11-1-2012

Performance Modeling to Support Multi-Tier Application Deployment to Infrastructure-As-A-Service Clouds

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

Performance Modeling to Support Multi-Tier Application Deployment to Infrastructure-as-a-Service Clouds

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*— Infrastructure-as-a-service (IaaS) clouds support migration of multi-tier applications through virtualization of diverse application stack(s) of components which may require various operating systems and environments. To maximize performance of applications deployed to IaaS clouds while minimizing deployment costs, it is necessary to create virtual machine images to host application components with consideration for component dependencies that may affect load balancing of physical resources of VM hosts including CPU time, disk and network bandwidth. This paper presents results of an investigation utilizing physical machine (PM) and virtual machine (VM) resource utilization statistics to build performance models to predict application performance and rank performance of application component deployment configurations deployed across VMs. Our objective was to predict which component compositions provide best performance while requiring the fewest number of VMs. Eighteen individual resource utilization statistics were investigated for use as independent variables to predict service execution time using four different modeling approaches. Overall CPU time was the strongest predictor of execution time. The strength of individual predictors varied with respect to the resource utilization profiles of the applications. CPU statistics including idle time and number of context switches were good predictors when the test application was more disk I/O bound, while disk I/O statistics were better predictors when the application was more CPU bound. All performance models built were effective at determining the best performing service composition deployments validating the utility of our approach.**

Keywords Cloud Computing; Infrastructure-as-a-Service; Performance Modeling; Provisioning Variation; Virtualization

I. INTRODUCTION

Migration of multi-tier client/server applications to Infrastructure-as-a-Service (IaaS) clouds requires applications be decomposed into sets of service-based components known as the application stack. Application stacks consist of components such as web server(s), application server(s), proxy server(s), database(s), file server(s) and other servers/services.

Infrastructure-as-a-Service clouds support better utilization of server infrastructure by enabling multiplexing of resources. Infrastructure supporting specific applications can be scaled based on demand while multiple applications share the physical infrastructure through the use of server virtualization. IaaS clouds consisting of many physical servers with one or more multi-core CPUs can host Virtual Machines (VMs) enabling resource elasticity where the quantity, size, and location of VMs can change dynamically to meet varying system demand.

Many challenges exist when deploying multi-tier applications to Infrastructure-as-a-Service clouds. VM *image composition* requires application components to be composed across a set of VM images. Resource contention should be minimized by taking advantage of opportunistic placements by collocation of codependent components. *Provisioning variation* refers to the uncertainty of the physical location of VMs when deployed to IaaS clouds [2]. VM physical location could lead to performance improvements or degradation depending on component resource requirements and interdependencies. *Internal resource contention* occurs when application VMs are provisioned to the same physical machines (PMs) while competing for the same resources. *External resource contention* can occur when different applications share physical infrastructure an important issue for public clouds. *Virtualization overhead* refers to the costs associated with emulating a computer as a software program on a physical host computer. This overhead varies depending on the approaches used to multiplex physical resources among virtual hosts. Virtualization hypervisors vary with respect to their ability to minimize this overhead with some generally responding better to certain resource sharing and simulation scenarios than others [1][4][5][16]. *Resource provisioning* refers to the challenge of allocating adequate virtual infrastructure to meet performance requirements while accounting for the challenges of image composition, provisioning variation, resource contention, and virtualization overhead. Research and investigation into approaches supporting autonomic resource provisioning also known as autonomic infrastructure management is an active area of cloud computing research [17][18][19][20].

Service compositions must be determined which map application stacks across VM images. Determining beneficial combinations of components which multiplex resources without causing unwanted resource contention poses a challenge. Component compositions will vary for multi-tier applications as applications have different

application stacks of components and resource utilization profiles further complicating determination of ideal VM component deployments.

Using brute force performance testing to determine optimal placements is only feasible for applications with small numbers of components. Bell's number is the number of partitions of a set (k) consisting of (n) members [21]. If we consider an application as a set of (n) components, then the total number of possible component compositions for an application is Bell's number (k). Table 1 shows the first few Bell numbers describing the possible number of component compositions. As the number of components increases, the possible number of service compositions grows rapidly making the use of brute force testing to benchmark performance impractical. Web applications such as mashup applications which aggregate many data sources and application programming interfaces (APIs) may have application stacks with a large number of components. Complicating matters further, public IaaS clouds often do not provide the ability to control VM placements making it difficult, if not impossible, to deploy all possible placements. Exclusive reservation of PMs may be available for an additional cost enabling granular control of physical placement of VMs.

