Future Generation Computer Systems 29 (2013) 1254–1264

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Performance implications of multi-tier application deployments on Infrastructure-as-a-Service clouds: Towards performance modeling

W. Lloyd [a,b,*], S. Pallickara [a], O. David [a,b], J. Lyon [b], M. Arabi [b], K. Rojas [c]

[a] Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA

[b] Department of Civil and Environmental Engineering, Colorado State University, Fort Collins, CO 80523, USA

[c] USDA-NRCS, 2150 Center Ave., Bldg. A, Fort Collins, CO 80526, USA

ARTICLE INFO

Article history:
Received 15 February 2012
Received in revised form
11 October 2012
Accepted 7 December 2012
Available online 22 December 2012

Keywords:
Service isolation
Infrastructure-as-a-Service
Provisioning variation
Virtualization

ABSTRACT

Hosting a multi-tier application using an Infrastructure-as-a-Service (IaaS) cloud requires deploying components of the application stack across virtual machines (VMs) to provide the application's infrastructure while considering factors such as scalability, fault tolerance, performance and deployment costs (# of VMs). This paper presents results from an empirical study which investigates implications for application performance and resource requirements (CPU, disk and network) resulting from how multi-tier applications are deployed to IaaS clouds. We investigate the implications of: (1) component placement across VMs, (2) VM memory size, (3) VM hypervisor type (KVM vs. Xen), and (4) VM placement across physical hosts (provisioning variation). All possible deployment configurations for two multi-tier application variants are tested. One application variant was computationally bound by the application middleware, the other bound by geospatial queries. The best performing deployments required as few as 2 VMs, half the number required for VM-level service isolation, demonstrating potential cost savings when components can be consolidated. Resource utilization (CPU time, disk I/O, and network I/O) varied with component deployment location, VM memory allocation, and the hypervisor used (Xen or KVM) demonstrating how application deployment decisions impact required resources. Isolating application components using separate VMs produced performance overhead of ∼1%–2%. Provisioning variation of VMs across physical hosts produced overhead up to 3%. Relationships between resource utilization and performance were assessed using multiple linear regression to develop a model to predict application deployment performance. Our model explained over 84% of the variance and predicted application performance with mean absolute error of only ∼0.3 s with CPU time, disk sector reads, and disk sector writes serving as the most powerful predictors of application performance.

1. 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 for hosting using separate VMs. Application VMs are hosted by one or more physical machines (PMs). Service isolation provides application components with their own explicit sandboxes to operate in, each having independent operating system instances. *Hardware virtualization* enables service isolation using separate VMs to host each application component instance. Before virtualization, service isolation using PMs required significant server capacity. Service isolation has been suggested as a best practice for deploying multi-tier applications across VMs. A 2010 Amazon Web Services white paper suggests applications be deployed using service isolation. The white paper instructs the user to "bundle the logical construct of a component into an Amazon Machine Image so that it can be deployed (instantiated) more often" [1]. Service isolation, a 1:1 mapping of application component(s) to VM images is implied. 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. Conversely service isolation adds an abstraction layer above the physical hardware which introduces overhead potentially degrading performance. Deploying all application

* Correspondence to: USDA-ARS, ASRU, 2150 Center Ave., Bldg. D, Suite 200, Fort Collins, CO 80526, USA. Tel.: +1 970 492 7311; fax: +1 970 492 7310.
E-mail address: wlloyd@acm.org (W. Lloyd).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.12.007

components using separate VMs may increase network traffic, particularly when VMs are hosted by separate physical machines. Consolidating components together on a single VM guarantees they will not be physically separated when deployed potentially improving performance by reducing network traffic.

Provisioning variation results from the non-determinism of where application VMs are physically hosted in the cloud, often resulting in performance variability [2–4]. IaaS cloud providers often do not allow users to control where VMs are physically hosted causing this provisioning variation. Clouds consisting of PMs with heterogeneous hardware and hosting a variable number of VMs complicates benchmarking application performance [5].

Service Isolation provides isolation at the guest operating system level as VMs share physical hardware resources and compete for CPU, disk, and network bandwidth. Quantifying VM interference and investigation of approaches to multiplex physical host resources are active areas of research [6–12]. Current virtualization technology only guarantees VM memory isolation. VMs reserve a fixed quantity of memory for exclusive use which is not released until VM termination. Processor, network I/O, and disk I/O resources are shared through coordination 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 specific CPU cores to guarantee resource availability though CPU caches are still shared [9]. Developing mechanisms which guarantee fixed quantities of network and disk throughput for VM guests is an open area for research.

This research investigates performance of multi-tier application component deployments to IaaS clouds to better understand implications of component distribution across VMs, VM placement across physical hosts and VM configuration. We seek to better understand factors that impact performance moving towards building performance models to support intelligent methodologies that better load balance resources to improve application performance. We investigate hosting two variants of a non-stochastic multi-tier application with stable resource utilization characteristics. Resource utilization statistics that we capture from host VMs are then used to investigate performance implications relative to resource use and contention. The following research questions are investigated:

(RQ-1) How does resource utilization and application performance vary relative to how application components are deployed? How does provisioning variation, the placement of VMs across physical hosts, impact performance?
(RQ-2) Does increasing VM memory allocation change performance? Does the virtual machine hypervisor (Xen vs. KVM) affect performance?
(RQ-3) How much overhead results from VM service isolation?
(RQ-4) Can VM resource utilization data be used to build models to predict application performance of component deployments?

2. Related work

Rouk identified the challenge of finding ideal service compositions for creating virtual machine images to deploy applications in cloud environments in [13]. Schad et al. [3] demonstrated the unpredictability of Amazon EC2 VM performance caused by contention for physical machine resources and provisioning variation of VMs. Rehman et al. tested the effects of resource contention on Hadoop-based MapReduce performance by using IaaS-based cloud VMs to host worker nodes [2]. 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 multiple unique components present a more complex challenge for resource provisioning than studied by Rehman et al. Zaharia et al. observed that Hadoop's native scheduler caused severe performance degradation by ignoring resource contention among Hadoop nodes hosted by Amazon EC2 VMs [4]. They proposed the Longest Approximate Time to End (LATE) scheduling algorithm which better addresses performance variations of heterogeneous Amazon EC2 VMs. Their work did not consider hosting of heterogeneous components.

