{F}$]	[$\mathcal{L}$]

D. Testing Setup

A Eucalyptus 2.0 IaaS private cloud [22] was built and hosted by Colorado State University consisting of 9 SUN X6270 blade servers sharing a private 1 Giga-bit VLAN. Servers had dual Intel Xeon X5560-quad core 2.8 GHz CPUs, 24GB ram, and two 15000rpm HDDs of 145GB and 465GB capacity respectively. The host operating system was CentOS 5.6 Linux (2.6.18-274) 64-bit server for the Xen hypervisor [23] and Ubuntu Linux 10.10 64-bit server (2.6.35-22) for the KVM hypervisor. VM guests ran Ubuntu Linux (2.6.31-22) 64-bit server 9.10. Eight servers were configured as Eucalyptus node-controllers, and one server was configured as the Eucalyptus cloud-controller, cluster-controller, walrus server, and storage-controller. Eucalyptus managed mode networking using a managed Ethernet switch was used to isolate VMs onto their own private VLANs.

Available versions of the Xen and KVM hypervisors were tested to establish which provided the fastest performance using SC1 from Table II. Ten trials of an identical 100-model run ensemble test were executed using the "m-bound" variant of the RUSLE2 application and average ensemble execution times are shown in Table IV. Xen 3.4.3 hvm represents the Xen hypervisor running in full virtualization mode using CPU virtualization extensions similar to the KVM hypervisor. Xen 3.4.3 using paravirtualization was shown to provide the best performance and was used for the majority of experimental tests. Our application-based benchmarks of XEN and KVM reflect similar results from previous investigations [5-6].

TABLE IV. HYPERVISOR PERFORMANCE

Hypervisor	Avg. Time (sec)	Performance
Physical server	15.65	100%
Xen 3.1	25.39	162.24%
Xen 3.4.3	23.35	149.20%
Xen 4.0.1	26.2	167.41%
Xen 4.1.1	27.04	172.78%
Xen 3.4.3 hvm	32.1	205.11%
KVM disk virtio	31.86	203.58%
KVM no virtio	32.39	206.96%
KVM net virtio	35.36	225.94%

The Linux virtual memory drop_caches function was used to clear all caches, dentries and inodes before each ensemble test to negate training effects from repeating identical ensemble tests. This cache-flushing technique was verified by observing CPU, file I/O, and network I/O utilization for the automated tests with and without cache clearing. When caches were not cleared, total disk sector reads decreased after the system was initially exposed to the same ensemble test. When caches were force-cleared for each ensemble run, the system reread data. As the test harness was exercised we observed that Codebeamer's Derby database grew large resulting in performance degradations. To eliminate decreased performance from log file and database growth our test script deleted log files and removed and reinstalled Codebeamer after each ensemble run. These steps prevented out of disk space errors and allowed uninterrupted testing without intervention.

VM resource utilization statistics were captured using a profiling script to capture CPU time, disk sector reads and writes (disk sector=512 bytes), and network bytes sent/received. To determine resource utilization of the component deployments (Table II), resource utilization statistics were totaled from all VMs hosting the application.

IV. EXPERIMENTAL RESULTS

Table V summarizes tests in this study totaling more than 3,720 ensemble runs consisting of over 372,000 individual model runs. Subsections A, B, and C report results for our investigation of RQ-1. Characteristics of the resource utilization of the component deployments are reported in subsection A followed by performance results of the deployments in subsection B. Subsection C reports performance when VM memory was increased from 4GB to 10GB and Subsection D describes results from an experiment which measured overhead resulting from service isolation (RQ-2), the use of separate VMs to isolate application components. Subsection E concludes by presenting results of using a multiple linear regression model with resource utilization statistics as independent variables to predict performance for component deployments (RQ-3).