TABLE I. MULTI-TIER APPLICATION SERVICE COMPOSITIONS

Number of Components (n)	Number of Compositions (B_n)
3	5
4	15
5	52
6	203
7	877
8	4,140

An exponential generating function to generate Bell numbers is given by the formula:

$$\sum_{n=0}^{\infty} \frac{B_n}{n!} x^n = e^{e^x - 1}.$$

Performance models hold promise as a means to rapidly evaluate a large number of possible service compositions without physically deploying and testing them. Good performance models should be able to predict performance outcomes of VM placement and service compositions allowing brute force testing of the entire configuration space to be avoided. Collecting training data to train performance models should be easier than performing brute force testing of all service compositions. Models should provide the ability to make reasonably accurate performance predictions with reasonable amounts of time spent collecting training data and training models.

This paper presents results of an exploratory study which investigates building multi-tier application performance models using resource utilization statistics. The utility of using different resource utilization statistics as independent variables for predicting service response time is investigated. Performance models of multi-tier applications deployed to IaaS clouds hold promise to (1) guide application component placement across VM images, and (2) support real-time virtual infrastructure management for IaaS clouds by predicting resource requirements for specific performance goals.

The following research questions are investigated in support of our investigation on Infrastructure-as-a-Service application performance modeling:

1) (Independent Variables) Which VM and PM resource utilization statistics are most helpful for predicting performance of different application service compositions?

2) (Profiling Data) How should resource utilization data be treated for use in performance models? Should VM profiling data from multiple VMs be combined or used separeatly?

3) (Exploratory Modeling) Comparing multiple linear regression (MLR), multivariate adaptive regression splines (MARS), and an artificial neural network (ANN), which model techniques appear to best predict application performance and service composition performance ranks?

II. RELATED WORK

Rouk identified the challenge of creating good virtual machine images which compose together application components for migrating multi-tier client/server applications to IaaS clouds in [3]. Negative performance implications and higher hosting costs may result when ad hoc compositions are used resulting in potential unwanted contention for physical resources. Xu et al. identify two classes of approaches for providing autonomic provisioning and management of virtual infrastructure in [17]: *multivariate optimization* (performance modeling), and *feedback control*. Multivariate optimization approaches attempt to support better application performance by modeling the tuning of multiple system variables to predict the best configurations. Feedback control approaches based on process control theory attempt to improve configurations by iteratively making changes and observing outcomes in real time using live systems. Feedback control approaches have been built using reinforcement learning [17], support vector machines (SVMs) [18], ANNs [27][28], and a fitness function [29]. Performance models have been built using MLR [20], ANNs [17][19], and SVMs [18]. Hybrid approaches which combine the use of a performance model for model initialization and apply real time feedback control include: [17][18][27][28].

Multivariate optimization approaches can model far more configurations enabling a much larger portion of the exploration space of system configurations to be considered. Time to collect and analyze model training datasets results in a trade-off between model accuracy vs. availability. Additionally performance models trade-off accuracy vs. complexity. More complex models with larger numbers of independent variables and data samples require more time to build and compute but this investment can lead to better model accuracy.

Feedback control approaches apply control system theory to actively tune resources to meet pre-stated service level agreements (SLAs). Feedback control systems do not determine optimal configurations as they only consider a subset of all possible configurations limited by observations of configurations seen in real time. Feedback control approaches may produce inefficient configurations, particularly upon system initialization. Hybrid approaches combine performance modeling and feedback control to provide better control decisions more rapidly. Hybrid systems use training datasets to initialize performance models to better inform control decisions immediately upon start-up. Control decisions are further improved as the system operates and collects additional data in real time. Hybrid approaches often use simplified performance models trading off accuracy for speed of computation and initialization.