Camargos et al. investigated virtualization hypervisor performance for virtualizing Linux servers with several performance benchmarks for CPU, file and network I/O [14]. Xen, KVM, VirtualBox, and two container based virtualization approaches OpenVZ and Linux V-Server were tested. Different parts of the system were targeted using kernel compilation, file transfers, and file compression benchmarks. Armstrong and Djemame investigated performance of VM launch time using Nimbus and OpenNebula, two IaaS cloud infrastructure managers [15]. Additionally they benchmarked Xen and KVM paravirtual I/O performance. Jayasinghe et al. investigated performance of the RUBBoS *n*-tier e-commerce system deployed to three different IaaS clouds: Amazon EC2, Emulab, and Open Cirrus [16]. They tested horizontal scaling, changing the number of VMs for each component, and vertical scaling, varying the resource allocations of VMs. They did not investigate consolidating components on VMs but used separate VMs for full service isolation. Matthews et al. developed a VM isolation benchmark to quantify the isolation level of co-located VMs running several conflicting tasks [6]. They tested VMWare, Xen, and OpenVZ hypervisors to quantify isolation. Somani and Chaudhary benchmarked Xen VM performance with co-located VMs running CPU, disk, or network intensive tasks on a single physical host [7]. They benchmarked the Simple Earliest Deadline First (SEDF) I/O credit scheduler vs. the default Xen credit scheduler and investigated physical resource contention for running different co-located tasks, similar to resource contention of co-hosting different components 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 [8].

Niehörster et al. developed an autonomic system using support vector machines (SVM) to meet predetermined quality-of-service (QoS) goals. Service specific agents were used to provide horizontal and vertical scaling of virtualization resources hosted by an IaaS Eucalyptus cloud [17]. Their agents scaled # of VMs, memory, and virtual core allocations. Support vector machines determined if resource requirements were adequate for the QoS requirement. They tested their approach by dynamically scaling the number of modeling engines for GROMACS, a molecular dynamics simulation and also for an Apache web application service 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 improve task scheduling and distribution on compute clusters [18]. Similar to Sharma's models to improve task placement, RQ-4 investigates building performance models which could be used to guide component deployments for multi-tier applications.

Previous studies have investigated a variety of related issues but none have investigated the relationship between application performance and resource utilization (CPU, disk, network) resulting from how components of multi-tier applications are deployed across VMs (isolation vs. consolidation).

3. Paper contributions

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 and VM placement, resource utilization and application performance are investigated. Additionally we investigate performance and resource utilization changes resulting from: (1) the use of different hypervisors (Xen vs. KVM), and (2) increasing VM memory allocation. Overhead from using separate VMs to host application components is also measured. Relationships between resource utilization and performance are used to develop a multiple linear regression model to predict application performance. Our approach for collecting application resource utilization data to construct performance model(s) can be generalized for any multi-tier application.

4. Experimental design

To support investigation of our research questions we studied the migration of a widely used Windows desktop environmental modeling application deployed to operate as a multi-tier web services application. Section 4.1 describes the application and our test harness. Section 4.2 describes components of the multi-tier application. Section 4.3 details the configuration of tested component deployments. Section 4.4 concludes by describing our private IaaS cloud and hardware configuration used for this investigation.

4.1. Test application

For our investigation we utilized two variants of the RUSLE2 (Revised Universal Soil Loss Equation — Version 2) soil erosion model [19]. 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 3000 field offices across the United States. RUSLE2 was originally developed as a Windows-based Microsoft Visual C++ desktop application and has been extended to provide soil erosion modeling as a REST-based web-service hosted by Apache Tomcat [20]. JSON was the transport protocol for data objects. To facilitate functioning as a web service a command line console was added. RUSLE2 consists of four tiers including an application server, a geospatial relational database, a file server, and a logging server. RUSLE2 is a good multi-component application for our investigation because with four components and 15 possible deployments 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. The Object Modeling System 3.0 (OMS3) framework [21,22] using WINE [23] provided middleware to facilitate interacting with RUSLE2's command line console. OMS3, developed by the USDA-ARS in cooperation with Colorado State University, 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. Results from individual runs of the ensemble are "reduced" into a single JSON response object. A test generation program created 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. Random selection of latitude and longitude coordinates led to variable geospatial query execution times because the polygons intersected with varied in complexity. To verify our test generation technique produced test sets with variable complexity we completed 2 runs of 20 randomly generated 100-model run ensemble tests run using the 15 RUSLE2 component deployments 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 ($R^2 = 0.914$, $df = 18$, $p = 5 \cdot 10^{-11}$).

Our investigation utilized two variants of RUSLE2 referred to as "*d*-bound" for the database bound variant and "*m*-bound" for the model bound variant, names based on the component dominating execution time. These application variants represent 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" 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 had a different resource utilization profile than the "*m*-bound" RUSLE2. On average the "*d*-bound" application required $\sim$2.45$\times$ more CPU time than the "*m*-bound" model.

4.2. Application services

Table 1 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 [24,25]. The file server $\mathcal{F}$ component provides static XML files to RUSLE2 to parameterize model runs. NGINX [26], a lightweight high performance web server hosted over 57,000 static XML files on average $\sim$5 kB 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 [27]. 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 proxy service to redirect modeling requests to the VM hosting the modeling engine. HAProxy is a dynamically configurable fast load balancer that supports proxying both TCP and HTTP socket-based network traffic [28].

4.3. Service configurations

RUSLE2's infrastructure components can be deployed 15 possible ways using 1–4 VMs. Table 2 shows the tested service configurations labeled as SC1–SC15. To create the deployments for

Table 1
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 305 MB.
$\mathcal{L}$	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.

Table 2
Tested component deployments.

	VM1	VM2	VM3	VM4
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 3
Service isolation tests.

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

testing, a composite VM image with all (4) application components installed 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. This approach required that the composite disk image was large enough to host all components, and that VMs had installed but non-running components.

For testing SC1–SC15, VMs were deployed with physical isolation. Each VM was hosted by its own exclusive physical host. This simplified the experimental setup and provided a controlled environment using homogeneous physical host machines to support experimentation without interference from external non-application VMs. For provisioning variation testing (RQ-1) and service isolation testing (RQ-3) physical machines hosted multiple VMs as needed. For all the tests VMs had 8 virtual CPUs, and 10 GB of disk space regardless of the number of components hosted. VMs were configured with either 4 GB or 10 GB memory.