TABLE V. SUMMARY OF TESTS

Model	Trials	Ensembles /Trial	Service Comps.	Model Runs	Ens. Runs
m-bound 4GB same	2	20	15	60k	600
d-bound 4GB same	2	20	15	60k	600
m-bound 4GB diff	1	20	15	30k	300
d-bound 4GB diff	1	20	15	30k	300
m-bound 10GB same	1	20	15	30k	300
d-bound 10GB same	1	20	15	30k	300
m-bound 10GB diff (kvm)	1	20	15	30k	300
d-bound 10GB diff (kvm)	1	20	15	30k	300
m-bound 4GB diff	3	20	6	36k	360
m-bound 4GB same	3	20	6	36k	360
Totals				**372,000**	**3,720**

A. Component Deployment Resource Utilization

Resource utilization statistics were captured for all component deployments to investigate how resource use varied. To validate component deployments exhibited consistent resource utilization behavior, linear regression was used to compare two test runs consisting of 20 different 100-model run ensembles using the "m-bound" model with 4GB VMs. Comparing resource utilization data, CPU time had the poorest correlation ($R^2=0.358316$, $df=13$, $p=.018$), followed by disk sector reads ($R^2=0.432673$, $df=13$, $p=.00769$), and disk sector writes ($R^2=0.773122$, $df=13$, $p=.000016$). Network bytes received and network bytes sent correlated very strongly between identical test runs at ($R^2=0.999808$, $df=13$, $p=1.52 \cdot 10^{-25}$) and ($R^2=0.999797$, $df=13$, $p=2.14 \cdot 10^{-25}$). Network utilization appeared similar for both model types as they communicated the same information. For the "d-bound" model $\mathcal{D}$ performed many more queries but this additional computation was independent of the other components $\mathcal{M}$, $\mathcal{F}$, and $\mathcal{L}$.

To determine why CPU time had a weak correlation between tests, CPU utilization data for the SC7 deployment

was studied. SC7 isolates each component onto a separate VM enabling collection of resource utilization statistics per component. CPU time was less consistent because of the $\mathcal{L}$ Codebeamer logging component's behavior. $\mathcal{L}$ spent about 11 seconds of total CPU time per ensemble test, but CPU utilization for $\mathcal{L}$ was inconsistent between ensemble tests ($R^2=0.06218$, $df=13$, $p=.289$). Comparing CPU utilization for the $\mathcal{M}$, $\mathcal{F}$, and $\mathcal{D}$ components, a strong correlation was observed between ensemble tests ($R^2=0.885884$, $df=13$, $p=6.43 \cdot 10^{-10}$). By ignoring the $\mathcal{L}$ component's CPU utilization behavior, component deployments had statistically consistent resource utilization behavior.

Application performance and resource utilization varied based on the component deployment configuration. Comparing resource utilization among deployments for the "m-bound" model, network bytes sent/received varied by ~144%, disk sector writes by ~22%, disk sector reads by ~15% and CPU time by ~6.5% as shown in table VI. Comparing the fastest and slowest deployments the performance variation was ~3.2 seconds nearly 14% of the average ensemble execution time for all deployments. Resource utilization differences among deployments of the "d-bound" model was greater than "m-bound" with ~820% for disk sector reads, ~145% for network bytes sent/received, 111% for disk sector writes but only ~5.5% for CPU time as shown in Table VII. "D-bound" model performance comparing the fastest versus slowest deployments varied by 25.7% (>34 seconds).

TABLE VI. "M-BOUND" DEPLOYMENT VARIATION

Parameter	M-bound	Deployment Difference
Avg. ensemble (sec)	23.4	13.7% (3.2 sec)
Avg. CPU time (sec)	11.7	6.5%
Avg. disk sector reads	57,675	14.8%
Avg. disk sector writes	286,297	21.8%
Avg. network bytes rec'd	9,019,414	144.9%
Avg. network bytes sent	9,037,774	143.7%

TABLE VII. "D-BOUND" DEPOLYMENT VARIATION

Parameter	D-bound	Deployment Difference
Avg. ensemble (sec)	133.4	25.7% (34.3 sec)
Avg. CPU time (sec)	27.8	5.5%
Avg. disk sector reads	2,836,144	819.6%
Avg. disk sector writes	246,364	111.1%
Avg. network bytes rec'd	9,269,763	145.0%
Avg. network bytes sent	9,280,216	143.9%

Comparing both applications a ~138% increase in CPU time was observed for the "d-bound" model vs. the "m-bound". Network utilization increased ~3% and disk sector reads where the $\mathcal{M}$ and $\mathcal{D}$ components were co-located increased 24,000%, but decreased 87% for deployments where $\mathcal{M}$ and $\mathcal{D}$ were not co-located. Network utilization likely increased for the "d-bound" model due to the longer duration of ensemble runs. More network traffic occurred as a result of having network connections open longer.