Wood et al. developed Sandpiper, a black-box and gray-box resource manager for VMs [20]. Sandpiper, a feedback control approach, was designed to oversee server partitioning and was not designed specifically for IaaS. Sandpiper detects "Hotspots" when provisioned architecture fails to meet service demand. Sandpiper performs only vertical scaling including increasing available resources to VMs, and VM migration to less busy PMs but does not horizontally scale the number of VMs for load balancing. Sandpiper uses a MLR performance model to predict service time by considering CPU utilization, network bandwidth utilization, page fault rate, memory utilization, request drop rate, and incoming request rate as independent variables. Xu et al. developed a resource learning approach for autonomic infrastructure management [17]. Both application agents and VM agents were used to monitor performance. A state/action table was built to record performance quality changes resulting from control events. Their resource learning approach only considered VM memory allocation, VM CPU cores, and CPU scheduler credit. An ANN model was added to predict reward values to help improve performance upon system initialization when the state/action table was only sparsely populated. Kousiouris et al. benchmarked all possible configurations for different task placements across several VMs running on a single PM [19]. From their observations they developed both a MLR model and an ANN to model performance. Their research was not extended to perform resource control but focused on performance modeling to predict the performance implications of task placements. Kousiouris et al.'s approach used an ANN to model task performance for different VM configurations on a single machine. They contrasted using a ANN model with a MLR model. Model independent variables included: CPU scheduling time, and location of tasks (same CPUs with L1 & L2 cache sharing, adjacent CPUs with L2 cache sharing, and non-adjacent CPUs). Niehorster et al. developed an autonomic resource provisioning system using support vector machines (SVMs) [18]. Their system responds to service demand changes and alters infrastructure configurations to enforce SLAs. They performed both horizontal and vertical scaling of resources and dynamically configured application specific parameters. Niehorster et al.'s performance model primarily considered application specific parameters. The only virtual infrastructure parameters considered in their performance model included # of VMs, VM memory allocation, and VM CPU cores.

III. Paper Contributions

Existing approaches using performance models to support autonomic infrastructure management do not adequately consider performance implications of where application components are physically hosted across VMs. Additionally, existing approaches do not consider disk utilization statistics, and only one approach has considered implications of network I/O throughput [20]. This paper extends prior work by investigating the utility of using VM and PM resource utilization statistics as predictors for performance models for applications deployed to IaaS clouds. Use of application performance models can support determination of ideal component compositions which maximize performance using minimal resources to support autonomic multi-tier application deployment across VMs. These performance models can also support autonomic IaaS cloud virtual infrastructure management by predicting outcomes of potential configuration changes without physically testing them. To support our investigation we modeled performance of two variants of a multi-tier scientific erosion model. The variants serve as surrogates for common multi-tier applications: an application-server bound application and a relational database bound application.

IV. Experimental Investigation

A. Experimental Setup

The test infrastructure used to explore multi-tier application migration in [26] was extended to explore our application performance modeling research questions presented in section 1. Two variants of the Revised Universal Soil Loss Equation – Version 2 (RUSLE2), an erosion model, were deployed as a web service and tested using a private IaaS cloud environment. RUSLE2 contains both empirical and process-based science that predicts rill and interrill soil erosion by rainfall and runoff [6]. RUSLE2 was developed primarily to guide conservation planning, inventory erosion rates, and estimate sediment delivery and is the USDA-NRCS agency standard model for sheet and rill erosion modeling used by over 3,000 field offices across the United States. RUSLE2 is a good candidate to prototype multi-tier application performance modeling because its architecture consisting of a web server, relational database, file server, and logging server is analogous to many typical multi-tier client/server based applications.

RUSLE2 was deployed as a JAX-RS RESTful JSON-based web service hosted by Apache Tomcat [9]. The Object Modeling System 3.0 (OMS 3.0) framework [7][22] using WINE [8] was used as middleware to support model integration and deployment as a web service. OMS 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.

A Eucalyptus 2.0 [10] IaaS private cloud was built and hosted by Colorado State University consisting of 9 SUN X6270 blade servers on the same chassis sharing a private Giga-bit VLAN with dual Intel Xeon X5560-quad core 2.8 GHz CPUs each with 24GB ram and 146GB HDDs. 8 blade servers were configured as Eucalyptus node-controllers, and 1 blade server was configured as the Eucalyptus cloud-controller, cluster-controller, walrus server, and storage-controller. The cloud controller server was supported by Ubuntu Linux (2.6.35-22) 64-bit server 10.10, while node controllers which hosted VMs used CentOS Linux (2.6.18) 64-bit server. Eucalyptus managed mode networking was used to isolate experimental VMs on their own private VLANs. The XEN hypervisor version 3.4.3 supported by QEMU version 0.8.2 was used to provide VMs [16]. Version 3.4.3 of the hypervisor was selected after testing indicated it provided the best performance when compared with other versions of XEN (3.1, 4.0.1, and 4.1).

To facilitate testing, ensemble runs, groups of individual modeling requests bundled together were used. To invoke the web service a client sends a JSON object representing a collection of parameterized model requests with values for management practice, slope length, steepness, latitude, and longitude. Model results are computed and returned using JSON object(s). Ensemble runs are processed by dividing grouped modeling requests into individual requests which are resent to the web service, similar to the "map" function of MapReduce. A configurable number of worker threads concurrently execute individual runs in parallel. Modeling results are then combined (reduced) and returned as a single JSON response object. A test generation program created randomized ensembles. Latitude and longitude coordinates were randomly selected within a bounding box from the U.S. state of Tennessee. Slope length, steepness, and the management practice parameters were also randomized. 20 randomly generated ensemble tests with 100 model runs each were used to test performance of 15 different service compositions. Before executing each 100 model-run ensemble test, a smaller 25 model-run ensemble test was executed to warm up the system. The warm up test was warranted after observing slow spatial query performance from postgresql on startup.