Table 3 describes component deployments used to benchmark service isolation overhead (RQ-3). Separate VMs are delineated using brackets. These tests measured performance overhead resulting from the use of separate VMs to isolate application components. Service isolation overhead was measured for the three fastest component deployments: SC2, SC6, and SC11.

4.4. Testing setup

A Eucalyptus 2.0 IaaS private cloud [29] 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, 24 GB RAM, and two 15 000 rpm HDDs of 145 GB and 465 GB capacity respectively. The host operating system was CentOS 5.6 Linux (2.6.18-274) 64-bit server for the Xen hypervisor [30] 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.

Table 4
Hypervisor performance.

Hypervisor	Avg. time (s)	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

Available versions of the Xen and KVM hypervisors were tested to establish which provided the fastest performance using SC1 from Table 2. 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 4. 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 [14,15].

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

Table 5
"*M*-bound" deployment variation.

Parameter	*M*-bound	Deployment difference
Avg. ensemble (s)	23.4	13.7% (3.2 s)
Avg. CPU time (s)	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	9019,414	144.9%
Avg. network bytes sent	9037,774	143.7%

Table 6
"*D*-bound" deployment variation.

Parameter	*D*-bound	Deployment difference
Avg. ensemble (s)	133.4	25.7% (34.3 s)
Avg. CPU time (s)	27.8	5.5%
Avg. disk sector reads	2836,144	819.6%
Avg. disk sector writes	246,364	111.1%
Avg. network bytes rec'd	9269,763	145.0%
Avg. network bytes sent	9280,216	143.9%

(disk sector = 512 bytes), and network bytes sent/received. To determine resource utilization of component deployments from all VMs hosting the application were totaled.

5. Experimental results

To investigate our research questions we completed nearly 10,000 ensemble tests totaling ∼1,000,000 individual model runs. Tests were conducted using both the "*m*-bound" and "*d*-bound" RUSLE2 model variants. VMs were hosted using either the Xen or KVM hypervisor and were configured with either 4 GB or 10 GB memory, 8 virtual cores, and 10 GB disk space. 15 component placements across VMs were tested, and these VMs were provisioned using physical hosts 45 different ways. Test sets executed 20 ensembles of 100 model runs each to benchmark performance and resource utilization of various configurations. All ensembles had 100 randomly generated model runs. Some test sets repeated the same ensemble test 20 times, while others used a set of 20 different ensemble tests for a total of 2,000 randomly generated model runs per test set. Results for our investigation of RQ-1 are described in Sections 5.1–5.3. Resource utilization characteristics of the component deployments are described in Section 5.1 followed by performance results of the deployments in Section 5.2. Section 5.3 reports on performance effects from provisioning variation, the variability resulting from where application VMs are physically hosted. Section 5.4 describes how application performance changed when VM memory was increased from 4 GB to 10 GB, and Section 5.5 reports on the performance differences of the Xen and KVM hypervisors (RQ-2). Section 5.6 presents results from our experiment measuring service isolation overhead (RQ-3). Section 5.7 concludes by presenting our multiple linear regression based performance model which predicts performance of component deployments based on resource utilization statistics (RQ-4).

5.1. Component deployment resource utilization

Resource utilization statistics were captured for all component deployments to investigate how they varied across all possible configurations. To validate that component deployments exhibited consistent resource utilization behavior, linear regression was used to compare two separate sets of runs consisting of 20 different 100-model run ensembles using the "*m*-bound" model with 4 GB Xen VMs. The coefficient of determination R^2 was calculated to determine the proportion of variance accounted for when regressing together the two datasets. Higher values indicate similarity in the datasets. Comparing R^2 resource utilization for CPU time ($R^2 = 0.937904$, $df = 298$), disk sector reads ($R^2 = 0.96413$, $df = 298$), and network bytes received/sent ($R^2 = 0.99999$, $df = 298$) for repeated tests appeared very similar. Only disk sector writes ($R^2 = 0.273696$, $df = 298$) was inconsistent. Network utilization appeared similar for both the "*m*-bound" and "*d*-bound" model variants 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 $\mathcal{MFL}$.

Application performance and resource utilization varied based on the deployment configuration of application components.

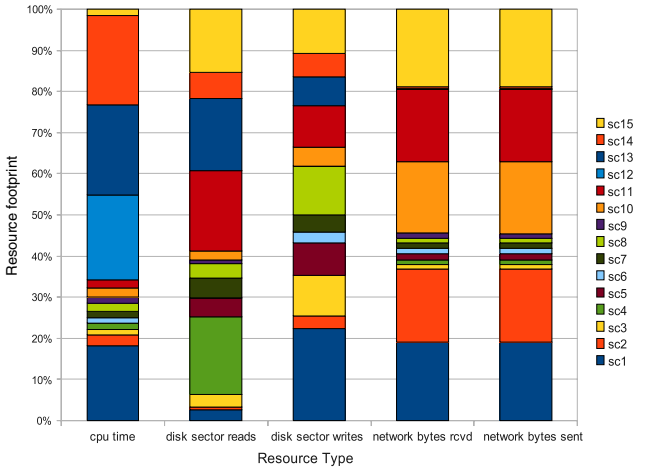

Fig. 1. Resource utilization variation of component deployments.

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 5. Comparing the fastest and slowest deployments the performance variation was ∼3.2 s, 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 6. "*D*-bound" model performance comparing the fastest versus slowest deployments varied by 25.7% (>34 s).

Comparing both applications hosted by 4 GB Xen VMs a ∼138% increase in CPU time was observed for the "*d*-bound" model. Network utilization increased ∼3% and disk sector reads for the "*d*-bound" model where the $\mathcal{M}$ and $\mathcal{D}$ components were co-located increased 24,000% vs. the "*m*-bound" model, but decreased 87% for deployments where $\mathcal{M}$ and $\mathcal{D}$ were not co-located. On average the Xen "*d*-bound" model ensemble execution times were 5.7× "*m*-bound", averaging 133.4 s versus 23.4. Network utilization likely increased for the "*d*-bound" model due to the longer duration of ensemble runs.

Fig. 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. The graph visually depicts the variance of resource utilization for our 15 component deployments.