Figure 1 shows resource utilization variation for component deployments of the "m-bound" model. Resource utilization statistics were totaled from all VMs comprising individual component deployments. The graph shows the absolute value of the deviation from average resource utilization for the component deployments (SC1 – SC15). The graph does not express positive/negative deviation from average but the magnitude of deviation. Larger boxes indicate a greater deviation from average resource utilization and smaller boxes indicate performance close to the average.

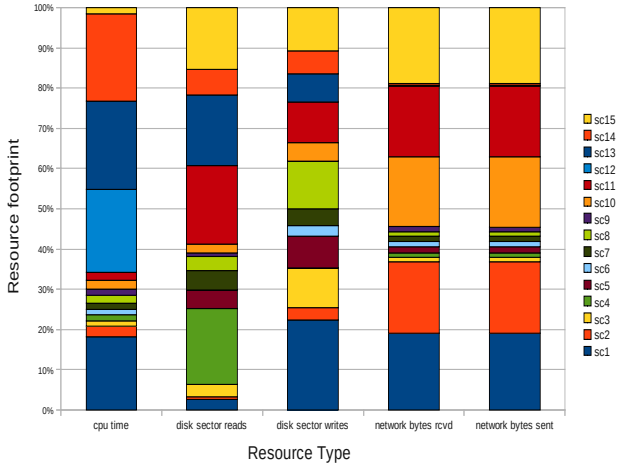

Figure 1. Resource Utilization Variation of Component Deployments

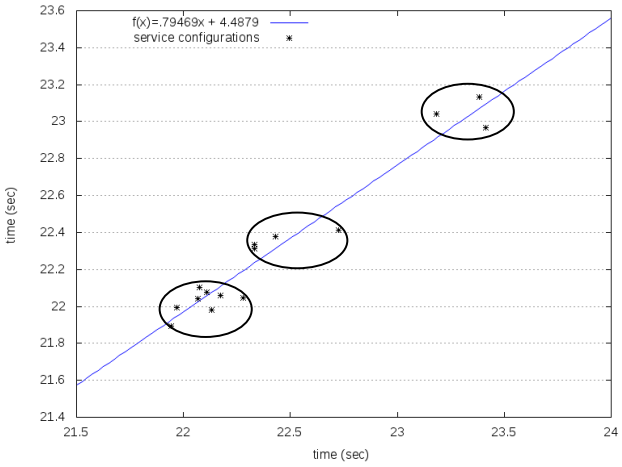

Figure 2. 4GB VM "M-bound" Component Deployment Performance Regression Plot

B. *Component Deployment Performance*

To verify that component deployments performed consistently two identical tests consisting of 20 runs of the same 100-model run ensemble test were performed using the 15 component deployments. The regression plot in Figure 2 compares the behavior of the two repeated test sets.

Linear regression confirms the consistency of component deployment performance for repeated test sets (R^2=0.949674, df=13, $p=8.09 \cdot 10^{-10}$). The three ellipses in the graph identify three different performance groups from left to right: fast, medium and slow. Performance consistency of "d-bound" tests was verified using the same linear regression technique. The consistency was not as strong due to higher variance of "d-bound" model execution times but was statistically significant (R^2=0.81501, df=13, $p=4.08 \cdot 10^{-6}$).