A test script was used to automatically configure service placements and collect VM and PM resource utilization statistics while executing ensemble tests. Cache clearing using the Linux virtual memory drop_caches function was used to purge all caches, dentries and inodes before each test was executed to negate training affects resulting from reusing ensemble tests. The validity of this approach was verified by observing CPU, file I/O, and network I/O utilization statistics for the automated tests with and without cache clearing. When caches were not cleared the number of disk sector reads dropped after the system was initially exposed to the test dataset. When caches were force-cleared the system exhibited more disk reads confirming it was forced to reread data each time. Initial experimental observations showed that as the number of records stored in the logging database increased, ensemble test performance declined. To work around performance effects of the growing logs and to eliminate running out of disk space, the Codebeamer logging component was removed and reinstalled after each ensemble test run. Additionally all log files for all application components were purged after each ensemble test. These steps allowed several thousand ensemble tests using all of the required service compositions to be automatically performed without intervention.

B. Application Components

Table II describes the four application services (components) used to implement RUSLE2's application stack. The Model $\mathcal{M}$ component hosts the model computation and web services using the Apache Tomcat application server. The Database $\mathcal{D}$ component hosts the geospatial database which resolves latitude and longitude coordinates to assist in parameterizing climate, soil, and management data for RUSLE2. Postgresql was used as a relational database and PostGIS extensions were used to support geospatial functionality [11] [12]. The file server $\mathcal{F}$ component was used by the RUSLE2 model to acquire XML files to parameterize data for model runs. NGINX [13], a lightweight high performance web server provided access to a library of static XML files which were on average ~5KB each. The logging $\mathcal{L}$ component provided historical tracking of modeling activity. The codebeamer tracking facility supported by the Derby relational database was used to log model activity [14]. A simple JAX-RS RESTful JSON-based web service was developed to decouple logging requests from the RUSLE2 service calls. This service implemented an independent logging queue to prevent logging delays from interfering with RUSLE2 performance. HAProxy was used to redirect modeling requests from a public IP to potentially one or more backend $\mathcal{M}$ VMs. HAProxy is a dynamically configurable very fast load balancer which supports proxying both TCP and HTTP socket-based network traffic [15].

TABLE II. RUSLE2 APPLICATION COMONENTS

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, Apache Tomcat (32-bit) Custom RESTful JSON-based logging wrapper web service. Ia-32libs support operation in 64-bit environment.

C. Tested Service Compositions

RUSLE2's application stack of 4 components can be deployed 15 possible ways across 4 physical node computers. Tables III shows the 15 service compositions tested labeled as SC1-SC15. To achieve each of the compositions a single composite VM image was created with

all components installed ($\mathcal{M}$, $\mathcal{D}$, $\mathcal{F}$, $\mathcal{L}$). Four PMs were used to host one composite VM each. The testing script automatically enabled/disabled services as needed to achieve all service compositions (SC1-SC15).

TABLE III. TESTED SERVICE COMPOSITIONS

	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	$\mathcal{MLF}$	$\mathcal{D}$		

Every VM ran Ubuntu Linux 9.10 64-bit server and was configured with 8 virtual CPUs, 4 GB memory and 10GB of disk space. Drawbacks to our scripted testing approach include that our composite image had to be large enough to host all components, and for some compositions VM disks contained installed but non-running components. These drawbacks are not expected to be significantly relevant to performance.

D. Resource Utilization Statistics

Table IV describes the 18 resource utilization statistics collected using an automated profiling script. The profiling script parsed the Linux operating system `/proc/stat`, `/proc/diskstats`, `/proc/net/dev` and `/proc/loadavg` files. Initial resource utilization statistics were captured before execution of each ensemble test. After ensemble tests completed resource utilization statistics were captured and deltas calculated representing the resources expended throughout the duration of the ensemble test's execution. This data was recorded to a series of output files and uploaded to the dedicated blade server performing the testing. The same resource utilization statistics were captured for both VMs and PMs, but 8 statistics were found to have a negligible value for PMs. Resource utilization statistics collected for PMs are designated with "P", and for VMs with "V" in the table. Some statistics collected are likely redundant in that they are different representations of the same system properties. Subtleties in how related statistics are collected and expressed may provide performance modeling benefits and were captured for completeness is this study.