5.2. Component deployment performance

To verify that component deployments performed consistently over time and to verify that we were not simply observing

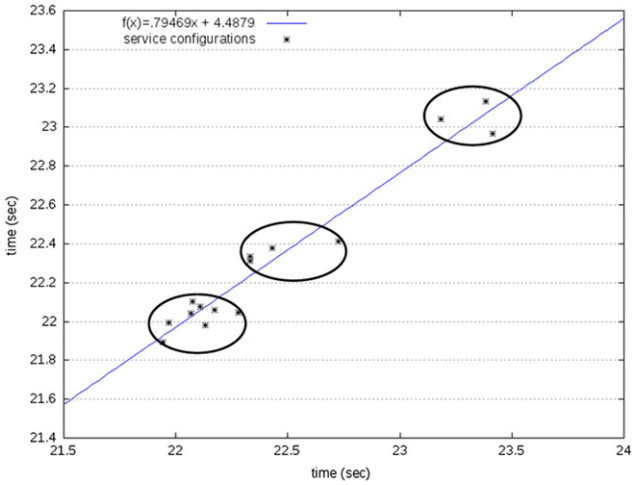

Fig. 2. 4 GB "*m*-bound" regression plot (Xen).

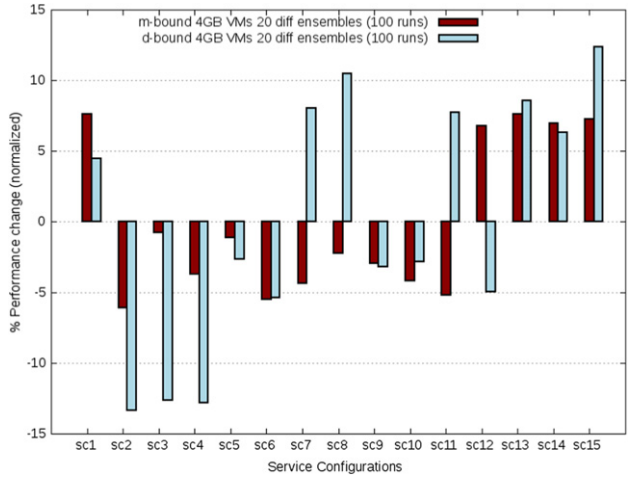

Fig. 3. Performance comparison — randomized ensembles (Xen).

random behavior, two test sets consisting of 20 runs of the same 100-model run ensemble test were performed using all component deployments. The regression plot in Fig. 2 compares the behavior of the two repeated test sets. Linear regression confirms the consistency of component deployment performance across subsequent 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 technique. The consistency was not as strong due to higher variance of "*d*-bound" model execution times but was statistically significant ($R^2 = 0.81501, df = 13, p = 4.08 \cdot 10^{-6}$).

To simulate a production modeling web service 20 randomized 100-model run ensembles were generated (2000 unique requests) and used to benchmark each of the 15 component deployments. Fig. 3 shows the performance comparison of the "*m*-bound" vs. "*d*-bound" model using the 20 different ensemble tests. Performance differences from average and overall rankings are shown in Table 7.

We observed performance variation of nearly ∼14% *for the "m-bound" model and* ∼26% *for the "d-bound" model comparing best-case vs. worse-case deployments.* 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 s.

Table 7
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

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.

Service isolation (SC7) did not provide the best performance for either model. SC7 was ranked 4th fastest for the "*m*-bound" model and 12th for the "*d*-bound" model. The top three performing deployments for both model variants required only two or three VMs. Prior to testing the authors posited that isolating the application server (SC5), total service isolation (SC7), and isolating the geospatial database isolation (SC15) could be the fastest deployments. None of these deployments were top performers demonstrating our intuition was insufficient. Testing was required to determine the fastest component placements. We observed up to ∼26% performance variation comparing component deployments while making no application changes only deployment changes. This variation illustrates the possible consequences for ad hoc component placement.

5.3. Provisioning variation testing

IaaS cloud providers often do not allow user-level control of VM placement to physical hosts. The non-determinism of where VMs are hosted results in provisioning variation [2–4]. In the previous section we identified the best performing application component deployments. We had two primary motivations for provision variation testing. First, to validate if deploying VMs using isolated physical hosts was sufficient to identify the best performing component deployments. For example does one of the deployments (SC11A, SC11B, SC11C) provide fundamentally different performance than SC11? And second, to quantify the average performance change for provisioning variation configurations. Intuition and previous research suggest that hosting multiple VMs on a single PM will reduce performance, but by how much?

There are 45 provisioning variations of the 15 component deployments described in Table 2 and tested in previous sections. 31 of the configurations were tested using the 20 randomized 100-model run ensembles and KVM-based VMs with 4 GB memory allocation. Test configurations are identified by their base service configuration id SC1–SC15 and the letters A–D to identify provisioning variation configurations as described in Table 8. There were 14 variations of SC7 which represent the VM-level service isolation variants of component configurations described in Table 2. These were not tested because service isolation only adds overhead relative to their equivalents (SC1–SC6, SC8–SC15) as discussed in Section 5.6 for RQ-3. To compare performance the provision variation deployments from Table 8 versus SC1–SC15, we calculated averages for provisioning variation configurations having more than 1 provisioning variation deployment (e.g. SC11A, SC11B, SC11C, SC11D). *Linear regression showed that component*

Table 8
Provisioning variation VM tests.