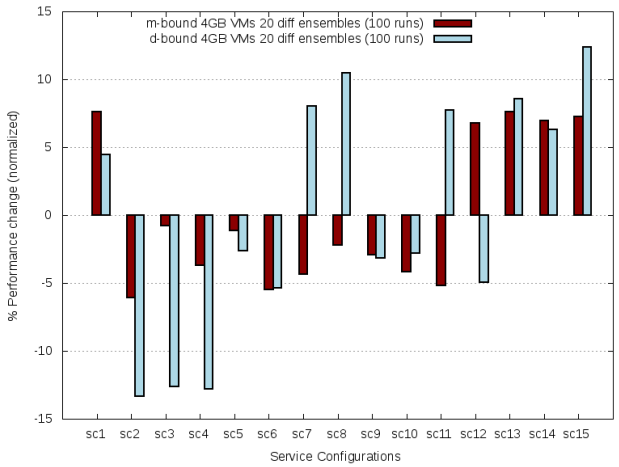

Figure 3. Performance Comparison – Randomized Ensembles

TABLE VIII. PERFORMANCE DIFFERENCES – RANDOMIZED ENSEMBLES

composition	m-bound	rank	d-bound	rank
SC1	7.59%	14	4.46%	9
SC2	-6.06%	1	-13.35%	1
SC3	-0.80%	10	-12.64%	3
SC4	-3.74%	6	-12.81%	2
SC5	-1.13%	9	-2.64%	8
SC6	-5.50%	2	-5.40%	4
SC7	-4.38%	4	7.98%	12
SC8	-2.21%	8	10.44%	14
SC9	-2.92%	7	-3.16%	6
SC10	-4.21%	5	-2.84%	7
SC11	-5.20%	3	7.72%	11
SC12	6.74%	11	-4.98%	5
SC13	7.63%	15	8.57%	13
SC14	6.97%	12	6.28%	10
SC15	7.22%	13	12.36%	15

To simulate a production modeling web service 20 randomized 100-model run ensembles were generated (2,000 unique requests) and used to benchmark each of the 15 component deployments. Figure 3 shows the performance comparison of the "m-bound" vs. "d-bound" model using 20 different ensemble tests. Performance differences from average and overall rankings are shown in table VIII.

Service compositions for the "m-bound" application with random ensembles can be grouped into three categories of performance fast {SC2, SC4, SC6, SC7, SC9, SC10, SC11}, medium {SC3, SC5, SC8}, and slow {SC1, SC12, SC13, SC14, SC15}. Compositions with $\mathcal{M}$ and $\mathcal{L}$ components co-located performed slower in all cases averaging 7.25% slower, about 1.7 seconds. When compositions had $\mathcal{M}$ and $\mathcal{L}$ co-located CPU time increased 14.6%, disk sector writes 18.4%, and network data sent/received about 3% versus compositions where $\mathcal{M}$ and $\mathcal{L}$ were separate.

Our results demonstrate variation in both resource utilization and application performance resulting from how components were deployed. Top performing deployments for both model variants utilized either two or three VMs. Service isolation (SC7) did not equate to best performance for either model. SC7 was ranked 4th fastest for the "m-bound" model and 12th for the "d-bound" model. Prior to testing the authors posited that application service isolation (SC5), total service isolation (SC7), and geospatial database isolation (SC15) could potentially be the fastest deployments. None of these deployments were top performers demonstrating that testing may be necessary to determine the best placements when intuition is insufficient.

C. Increasing VM Memory

In [24] the RUSLE2 model was used to investigate multi-tier application scaling with components deployed using isolated VMs. VMs hosting the $\mathcal{M}$, $\mathcal{F}$, and $\mathcal{L}$ components were allocated 2GB memory, and the $\mathcal{D}$ component VM was allocated 4GB. To avoid performance degradation due to memory contention, VM memory was increased to 10GB equal to the total allocation provided for previous testing in isolation [24]. Intuition suggests increasing VM memory should result in a performance improvement or no negative effect on performance. 20 runs of an identical 100-model run ensemble were repeated for all 15 component deployments using 10GB VMs. Figure 4 shows performance changes resulting from increasing VM memory allocation from 4GB to 10GB for both the "m-bound" and "d-bound" applications.

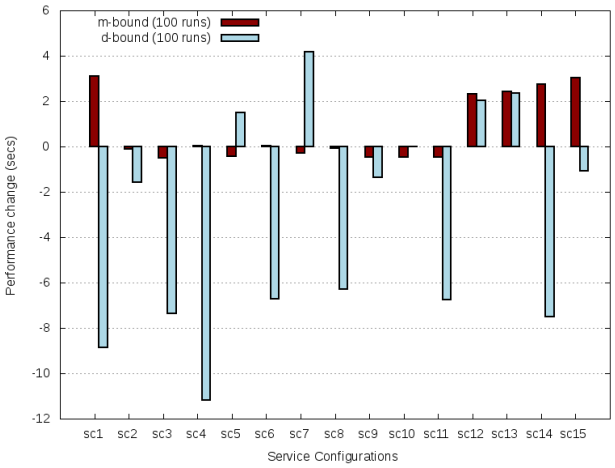

Figure 4. 10GB VM Performance Change (seconds)

For the "m-bound" application using 10GB VMs resulted in average ensemble performance .727 seconds (-3.24%) **slower** than with 4GB VMs. The *SC11* composition

performed best at 6.7% faster than the average 10GB "m-bound" ensemble test, or about .5 seconds faster than when using VMs with 4GB memory. *SC1*, total service combination, performed the slowest at 8.9% slower than average, 3.1 seconds longer than with 4GB VMs. Only "m-bound" application component deployments which combined $\mathcal{M}$ and $\mathcal{L}$ components on the same VM experienced performance degradation. Both $\mathcal{M}$ and $\mathcal{L}$ used the Apache Tomcat web application server, but $\mathcal{L}$ used a 32-bit version for hosting Codebeamer which required ia32 Linux 32-bit compatibility libraries to run on a 64-bit VM. These observed performance degradations may have resulted from virtualization of the ia32 library as 32-bit Linux only natively addresses up to 4GB ram.