Performance models were built to predict ensemble execution time for different service compositions of RUSLE2. Using estimated average ensemble execution times for service composition rank predictions were made. Accurate performance rank predictions can be used to identify ideal compositions of components to support autonomic component deployment.

TABLE IV. RESOURCE UTILIZATION STATISTICS

Statistic		Description
P/V	CPU time	CPU time in ms
P/V	cpu usr	CPU time in user mode in ms
P/V	cpu krn	CPU time in kernel mode in ms
P/V	cpu_idle	CPU idle time in ms
P/V	contextsw	Number of context switches
P/V	cpu_io_wait	CPU time waiting for I/O to complete
P/V	cpu_sint_time	CPU time servicing soft interrupts
V	dsr	Disk sector reads (1 sector = 512 bytes)
V	dsreads	Number of completed disk reads
V	drm	Number of adjacent disk reads merged
V	readtime	Time in ms spent reading from disk
V	dsw	Disk sector writes (1 sector = 512 bytes)
V	dswrites	Number of completed disk writes
V	dwm	Number of adjacent disk writes merged
V	writetime	Time in ms spent writing to disk
P/V	nbr	Network bytes sent
P/V	nbs	Network bytes received
P/V	loadavg	Avg # of running processes in last 60 sec

E. Application Variants

Our investigation tested two variants of RUSLE2 which we refer to herein as the "d-bound" for the database bound and the "m-bound" for the model bound application. By testing two variants of RUSLE2 we hoped to gain insight into performance modeling by using two versions of RUSLE2 with different resource utilization profiles. For the d-bound RUSLE2, two primary geospatial queries were modified to perform a join on a nested query (as opposed to a table). The m-bound RUSLE2's geospatial queries used the ordinary table joins. The SC1 "d-bound" deployment required on average 104% more CPU time and 17,962% more disk sector reads (dsr) than the "m-bound" model. This modification significantly increased database CPU time and disk reads. Average ensemble execution time for all service compositions was approximately ~29.3 seconds for the m-bound model, and 4.7x greater at ~137.2 seconds for the d-bound model.

V. EXPERIMENTAL RESULTS

Table V summarizes tests completed for this study totaling approximately 300,000 model runs in 3,000 ensemble tests. The effectiveness of using the resource utilization statistics from table IV as independent variables to predict service composition performance (RQ1) are presented in section 5.1. Section 5.2 discusses experimental results which investigate how to best compose resource utilization statistics for use in performance models (RQ2). Section 5.3 concludes by presenting results of performance model effectiveness for predicting ensemble

execution time and service composition performance ranks for different application component compositions (RQ3).

TABLE V. SUMMARY OF TESTS

Model	Trials	Ensembles /Trial	Service Comps.	Model Runs	Ens. Runs
d-bound	2	20	15	60k	600
m-bound	3	20	15	90k	900
m-bound	1	100	15	150k	1,500
Totals	**6**			**300,000**	**3,000**

A. Independent Variables

This study investigated the utility of resource utilization statistics describing CPU utilization, disk I/O, and network I/O of both VMs and PMs for performance modeling as described in table IV. To investigate the predictive strength of each independent variable we performed separate linear regressions for each independent variable to predict ensemble execution time. R^2 is a measure of model quality which describes the percentage of variance explained by the model's independent variable. Adjusted R^2 is reported opposed to multiple R^2 because adjusted R^2 is more conservative as it includes an adjustment which takes into account the number of predictors in the model [23]. Statistics reported in table VI used 20 ensemble runs each for the 15 service compositions for both the "m-bound" and "d-bound" models. Untested statistics are indicated by "n/a". In these cases resource utilization was typically zero. Total resource utilization statistics were calculated by totaling values from VMs and PMs used in the service compositions.

CPU time is shown to predict the most variance for both models. Large differences in R^2 for "d-bound" compared to "m-bound" are shown in bold. For the "d-bound" model dsr and dsreads were less useful predictors, while contextsw and cpu_idle were shown to be better predictors. PM resource utilization statistics were generally found to be less useful as indicated by total R^2 values. No single PM statistic for the "m-bound" model achieved better than R^2=.086, while PM statistics for the "d-bound" model appeared better but not great with nbs as the strongest predictor at R^2=.3385.