	PM 1	PM 2		PM 1	PM 2
SC2A	$[\mathcal{MDF}]$ $[\mathcal{L}]$		SC9B	$[\mathcal{M}]$ $[\mathcal{DL}]$ $[\mathcal{F}]$	
SC3A	$[\mathcal{MD}]$ $[\mathcal{FL}]$		SC9C	$[\mathcal{M}]$	$[\mathcal{DL}]$ $[\mathcal{F}]$
SC4A	$[\mathcal{MD}]$ $[\mathcal{F}]$	$[\mathcal{L}]$	SC9D	$[\mathcal{M}]$$[\mathcal{F}]$	$[\mathcal{DL}]$
SC4B	$[\mathcal{MD}]$ $[\mathcal{F}]$ $[\mathcal{L}]$		SC10A	$[\mathcal{MF}]$ $[\mathcal{DL}]$	
SC4C	$[\mathcal{MD}]$	$[\mathcal{F}]$ $[\mathcal{L}]$	SC11A	$[\mathcal{MF}]$ $[\mathcal{D}]$	$[\mathcal{L}]$
SC4D	$[\mathcal{MD}]$ $[\mathcal{L}]$	$[\mathcal{F}]$	SC11B	$[\mathcal{MF}]$ $[\mathcal{D}]$ $[\mathcal{L}]$	
SC5A	$[\mathcal{M}]$ $[\mathcal{DFL}]$		SC11C	$[\mathcal{MF}]$	$[\mathcal{D}]$ $[\mathcal{L}]$
SC6A	$[\mathcal{M}]$ $[\mathcal{DF}]$	$[\mathcal{L}]$	SC11D	$[\mathcal{MF}]$ $[\mathcal{L}]$	$[\mathcal{D}]$
SC6B	$[\mathcal{M}]$ $[\mathcal{DF}]$ $[\mathcal{L}]$		SC12A	$[\mathcal{ML}]$ $[\mathcal{DF}]$	
SC6C	$[\mathcal{M}]$	$[\mathcal{DF}]$ $[\mathcal{L}]$	SC13A	$[\mathcal{ML}]$ $[\mathcal{D}]$	$[\mathcal{F}]$
SC6D	$[\mathcal{M}]$ $[\mathcal{L}]$	$[\mathcal{DF}]$	SC13B	$[\mathcal{ML}]$ $[\mathcal{D}]$ $[\mathcal{F}]$	
SC8A	$[\mathcal{M}]$ $[\mathcal{D}]$	$[\mathcal{FL}]$	SC13C	$[\mathcal{ML}]$	$[\mathcal{D}]$$[\mathcal{F}]$
SC8B	$[\mathcal{M}]$ $[\mathcal{D}]$ $[\mathcal{FL}]$		SC13D	$[\mathcal{ML}]$ $[\mathcal{F}]$	$[\mathcal{D}]$
SC8C	$[\mathcal{M}]$	$[\mathcal{D}]$ $[\mathcal{FL}]$	SC14A	$[\mathcal{MDL}]$	$[\mathcal{F}]$
SC8D	$[\mathcal{M}]$ $[\mathcal{FL}]$	$[\mathcal{D}]$	SC15A	$[\mathcal{MLF}]$ $[\mathcal{D}]$	
SC9A	$[\mathcal{M}]$ $[\mathcal{DL}]$	$[\mathcal{F}]$			

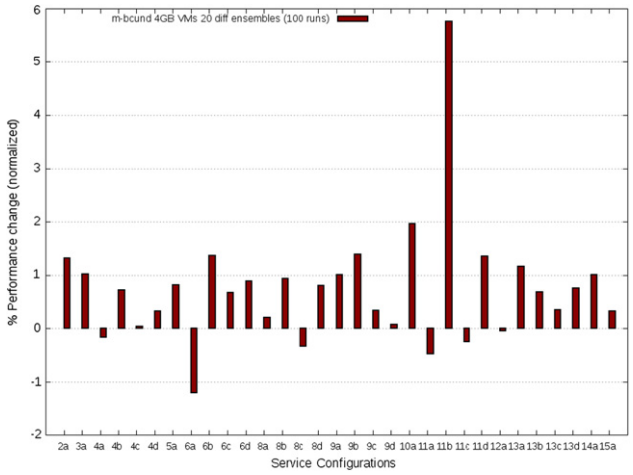

Fig. 4. Provisioning variation performance differences vs. physical isolation (KVM).

deployments performed the same regardless of provisioning variation ($R^2 = 0.956701, df = 13, p = 3.03 \cdot 10^{-10}$), *though they generally performed slower*. Performance differences observed appeared to result from hosting multiple VMs on physical hosts. On average performance for provision variation configurations was 2.5% slower. Configurations with 2 VMs averaged 3.05% slower, with 3 VMs 2.33% slower. 6 of 31 configurations exhibited small performance gains: SC4A, SC6A, SC8C, SC11A, SC11C, and SC12A. Provisioning variation configurations which separated physical hosting of the $\mathcal{M}$ and $\mathcal{L}$ components provided an average improvement of 0.39% (10 configurations) whereas those which combined hosting of $\mathcal{M}$ and $\mathcal{L}$ were on average 3.93% slower. Performance differences for provisioning variation configurations are shown in Fig. 4.

5.4. Increasing VM memory

In [31] the RUSLE2 model was used to investigate multi-tier application scaling with components deployed on isolated VMs. VMs hosting the $\mathcal{M}$, $\mathcal{F}$, and $\mathcal{L}$ components were allocated 2 GB memory, and the $\mathcal{D}$ component VM was allocated 4 GB. To avoid performance degradation due to memory contention VM memory was increased to 10 GB, the total amount provided using individual VMs in [31]. Intuitively increasing VM memory should provide either a performance improvement or no change of performance. 20 runs of an identical 100-model run ensemble were repeated for the SC1–SC15 component deployments using 10 GB VMs. Fig. 5 shows performance changes resulting from increasing VM memory allocation from 4 GB to 10 GB for both the "*m*-bound" and "*d*-bound" applications.

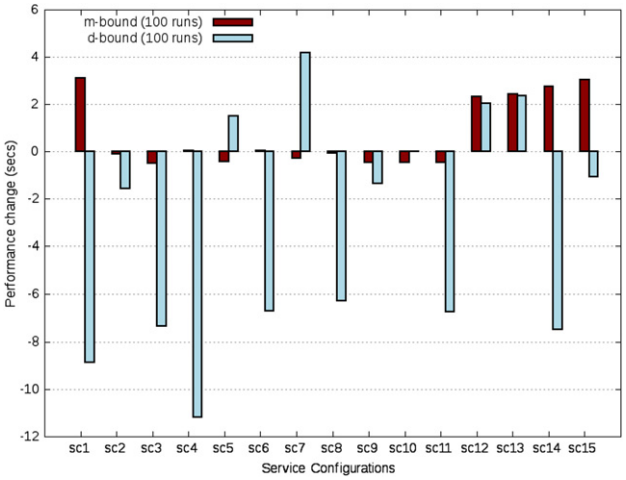

Fig. 5. 10 GB VM performance changes (seconds).

For the "*m*-bound" application using 10 GB VMs reduced average ensemble performance 0.727 s (−3.24%) versus using VMs with 4 GB. SC11 provided had the best performance, 6.7% faster than average component deployment performance for 10 GB VM "*m*-bound" ensemble tests. This was half a second faster than with 4 GB VMs. SC1, total service combination, performed the slowest, 8.9% slower than average, 3.1 s longer than with 4 GB VMs. For the "*m*-bound" application only component deployments which combined $\mathcal{M}$ and $\mathcal{L}$ 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 and required the ia32 Linux 32-bit compatibility libraries to run on a 64-bit VM. The performance degradations may have resulted from virtualization of the ia32 library as 32-bit Linux can only natively address up to 4 GB RAM.