The "d-bound" application using 10GB VMs performed on average 3.24 seconds (2.46%) faster than when using 4GB VMs. More available memory improved database query performance. *SC4* performed best at 12.5% faster or about 11.2 seconds faster. *SC7*, total service isolation, performed the slowest at 12.6% slower than average equaling about 4.2 seconds longer than tests with 4GB VMs.

To confirm this result was not unique to the XEN hypervisor and the use of an identical 100-model run ensemble 20 times, we repeated this test using the KVM hypervisor with 20 different ensembles. Results were similar. The "m-bound" model's 15 component deployments performed on average 342 ms slower (-1.13%) with 10GB VMs and the "d-bound" model performed 3.24 seconds (2.46%) faster on average. Our results demonstrate that increasing VM memory allocation can result in non-intuitive changes to application performance with some deployments experiencing performance changes exceeding +/-10%.

D. Service Isolation Performance Tests

To investigate overhead resulting from the use of separate VMs to host application components the three highest performing component deployments for the "m-bound" model were studied. Components were deployed using the SC2, SC6, and SC11 configurations with and without using separate VMs to host individual components. 60 runs of the same 100-model run ensemble and also 3 runs using 20 different 100-model run ensembles for each composition were completed. The percentage performance change resulting from service isolation is shown in Figure 5.

In all configurations but one, service isolation resulted in overhead and performance degradation compared with deployments which combined multiple components on VMs. The average overhead from service isolation was ~1%. For tests using different ensembles the normalized performance degradation observed for service isolation deployments was 1.2%, .3%, and 2.4%. For same ensemble tests performance degradation was 1.1%, -.6%, and 1.4%. The same result was reproduced using KVM as the hypervisor with an average observed performance degradation of 2.4% using separate VMs. Although the measured performance overhead was not large, it is important to consider that using additional VMs incurs higher hosting costs without performance benefits. The isolated nature of our test configuration using isolated physical hardware running no other applications allows us to be certain that observed overhead was entirely from VM-level service isolation.

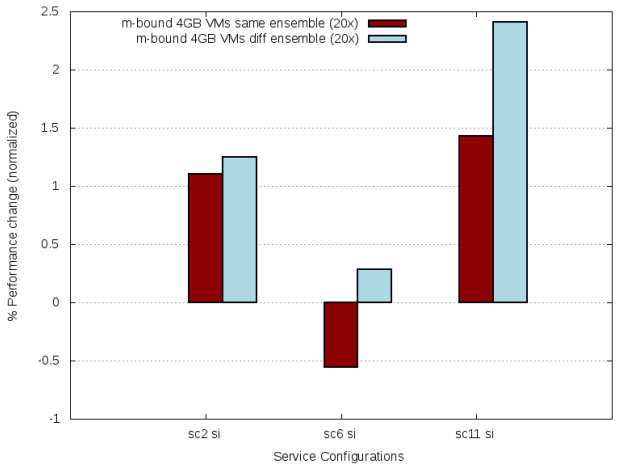

Figure 5. Performance Overhead From Service Isolation

E. Predictive Model

Resource utilization data was collected for CPU time, disk sector reads/writes, and network bytes sent/received as described in subsection A. We observed that the resource utilization varied for each of the deployments tested. To test whether resource utilization data is useful for predicting performance of component deployments in support of RQ-3, multiple linear regression (MLR) was used to build performance models.