Besides having strong R^2 values, good predictor variables for use in MLRs should have normally distributed data. To test normality of our resource utilization statistics the Shapiro-Wilk normality test was used [24]. 100 ensemble runs were made for each of the 15 service compositions for the "m-bound" model. Combining service composition data together was shown to decrease normality. Normality tests showed an average of 9 resource utilization statistics had normal distributions for individual compositions. When data for compositions was combined only loadavg, cpu_sint_time, and cpu_krn had strong normal distributions for the "m-bound" model and loadavg, CPU time, cpu_usr, and cpu_krn for the "d-bound" model. Ensemble time appeared to be normally distributed for both applications, but appeared more strongly normally distributed for "d-bound". Histogram plots for CPU time and dsr are shown in figure 1. CPU time and other related CPU time statistics (cpu_usr, cpu_krn) were among the strongest predictors of ensemble execution time for both models. Dsr was a better predictor for "m-bound" and its distribution appears more normal than for "d-bound". The plots visually confirm results of the Shapiro-Wilk normality tests.

TABLE VI. INDEPENDENT VARIABLE STRENGTH

Statistic	Adjusted R^2 "m-bound"		Adjusted R^2 "d-bound"	
	VM	*PM*	*VM*	*PM*
CPU time	0.7162	-0.0033	0.5096	0.1406
cpu usr	0.7006	-0.0019	0.444	0.04437
dsr	0.3693	n/a	**0.02613**	n/a
dsreads	0.3129	n/a	**0.02606**	n/a
cpu krn	0.1814	n/a	0.2958	0.2221
dswrites	0.1705	n/a	0.1151	n/a
dsw	0.1412	n/a	0.02292	n/a
dwm	0.1374	n/a	0.01528	n/a
contextsw	0.0618	-0.001	**0.4592**	0.1775
cpu_io_wait	0.0514	0.086	0.02528	0.05718
writetime	0.0451	n/a	-0.001199	n/a
loadavg	0.0168	0.0132	0.04321	0.004962
cpu_sint_time	0.0112	0.0141	0.02251	0.00003713
readtime	0.0094	n/a	0.02753	n/a
nbs	0.0042	0.0039	0.01852	0.3385
nbr	0.0041	n/a	0.01858	0.3368
cpu_idle	0.004	-0.0001	**0.2468**	0.2542
drm	0.0005	n/a	0.0261	n/a
Total R^2	2.938	0.1109	2.341	1.576

B. Treatment of Resource Utilization Data

The RUSLE2 application's 4 components ($\mathcal{M}$, $\mathcal{D}$, $\mathcal{F}$, $\mathcal{L}$) were distributed across 1 to 4 VMs. Resource utilization statistics were collected at the VM and PM level. Two treatments of the data are possible. Resource utilization statistics can be combined for all VMs and used to model performance: $RU_{data}=RU_{\mathcal{M}}+RU_{\mathcal{D}}+RU_{\mathcal{F}}+RU_{\mathcal{L}}$, or only resource utilization statistics for the VM hosting a particular component can be used to model performance: $RU_{data}=\{RU_{\mathcal{M}}; RU_{\mathcal{D}}; RU_{\mathcal{F}}; RU_{\mathcal{L}};\}$ To test the utility of both data handling approaches 10 MLR models were generated. A separate training and test data set were collected using 20 ensemble runs for each of the 15 service compositions for both the "m-bound" and "d-bound" RUSLE2. Results of the MLR models are summarized in table VII.

For the models described in table VII VM data (not PM) for all 18 independent variables was used. Adjusted R^2 values describe the variance explained by the models. The root mean squared error (RMS) expresses the differences between the predicted and observed values and serves to provide a measure of model accuracy. A statistically significant model ($p<.05$) will predict 95% of ensemble execution times with less than +/- 2 RMS error from the actual values [25]. RMS_{train} describes error at predicting ensemble times for the training dataset and RMS_{test} describes error at predicting ensemble times using the test dataset. For each of the service