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

To verify that these results were not specific to repeated runs of an identical 100-model run ensemble using the Xen hypervisor, we also tested increasing VM memory allocation using 20 different ensembles and the KVM hypervisor. Results were similar for both cases. The "*m*-bound" model's 15 component deployments performed on average 342 ms slower (−1.13%) with 10 GB VMs and the "*d*-bound" model performed 3.24 s (2.46%) faster on average. Our results demonstrate that *increasing VM memory allocation may result in unexpected performance changes in some cases exceeding* +/−10%. For VM memory allocation, depending on the application, more may not always be better.

5.5. Xen vs. KVM

To compare performance differences between the Xen and KVM hypervisors we ran test sets using 20 different ensemble runs using 4 GB VMs and the "*m*-bound" application. Tests were repeated using both Xen and KVM hypervisors, random ensembles and the "*d*-bound" model. *On average KVM ensemble performance was* ∼29% *slower than Xen for the "m-bound" model, but* ∼1% *faster for the "d-bound" model.* The "*d*-bound" model was more CPU bound enabling performance improvement compared with Xen. The "*m*-bound" model had a higher proportion of I/O relative to CPU use and performed faster using Xen. "*D*-bound" ensemble tests using KVM required on average 4.35 xs longer than the "*m*-bound" model, while Xen "*d*-bound" runs were 5.7× longer than "*m*-bound". The average performance difference between the

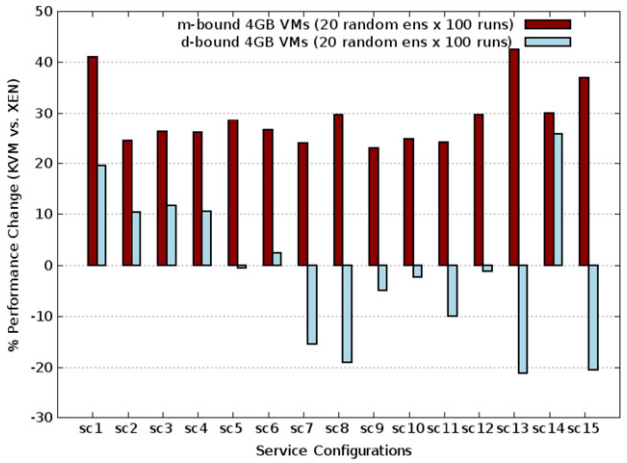

Fig. 6. Xen vs. KVM performance differences, 4 GB VM different ensembles.

Table 9
KVM vs. Xen resource utilization — randomized ensembles.

Parameter	KVM resource utilization (% of Xen)	R^2	p
CPU time (s)	135.2	0.787769	0.00001
Disk sector reads	97.91	0.804115	$5.96 \cdot 10^{-6}$
Disk sector writes	50.48	0.829572	$2.38 \cdot 10^{-6}$
Network bytes rec'd	101.77	0.999872	$1.09 \cdot 10^{-26}$
Network bytes sent	101.85	0.999874	$9.61 \cdot 10^{-27}$

Xen and KVM hypervisors for running both the "*m*-bound" and "*d*-bound" models using 20 random ensemble tests is shown in Fig. 6.

Resource utilization data was collected for the "*m*-bound" model for all component deployments (SC1–SC15) using 20 random 100-model run ensemble tests for the Xen and KVM hypervisors. Resource utilization differences and correlations are summarized in Table 9. Resource utilization for Xen and KVM correlated for all statistics. On average KVM used 35% more CPU time than Xen, but nearly an equal number of disk sector reads (98%), but performed far fewer disk sector writes (50%). KVM exhibited 1.8% more network traffic (bytes sent/received) than Xen. Increased CPU time for KVM may result from KVM's full virtualization of devices where devices are entirely emulated by software. Xen I/O uses paravirtual devices which offer more direct device I/O.

We used a simple linear regression to compare Xen and KVM performance of component deployments for the "*m*-bound" and "*d*-bound models. Deployments using 4 GB VMs for the "*m*-bound" model with random ensembles were shown to perform similarly ($R^2 = 0.749912, df = 13, p = 0.00003$). Component deployment performance of Xen vs. KVM using the "*d*-bound" model performance did not correlate. Given KVM's improved "*d*-bound" performance relative to Xen, this result was expected. Application performance using the KVM hypervisor appeared to be more sensitive than Xen to disk I/O.

5.6. Service isolation overhead

To investigate overhead resulting from the use of separate VMs to host application components the three fastest component deployments for the "*m*-bound" model were tested. Components were deployed using the SC2, SC6, and SC11 configurations with and without using separate VMs to host individual components.

60 runs using the same 100-model run ensemble, and 3 test sets of 20 different 100-model run ensembles were completed for each configuration. The percentage performance change resulting from service isolation is shown in Fig. 7. *For all but one configuration, service isolation resulted in overhead which degraded performance compared to deployments where multiple components were combined on VMs.* The average overhead from service isolation was ∼1%. For tests using different ensembles the observed performance degradation for service isolation deployments was 1.2%, 0.3%, and 2.4% for SC2-SI, SC6-SI and SC11-SI respectively. The same ensemble test performance degradation was 1.1%, −0.06%, and 1.4%. These results were reproduced using the KVM hypervisor with an average observed performance degradation of 2.4%.

Although the 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 design using isolated physical hardware, running no other applications, allows us to be certain that observed overhead resulted entirely from VM-level service isolation. This overhead is one of the tradeoffs for easier application tier-scalability with service isolation.

Table 10
Resource utilization — predictive power.

Parameter	R^2	RMSD
CPU time	0.7171	887.64
# Disk sector reads	0.3714	1323.25
# Disk sector writes	0.1441	1544.05
Network bytes rec'd.	0.0074	1662.76
Network bytes sent	0.0075	1662.68
Number of VMs	0.0444	1631.44

5.7. Predictive model

Resource utilization data was collected for CPU time, disk sector reads/writes, and network bytes sent/received as described in Section 5.1. We observed resource utilization variation for each of the deployments tested. Multiple linear regression (MLR) was used to build models to predict component deployment performance using resource utilization data to support investigation of RQ-4.