TABLE IX. RESOURCE UTILIZATION VARIABLE PREDICTIVE POWER

Parameter	R^2	RMSD
CPU time	.7171	887.64
# Disk sector reads	.3714	1323.25
# Disk sector writes	.1441	1544.05
Network bytes recv'd.	.0074	1662.76
Network bytes sent	.0075	1662.68
Number of VMs	.0444	1631.44

Multiple linear regression (MLR) is a statistical technique used to model the linear relationship between a dependent variable and one or more independent variables [25]. The dependent variable for our MLR models is ensemble execution time and the independent variables were VM resource utilization statistics including: CPU time, disk sector reads/writes, network bytes sent/received, and the number of virtual machines. Predictors in MLR models may be interrelated making evaluation of their individual importance challenging. The "R-squared" value, also known as the coefficient of determination, explains the explanatory power of the entire model and its independent variables as the proportion of variance accounted for. R-squared values were calculated separately for each independent variable

using single linear regression models to determine their predictive value when used independently. Root mean squared deviation (RMSD) was calculated for each model. The RMSD expresses differences between the predicted and observed values and serves to provide a measure of model accuracy. Ideally 95% of predictions should be less than +/- 2 RMSD's from the actual value.

A MLR model was built using resource utilization variables from the "m-bound" model using Xen 4GB VMs with 20 different ensemble tests. All of our resource utilization variables together produced a model which accounted for 84% of the variance with a RMSD of only ~676 ms (R^2=*.8416, RMSD=664.17 ms*). Table IX shows individual R^2 values for the resource utilization statistics used in a simple linear regression model with ensemble execution time to determine how much variance each explained. Additionally the average error (RMSD) is shown. The most predictive parameters were CPU time which positively correlated with ensemble time and explained over 70% of the variance (R^2=*.7171*) and disk sector reads (R^2=*.3714*) with a negative correlation. Disk sector writes had a positive correlation with ensemble performance (R^2=*.1441*), while the number of VMs negatively correlated with ensemble time but predicted little variance (R^2=*.0444*). Network bytes received/sent both predicted less than 1% of the variance.

We then used our MLR performance model to predict performance of component deployments. Resource utilization data used to generate the model was fed back in to generate ensemble time predictions. Average predicted ensemble execution times were calculated for each component deployment (SC1-SC15) and predicted ranks were assigned. Predicted vs. actual performance ranks are shown in table X. The mean absolute error (MAE) was 462 ms, and estimated ranks were on average +/-1.33 units from the actual ranks. Eleven predicted ranks for component compositions were off by 1 unit or less from their actual rank, with six exact predictions for SC2, SC10, SC11, SC13, SC14, and SC15. The top three performing deployments were predicted correctly in order. A second dataset was collected by rerunning the 20 ensembles using the 15 component deployments while collecting resource utilization data. This data was fed into our MLR performance model and we observed a MAE of only 324ms. The average rank error was +/- 2 units. Seven predicted ranks were off by 1 unit or less from their actual rank, with three exact predictions. The fastest deployment was predicted accurately while the second and third fastest were predicted as 6th, and 8th.

Building models to predict component deployment performance requires careful consideration of resource utilization variables. Using multiple linear regression was helpful to identify which independent variables had the greatest impact on deployment performance. Additional work which explores the use of resource utilization statistics to predict application deployment performance appears in [37]. This work investigates the use of additional independent variables including CPU statistics, kernel scheduler statistics, and guest/host load averages using both MLR and neural networks. Future work should further investigate the use of neural networks, genetic algorithms, and/or support vector machines to assess their potential to improve performance predictions where available training data is non-linear extending related research [11, 27-31].

TABLE X. DEPLOYMENT PERFORMANCE RANK PREDICTIONS

Composition	Predicted Rank	Actual Rank	Rank Error
SC1	12	15	-3
SC2	2	2	0
SC3	7	8	-1
SC4	6	9	-3
SC5	10	4	6
SC6	9	10	-1
SC7	4	5	-1
SC8	8	7	1
SC9	5	6	-1
SC10	3	3	0
SC11	1	1	0
SC12	15	12	3
SC13	14	14	0
SC14	13	13	0
SC15	11	11	0

V. CONCLUSIONS

(RQ-1) This research investigated the scope of performance implications which occur based on how components of multi-tier applications are deployed across VMs on a private IaaS cloud. All possible deployments were tested for two variants of the RUSLE2 soil erosion model, a 4-component application. Up to a 14% and 25.7% performance variation was observed for the "m-bound" and "d-bound" RUSLE2 models respectively. Significant resource utilization (CPU, disk, network) variation was observed based on how application components were deployed across VMs. Some configurations reduced overall resource consumption while others significantly increased it leading to performance degradations. Increasing VM memory allocation did not guarantee performance improvements and intuition was insufficient to determine the best performing deployments. Ad hoc worst case scenario component placements significantly degraded application performance demonstrating consequences for ignoring component composition. Providing VM/application level resource load balancing and using compact application deployments holds promise for improving application performance while lowering hosting costs (# of VMs) for applications and should be investigated further for supporting application deployment to IaaS clouds.