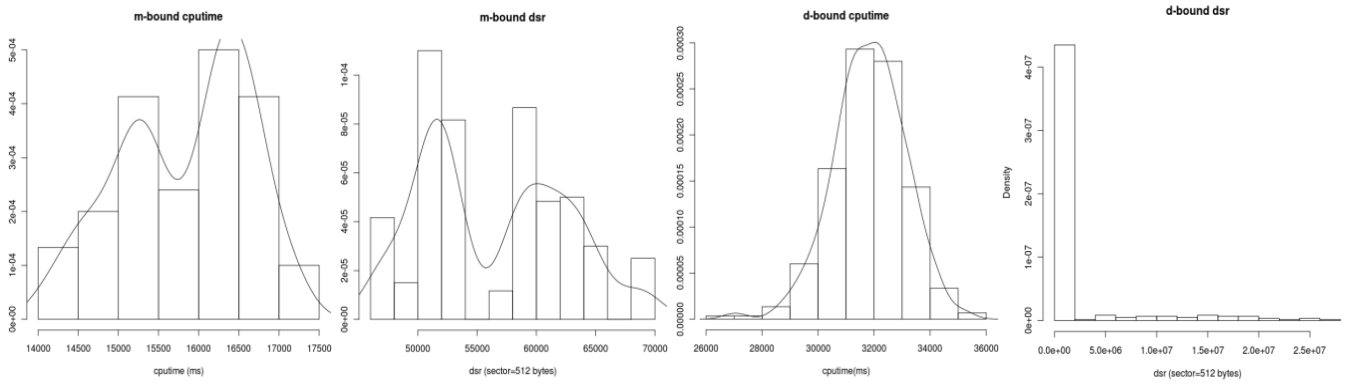

Figure 1. CPU time and Disk Sector Read Distribution Plots

TABLE VII. MULTIPLE LINEAR REGRESSION PERFORMANCE MODELS

Model	Data	Adj. R^2	RMS_{train}	RMS_{test}	Avg. Rank Error
d-bound	$RU_{\mathcal{M}}$	.9982	642.78	967.35	.13
d-bound	$RU_{\mathcal{D}}$	.9983	622.24	1248.24	.4
d-bound	$RU_{\mathcal{F}}$	.9984	615.64	606.94	.27
d-bound	$RU_{\mathcal{L}}$	.9983	621.99	978.92	.4
d-bound	$RU_{\mathcal{MDFL}}$	.9107	4532.85	44903.96	1.73
m-bound	$RU_{\mathcal{M}}$	.8733	576.05	759.36	1.47
m-bound	$RU_{\mathcal{D}}$	.67	929.54	971.85	2.13
m-bound	$RU_{\mathcal{F}}$	.7833	775.70	866.18	2
m-bound	$RU_{\mathcal{L}}$	.6247	991.29	42570.5	2.4
m-bound	$RU_{\mathcal{MDFL}}$	.8546	616.98	807.34	1.2

compositions an average estimate for ensemble execution time was calculated. The estimated average ensemble execution time was used to generate performance rank predictions for each of the 15 service compositions. The average rank error is the average error of actual vs. predicted ranks.

Analysis of model results shows that for the "d-bound" performance model, CPU idle time from individual VMs is an excellent predictor of ensemble execution time. R^2 for cpu_idle_time for the $\mathcal{M}$, $\mathcal{D}$, $\mathcal{F}$, $\mathcal{L}$ models is .7716, .7844, .6041, and .4223 respectively but only .2468 when combining VM statistics. This is in contrast to .0223, -.0024, .0271, .1199 and combined .0039 for the "m-bound" model. Further analysis reveals that the "d-bound" model makes 93.6x more disk sector reads than "m-bound" but only requires 2x as much CPU time while having 5.1x more idle CPU time. The "d-bound" model waits while this I/O is occurring making CPU idle time an excellent predictor for ensemble execution time for the "d-bound" application. The number of context switches for the busiest component seems to be a good predictor with $\mathcal{D}$ for "d-bound" at R^2=.4619 and $\mathcal{M}$ for the "m-bound" at R^2=.4786. The strength of using the number of context switches as a predictor of other VMs was less significant.

C. Performance Models

Combined resource utilization statistics ($RU_{\mathcal{MDFL}}$) were used as training data for 4 modeling approaches: MLR, stepwise multiple linear regression (MLR-step), MARS, and a simple single hidden layer ANN [24]. We investigated both MLR and stepwise MLR. MLR models use every independent variable provided to predict the dependent variable. Stepwise MLR begins by modeling the dependent variable using the complete set of independent variables but after each step adds or drops predictors based on their significance to test various combinations until the best model is found which explains the most variance (R^2). MARS is an adaptive extension of MLR which works by splitting independent variables into multiple basis functions and then fits a linear regression model to those basis functions [24]. Basis functions used by MARS are piecewise linear functions in the form of: f(x)={x-t if x>t, 0 otherwise} and g(x)=(t-x if x<t, 0 otherwise}. Both stepwise MLR and MARS were chosen because they generally provide some small improvement over traditional MLR and were easy to implement in R. ANNs are a very popular statistical modeling technique which excels at handling complex nonlinear relationships in the data. We tested single hidden layer ANN models supported by the R statistical software package [24] to predict ensemble execution times. R's ANNs use the sigmoid function, a bounded logistic function used to introduce nonlinearity in the model. A summary of performance models for the "m-bound" and "d-bound" application are shown in table VIII.