Multiple linear regression (MLR) is used to model the linear relationship between a dependent variable and one or more independent variables [32]. The dependent variable was 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 of the component deployment. 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 for each independent variable using single linear regression. Root mean squared deviation (RMSD) was calculated for each variable. 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 4 GB 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 ($R^2 = 0.8416$, *RMSD* = 664.17 ms). Table 10 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 ($R^2 = 0.7171$) and disk sector reads ($R^2 = 0.3714$) with a negative correlation. Disk sector writes had a positive correlation with ensemble

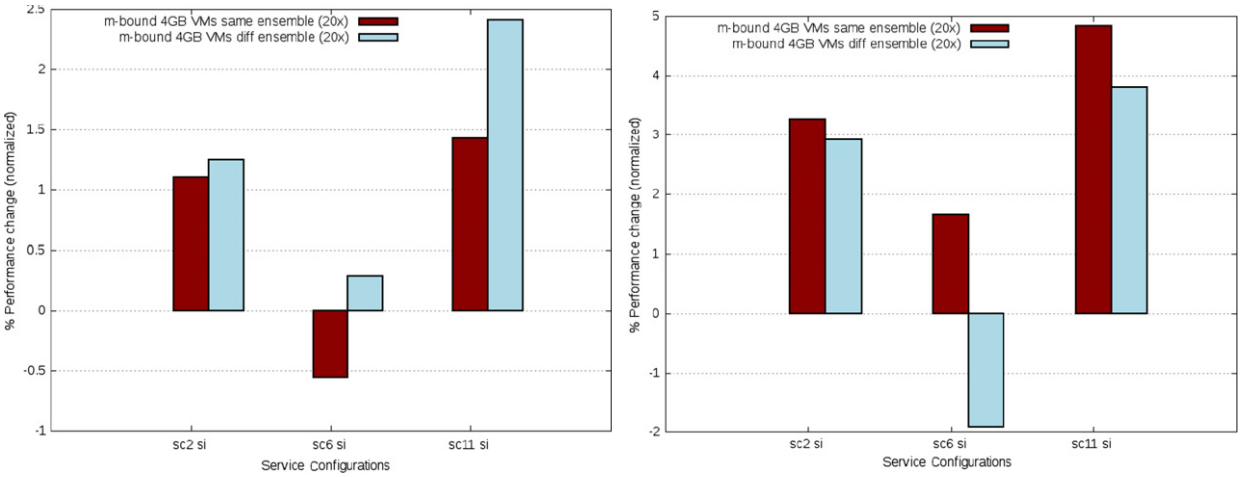

Fig. 7. Performance overhead from service isolation (Xen left, KVM right).

Table 11
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

performance ($R^2 = 0.1441$). The number of deployment VMs ($R^2 = 0.0444$) and network bytes received/sent were not strong predictors of ensemble performance and explained very little variance.

We applied our MLR performance model to predict performance of component deployments. Resource utilization data used to generate the model was reused to generate ensemble time predictions. Average predicted ensemble execution times were calculated for each component deployment (SC1–SC15) and rank predictions were calculated. Predicted vs. actual performance ranks are shown in Table 11. 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 set of resource utilization data was collected for the "*m*-bound" model using 4 GB VMs and 20 random ensembles for SC1–SC15. This data was fed into our MLR performance model and observed MAE was only 324 ms. 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 top fastest deployment was correctly predicted for the second dataset.

Building models to predict component deployment performance requires careful consideration of resource utilization variables. This initial attempt using multiple linear regression was helpful to identify which independent variables had the greatest impact on deployment performance. Future work to improve performance prediction should investigate using additional resource utilization statistics as independent variables to improve model accuracy. New variables including CPU statistics, kernel scheduler statistics, and guest/host load averages should be explored. The utility of neural networks, genetic algorithms, and/or support vector machines to improve our model should be investigated extending related research [17,33–37]. These techniques can help improve performance predictions if resource utilization data is not normally distributed.

6. 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. 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. Component deployment using total service isolation did not provide the fastest performance for our application. Provisioning variation did not change the fundamental performance of component deployments but did produce overhead of ∼2%–3% when two or more VMs resided on the same physical host.

(RQ-2) Increasing VM memory allocation did not guarantee application performance improvements. Increasing VM memory to improve performance appears useful only if memory is the application's performance bottleneck. The KVM hypervisor performed 29% slower than Xen when application performance was bound by disk I/O but slightly faster ∼1% when the application was CPU bound. KVM resource utilization correlated with Xen but CPU time was 35% greater when KVM was used to perform the same work.

(RQ-3) Service isolation, the practice of using separate VMs to host individual application components resulted in performance overhead up to 2.4%. Though overhead may be small, the hosting costs for additional VMs should be balanced with the need to granularly scale application components. Deploying an application using total service isolation will always result in the highest possible hosting costs in terms of the # of VMs.

(RQ-4) Resource utilization statistics were helpful for building performance models to predict performance of component deployments. Using just 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.

Providing VM/application level resource load balancing and using compact application deployments holds promise for improving application performance while lowering application hosting costs. To support load balancing and cloud infrastructure management, performance models should be investigated further as they hold promise to help guide intelligent application deployment and resource management for IaaS clouds.

References

[1] J. Varia, Architecting for the cloud: best practices. Amazon Web Services White Paper, 2010. https://jineshvaria.s3.amazonaws.com/public/cloudbestpractices-jvaria.pdf.

[2] M. Rehman, M. Sakr, Initial findings for provisioning variation in cloud computing, in: 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.

[3] J. Schad, J. Dittrich, J. Quiane-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proceedings of the VLDB Endowment 3 (1–2) (2010) 460–471.

[4] M. Zaharia, A. Konwinski, A. Joesph, R. Katz, I. Stoica, Improving MapReduce performance in heterogeneous environments, in: Proc. 8th USENIX Conf. Operating Systems Design and Implementation, OSDI '08, San Diego, CA, USA, Dec 8–10, 2008, pp. 29–42.

[5] M. Schwarzkopf, D. Murray, S. Hand, The seven deadly sins of cloud computing research, in: Proc. 4th USENIX Conf. on Hot Topics in Cloud Computing, HotCloud '12, Boston, MA, June 12–13, 2012, p. 5.

[6] J. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, J. Owens, Quantifying the performance isolation properties of virtualization systems, in: Proc. ACM Workshop on Experimental Computer Science, ExpCS '07, New York, NY, USA, 2007, Article 6.

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

[8] H. Raj, R. Nathuji, A. Singh, P. England, Resource Management for isolation enhanced cloud services, in: Proc. ACM Cloud Comp. Security Wrkshp, CCSW '09, Chicago, IL, USA, Nov 2009, pp. 77–84.