(RQ-2) Service isolation, the practice of using separate VMs to host individual application components, is beneficial where scalability of individual components is a priority, but results in performance overhead. We observed up to 2.4% average performance overhead from using multiple VMs on the same physical host as isolation containers.

(RQ-3) Resource utilization statistics are useful for building performance models to predict performance of component deployments. Using six resource utilization variables our multiple linear regression model accounted for 84% of the variance in predicting performance of component deployments and accurately predicted the top performing component deployments. VM/resource based performance models should be investigated further as they hold promise to help guide intelligent application deployment and resource load balancing for IaaS clouds.

REFERENCES

[1] M. Rehman, M. Sakr, Initial findings for provisioning variation in cloud computing, Proc. of the IEEE 2nd Intl. Conf. on Cloud Computing Technology and Science (CloudCom '10), Indianapolis, IN, USA, Nov 30 – Dec 3, 2010, pp. 473-479.

[2] J. Schad, J. Dittrich, J. Quiane-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proc. of the VLDB Endowment, vol. 3, no. 1-2, Sept. 2010, pp. 460-471.

[3] M. Zaharia et al., Improving MapReduce performance in heterogeneous environments, Proc. 8th USENIX Conf. Operating systems design and implementation (OSDI '08), San Diego, CA, USA, Dec 8-10, 2008, pp. 29-42.

[4] M. Vouk, Cloud Computing – Issues, research, and implementations, Proc. 30th Intl. Conf. Information Technology Interfaces (ITI 2008), Cavtat, Croatia, June 23-26, 2008, pp. 31-40.

[5] F. Camargos, G. Girard, B. Ligneris, Virtualization of Linux servers, Proc. 2008 Linux Symposium, Ottawa, Ontario, Canada, July 23-26, 2008, pp. 73-76.

[6] D. Armstrong, K. Djemame, Performance issues in clouds: An evaluation of virtual image propagation and I/O paravirtualization, The Computer Journal, June 2011, vol. 54, iss. 6, pp. 836-849.

[7] D. Jayasinghe et al., Variations in performance and scalability when migrating n-tier applications to different clouds, Proc. 4th IEEE Conf on Cloud Computing(Cloud '11), Washington D.C., USA, July 2011, pp.73-80.

[8] J. Matthews, et al., Quantifying the performance isolation properties of virtualization systems, Proc. ACM Workshop on Experimental Comp. Science (ExpCS '07), New York, NY, USA, 2007, Article 6.

[9] G. Somani, S. Chaudhary, Application performance isolation in virtualization, Proc. 2nd IEEE Intl. Conf on Cloud Computing (Cloud 2009), Bangalore, Indian, Sept 2009, pp. 41-48.

[10] H. Raj et al., Resource Management for isolation enhanced cloud services, Proc. ACM Cloud Comp. Security Wrkshp (CCSW '09), Chicago, IL, USA, Nov 2009, pp. 77-84.

[11] O. Niehörster, A. Krieger, J. Simon, A. Brinkmann, Autonomic resource management with support vector machines, Proc. 12th IEEE/ACM Int. Conf. On Grid Computing (GRID 2011), Lyon, France, Sept 21-23, 2011, pp. 157-164.

[12] United States Department of Agriculture – Agricultural Research Service (USDA-ARS), Revised Universal Soil Loss Equation Version 2 (RUSLE2), http://www.ars.usda.gov/SP2UserFiles/Place/64080510/RUSLE/RUSLE2_Science_Doc.pdf

[13] L. Ahuja, J. Ascough II, and O. David, Developing natural resource modeling using the object modeling system: Feasibility and challenges," Advances in Geosciences, vol. 4, 2005, pp. 29-36.

[14] O. David et al., Rethinking modeling framework design: Object Modeling System 3.0, Proc. iEMSs 2010 Intl. Congress on Env. Modeling and Software, Ottawa, Canada, July 5-8, 2010, 8 p.