TABLE VIII. PERFORMANCE MODELS

Model	Type	Adj. R^2	RMS_{train}	RMS_{test}	Avg. Rank Error
d-bound	MLR	.9107	4532.85	44903.96	1.73
d-bound	MLR-step	.9118	4589.27	43918.55	1.73
d-bound	MARS	.9180	4472.32	45137.28	1.33
d-bound	ANN	n/a	4440.03	44094.03	1.6
m-bound	MLR	.8546	616.98	807.34	1.2
m-bound	MLR-step	.8571	621.41	799.22	1.33
m-bound	MARS	.8718	596.45	825.34	1.86
m-bound	ANN	n/a	595.49	800.71	1.73

R^2 values were not available for the ANN. For both applications, the ANN provided the lowest RMS error for the training dataset but slightly higher RMS error for the test dataset compared with stepwise MLR. For the 8 models RMS_{train} and RMS_{test} values correlated strongly (R^2=.999, $p=2.4\bullet10^{-10}$, df=6) suggesting that where a model performs

well on training data it will likely perform well on test data. There was no relationships between rank error and RMS_{test} (R^2=.02064, p=.734, df=6) suggesting that low error for ensemble time predictions does not guarantee low rank error. All of the models had some error at predicting service composition rank but provided functional predictions as they easily differentiated fast vs. slow service compositions and accurately determined the top 2 or 3 compositions.

VI. CONCLUSIONS

Modeling performance of service compositions of multi-tier applications deployed to IaaS clouds can help guide component composition for application deployments aiming to provide best performance with minimal virtual resources. Results of our exploratory investigation on performance modeling using resource utilization statistics for two variants of a multi-tier application include:

(RQ1) CPU time and other CPU related statistics were the strongest predictors of execution time, while disk and network I/O statistics were less useful. Measured disk and network I/O utilization statistics for our study suffered from non-normality and large variance when data from multiple service compositions were combined together for modeling purposes. CPU idle time and number of context switches were good predictors of execution time when the application's performance was I/O bound. Disk I/O statistics were better predictors when the application was more CPU bound.

(RQ2) The best treatment of resource utilization statistics for performance modeling, either combining data or using VM data separately, to achieve best model accuracy was dependent on each application's resource utilization profile.

(RQ3) Advanced modeling techniques such as MARS and ANN provided lower RMS_{error} for training and test data sets than MLR but overall all of the modeling approaches tested had similarly performance at minimizing RMS_{error}. Additionally all models determined the best 2 or 3 service compositions confirming the value of our performance modeling approach for determining ideal component compositions to support IaaS cloud multi-tier application deployment.

REFERENCES

[1] A. Kivity et al., "kvm: the Linux Virtual Machine Monitor," Proc. 2007 Ottawa Linux Symposium (OLS 2007), Ottawa, Canada, June 27-30, 2007, pp. 225-230.

[2] 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.

[3] 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.

[4] F. Camargos, G. Girard, B. Ligneris, "Virtualization of Linux servers," Proc. 2008 Linux Syposium, Ottawa, Ontario, Canada, July 23-26, 2008, pp. 63-76.

[5] 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.

[6] 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

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

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

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

[10] 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.

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

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

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

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

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

[16] 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.

[17] 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.

[18] O. Niehörster et al., "Autonomic Resource Management with Support Vector Machines," Proc. 12th IEEE/ACM Intl. Conf. on Grid Computing (GRID '11), Lyon, France, Sept 21-23, 2011, pp.157-164.

[19] 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, vol. 84, 2011, pp. 1270-1291.

[20] T. Wood et al., Sandpiper: Black-box and gray-box resource management for virtual machines, Computer Networks, vol. 53, 2009, pp. 2923-2938.

[21] W. Chen et al., Crossing and Nesting of Matching and Partitions, Transactions of the American Mathematical Society, vol. 359, No. 4, April 2007, pp. 1555-1575.

[22] O. David et al., A software engineering perspective on environmental modeling framework design: The Object Modeling System, Environmental Modelling & Software, Available 13 June 2012, http://www.sciencedirect.com/science/article/pii/S1364815212000886

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

[24] J. Adler, R In a Nutshell: A Desktop Quick Reference, First Edition, O'Reilly, 2010.

[25] P. Teetor, R Cookbook: Proven Recipes for Data Analysis, Statistics, and Graphics, First Edition, O'Reilly, 2011.

[26] W. Lloyd et al., 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.

[27] 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.

[28] 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.

[29] 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. 434-44.