[9] S. Govindan, J. Liu, A. Kansal, A. Sivasubramaniam, Cuanta: quantifying effects of shared on-chip resource interference for consolidated virtual machines, in: Proc. 2nd ACM Symposium on Cloud Computing, SOCC 2011, Cascais, Portugal, Oct 26–28, 2011, p. 14.

[10] A. Gulati, G. Shanmuganathan, I. Ahmad, C. Waldspurger, M. Uysal, Pesto: online storage performance management in virtualized datacenters, in: Proc. 2nd ACM Symposium on Cloud Computing, SOCC 2011, Cascais, Portugal, Oct 26–28, 2011, p. 14.

[11] H. Kang, Y. Chen, J. Wong, J. Wu, Enhancement of Xen's scheduler for MapReduce workloads, in: Proc. 20th Intl. ACM Symposium on High-Performance Parallel and Distributed Computing, HPDC '11, San Jose, CA, June 8–11, 2011, pp. 251–262.

[12] T. Voith, K. Oberle, M. Stein, Quality of service provisioning for distributed data center inter-connectivity enabled by network virtualization, Future Generation Computer Systems 28 (2012) 554–562.

[13] M. Vouk, Cloud computing — issues, research, and implementations, in: Proc. 30th Intl. Conf. Information Technology Interfaces, ITI 2008, Cavtat, Croatia, June 23–26, 2008, pp. 31–40.

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

[15] D. Armstrong, K. Djemame, Performance issues in clouds: an evaluation of virtual image propagation and I/O paravirtualization, The Computer Journal 54 (6) (2011) 836–849.

[16] D. Jayasinghe, S. Malkowski, Q. Wang, J. Li, P. Xiong, C. Pu, Variations in performance and scalability when migrating *n*-tier applications to different clouds, in: Proc. 4th IEEE Conf on Cloud Computing, Cloud '11, Washington D.C., USA, July 2011, pp.73–80.

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

[18] B. Sharma, V. Chudnovsky, J. Hellerstein, R. Rifaat, C. Das, Modeling and synthesizing task placement constraints in Google compute clusters, in: Proc. 2nd ACM Symposium on Cloud Computing, SOCC 2011, Cascais, Portugal, Oct 26–28, 2011, p. 14.

[19] 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.

[20] Apache Tomcat — Welcome, 2011. http://tomcat.apache.org/.

[21] L. Ahuja, J. Ascough II, O. David, Developing natural resource modeling using the object modeling system: feasibility and challenges, Advances in Geosciences 4 (2005) 29–36.

[22] O. David, J. Ascough II, G. Leavesley, L. Ahuja, Rethinking modeling framework design: Object Modeling System 3.0, in: Proc. iEMSs 2010 Intl. Congress on Environmental Modeling and Software, Ottawa, Canada, July 5–8, 2010, p. 8.

[23] WineHQ — run windows applications on Linux, BSD, Solaris, and Mac OS X. http://www.winehq.org/.

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

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

[26] Nginx news, 2011. http://nginx.org/.

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

[28] HAProxy — The reliable, high performance TCP/HTTP load balancer. http://haproxy.1wt.eu/.

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

[30] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, A. Warfield, Xen and the art of virtualization, in: Proc. 19th ACM Symposium on Operating Systems Principles, SOSP '03, Bolton Landing, NY, USA, Oct 19–22, 2003, p. 14.

[31] 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, in: Proc. 12th IEEE/ACM Intl. Conf. On Grid Computing, GRID 2011, Lyon, France, Sept 21–23, 2011, pp. 137–143.

[32] R.H. Myers, Classical and Modern Regression with Applications, second ed., PWS-KENT Publishing Company, Boston, MA, 1994.

[33] 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, The Journal of Systems and Software 84 (2011) 1270–1291.

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

[35] C. Xu, J. Rao, X. Bu, URL: a unified reinforcement learning approach for autonomic cloud management, Journal of Parallel and Distributed Computing 72 (2012) 95–105.

[36] 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 23 (1) (2012) 78–86.

[37] P. Lama, X. Zhou, Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee, in: 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.

W. Lloyd is a Ph.D. student in the Department of Computer Science at Colorado State University (CSU). Wes is also a research associate with the Department of Civil Engineering at CSU. Wes works via a cooperative agreement with the US Department of Agriculture — Natural Resources Conservation Service (USDA-NRCS) on the Cloud Services Innovation Platform project which aims to harness cloud-computing technology to enable web services based deployment and computational scaling of environmental science models for agency wide use.

S. Pallickara is an Assistant Professor in the Department of Computer Science at Colorado State University. Dr. Pallickara's research interests are in the area of large-scale distributed systems specifically, computing and streaming. Dr. Pallickara has created two systems: Granules which is a distributed stream processing system, and the NaradaBrokering system for the scalable dissemination of voluminous streams. All systems software from these projects are in the open-source domain.

O. David is a research scientist at the Department of Civil Engineering at Colorado State University. Olaf works via a cooperative agreement with the US Department of Agriculture — Natural Resources Conservation Service (USDA-NRCS) on the Cloud Services Innovation Platform project and the Object Modeling System (OMS) framework. The OMS framework is a component-based non-invasive lightweight framework which encourages good software engineering practices for environmental model development.

J. Lyon is a software developer at the Department of Civil Engineering at Colorado State University. Jim works via a cooperative agreement with the US Department of Agriculture — Natural Resources Conservation Service (USDA-NRCS) on the Cloud Services Innovation Platform project and the Object Modeling System (OMS) framework.

M. Arabi is an Assistant Professor in the Department of Civil Engineering at Colorado State University. Dr. Arabi is teaching and actively conducting research in the water resources management and planning and environmental engineering areas. Mazdak received his Ph.D. in August 2005 from Purdue University.

K. Rojas is a senior project manager with the Information Technology Center (ITC) of the USDA-Natural Resources Conservation Service (NRCS), in Fort Collins Colorado. Ken manages the development, enhancement, and maintenance of 70+ client and enterprise applications for the NRCS which delivers technical assistance to farmers and ranchers across the US. Most efforts are directed towards supporting the transfer of science applications for use as conservation assessment tools for the conservation streamlining delivery initiative (CDSI). Using an enterprise service approach, CDSI will help NRCS conservationists deliver assistance to customers from 2000+ field offices located nationwide.