[15] WineHQ – Run Windows applications on Linux, BSD, Solaris, and Mac OS X, http://www.winehq.org/

[16] Apache Tomcat – Welcome, 2011, http://tomcat.apache.org/

[17] PostGIS, 2011, http://postgis.refractions.net/

[18] PostgreSQL: The world's most advanced open source database, http://www.postgresql.org/

[19] nginx news, 2011, http://nginx.org/

[20] Welcome to CodeBeamer, 2011, https://codebeamer.com/ cb/user/

[21] HAProxy – The Reliable, High Performance TCP/HTTP Load Balancer, http://haproxy.1wt.eu/

[22] D. Nurmi et al., The Eucalyptus open-source cloud-computing system, Proc. IEEE Intl. Symposium on Cluster Computing and the Grid (CCGRID 2009), Shanghai, China, May 18-21, 8p.

[23] P. Barham, et al., Xen and the art of virtualization, Proc. 19th ACM Symposium on Operating Systems Principles (SOSP '03), Bolton Landing, NY, USA, Oct 19-22, 2003, 14 p.

[24] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, K. Rojas, Migration of multi-tier applications to infrastructure-as-a-service clouds: An investigation using kernel-based virtual machines, Proc. 12th IEEE/ACM Intl. Conf. On Grid Computing (GRID 2011), Lyon, France, Sept 21-23, 2011, pp. 137-143.

[25] R.H. Myers, Classical and modern regression with applications, 2nd Edition, PWS-KENT Publishing Company, Boston, MA, 1994.

[26] S. Govindan et al., Quantifying effects of shared on-chip resource interference for consolidated virtual machines, Proc. 2nd ACM Symposium on Cloud Computing (SOCC 2011), Cascais, Portugal, Oct 26-28, 2011, 14p.

[27] G. Kousiouris, T. Cucinotta, T. Varvarigou, The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks, Jrnl. of Sys & Software, vol. 84, 2011, pp. 1270-1291.

[28] N. Bonvin, T. Papaioannou, K. Aberer, Autonomic SLA-driven provisioning for cloud applications, Proc. IEEE/ACM Int. Symposium on Cluster, Cloud, and Grid Computing (CCGRID 2011), Newport Beach, CA, USA, 2011, pp. 434-443.

[29] C. Xu, J. Rao, X. Bu, URL: A unified reinforcement learning approach for autonomic cloud management, Journal of Parallel and Distributed Computing, vol. 72, 2012, pp. 95-105.

[30] P. Lama, X. Zhou, Efficient server provisioning with control for end-to-end response time guarantee on multitier clusters, IEEE Transactions on Parallel and Distributed Systems, vol. 23, No. 1, Jan 2012, pp. 78-86.

[31] P. Lama, X. Zhou, Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee, Proc. 18th IEEE/ACM Int. Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2010), Miami Beach, FL, USA, August 17-19, 2010, pp. 151-160.

[32] B. Sharma et al., Modeling and synthesizing task placement constraints in Google compute clusters, Proc. 2nd ACM Symp.on Cloud Computing (SOCC11), Cascais, Portugal, Oct 2011, 14p.

[33] A. Gulati et al., Pesto: Online storage performance management in virtualized datacenters, Proc. 2nd ACM Symposium on Cloud Computing (SOCC 2011), Cascais, Portugal, Oct 26-28, 2011, 14p.

[34] H. Kang et al., Enhancement of Xen's scheduler for MapReduce workloads, Proc. 20th Intl. ACM Symposium on High-Performance Parallel and Distributed Computing (HPDC '11), San Jose, CA, June 8-11, 2011, pp. 251-262.

[35] T. Voith, K. Oberle, M. Stein, Quality of service provisioning for distributed data center inter-connectivity enabled by network virtualization, Future Gen. Comp. Systems, vol.28, 2012, pp.554-562.

[36] J. Varia, Architecting for the Cloud: Best Practices. Amazon Web Services White Paper, 2010, https://jineshvaria.s3.amazonaws.com/public/cloudbestpractices-jvaria.pdf

[37] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, K. Rojas, Performance Modeling to Support Multi-Tier Application Deployment to Infrastructure-as-a-Service Clouds, Proc. 5th IEEE/ACM Intl. Conf. On Utility and Cloud Computing (UCC 2012), Chicago, IL, USA, Nov 5-8, 2012, 8p.