Dissertation Proposal
Computer Science Department
Colorado State University

Autonomous Resource Management for Multi-tier Application Deployment to Infrastructure-as-a-Service (IaaS) Clouds

Wes J. Lloyd
wlloyd@cs.colostate.edu

Advisor: Dr. Shrideep Pallickara
shrideep@cs.colostate.edu

Summer 2012

Infrastructure-as-a-service (IaaS) clouds provide a new medium for deployment of multi-tier applications. By harnessing advancements in virtualization, IaaS clouds can provide dynamic scalable infrastructure which can be managed automatically, enabling applications to scale and load balance stressed components to better meet workload demands. To maximize performance while minimizing deployment costs, it is necessary to optimize physical location of virtual machines (VMs) which host application components while considering component dependencies that affect multiplexing of physical resources such as CPU time, disk and network bandwidth. Service isolation, hosting each application component using a separate VM has been suggested as a best practice, but our work has shown this deployment scheme can lead to increased deployment costs (total number of VMs) without providing performance benefits. This research proposal focuses on solving two primary problems of multi-tier application deployment to IaaS clouds: (1) automating the determination of good compositions of application components for deployment to VMs, and (2) autonomic management of virtual infrastructure to support varying application workloads. Five primary research questions are identified and five experiments are proposed and described supported by four development tasks in support of this research proposal.

1. Introduction

Cloud computing strives to provide computing as a utility to end users. Three service levels delineate cloud computing, with each offering additional infrastructure control to the end user. These include ***software-as-a-service (SaaS)***, where computational services such as computational libraries, modeling engines, and/or application programming interfaces (APIs) are hosted and made easily accessible to end users. ***Platform-as-a-service (PaaS)*** provides environment hosting allowing developers freedom to design and deploy services as long as they adhere to specific platform(s). PaaS provides specific relational databases, application servers, and vendor/platform specific programming APIs while abstracting, hosting, of the underlying infrastructure. Developers are freed from the burden of infrastructure management enabling them to focus on the design and development of application middleware. ***Infrastructure-as-a-service (IaaS)*** provides maximum freedom to developers enabling control of the middleware design, as well as the application infrastructure stack. Developers can freely choose databases, application servers, cache/logging servers as needed which are all hosted by virtual machines (VMs). IaaS enables diverse application stacks to be supported through the virtualization of various operating systems.

IaaS provides many benefits including easier application migration as existing applications can often be deployed as-is without extensive rearchitecting or redevelopment which may be required when deploying applications to PaaS clouds which provide vendor specific infrastructure. Legacy infrastructure can often be run under IaaS minimizing the need to rearchitect systems enabling a faster approach towards cloud migration. Avoiding lock-in to vendor specific application infrastructures and APIs can improve maintainability throughout an application's life-cycle as vendor specific APIs may incur special costs and have limited support if abandoned due to business reasons. IaaS clouds also allow granular scaling as individual components of the application can be scaled as needed to meet demand.

Multi-tier applications are decomposed into a stack of service-based components including web server(s), proxy server(s), application server(s), relational and/or NoSQL database(s), file server(s), distributed caches, log services and others. ***Service isolation*** involves separating the hosting of components of the application stack so they execute using separate VM instances. Isolation provides components explicit sandboxes, not shared by other systems. Using hardware ***virtualization***, isolation can be accomplished multiple times for separate components on a single physical server. Previously, service isolation using a physical data center required significant server real estate. Hardware virtualization refers to the creation and use of VMs which run on a physical host computer. Virtualization provides for resource ***elasticity*** where the quantity, location, and size of VM allocations can change dynamically to meet varying system loads, as well as increased agility to add and remove services as an application evolves. Service isolation, hardware virtualization, and resource elasticity are key

benefits motivating the adoption of IaaS based cloud-computing environments such as Amazon's Elastic Compute Cloud (EC2).

1.1. Problem

Given the advantages, IaaS clouds have become very attractive for hosting multi-tier applications. But how should applications be rearchitected and deployed to take advantage of the unique characteristics of IaaS? How can the cost of application hosting be minimized while maximizing application performance? Research to investigate these questions forms the basis of this research proposal.

1.2. Challenges

Many challenges ensue when deploying applications to IaaS clouds. Application components must be distributed using a series of VM images which are used to instantiate VMs, a concept know as ***component composition***. The number of images and the composition of components vary. Ideal multi-tier application compositions exhibit very good performance using a minimum number of images/VMs. These component aggregations exhibit performance based on how well resource conflicts can be minimized.

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 [52]. If considering an application as a set of (n) components, then the total number of possible component compositions is Bell's number (k). Table 1 shows the first few bell numbers describing the possible component compositions for an application with (n) components. Beyond four components the number of compositions grows large such that brute force testing to benchmark performance becomes an unwieldy arduous task. Further complicating testing, public IaaS clouds typically do not provide the ability to control VM placements making it difficult, if not impossible, to deploy all possible placements. Exclusive use of **physical machines** (PMs) is sometimes available for public IaaS clouds for an additional cost which would allow the needed level of control.

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

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

Table 1

Number of Multi-tier Application Component Compositions

Number of components (n)	Number of compositions (k)
3	5
4	15
5	52
6	203
7	877
8	4140

Provisioning variation, the variability of where application VMs are deployed across the PMs of a cloud, results in performance variation and degradation [2] [3] [4]. Unwanted multi-tenancy occurs when multiple VMs which intensively consume the same resource (CPU, Disk I/O, and Network I/O) reside on the same physical host computer leading to resource contention and performance degradation. Given an application with 4 components and 15 possible compositions provisioning variation yields 46 variations on how the 15 VMs can be deployed across the PMs. Both component composition and VM provisioning result in an explosion of the search space, making brute force testing of all possible deployments impractical.

Virtualization enables the resources of physical hardware to be partitioned for use by VMs. Memory is physically reserved and not shared, while CPU time, Disk I/O and network I/O are multiplexed and shared by all VMs running on a PM. The VM hypervisor either fully simulates physical devices using software, a practice known as full virtualization, or virtual device requests are passed directly to physical devices using para-virtualization. Full and para-virtualization of disk and network devices both incur overhead because these device operations incur additional management and in some cases full device simulation in addition to the actual I/O operations.

2. Prior Work

Placement of application components across a series of VM images can be envisioned as a bin packing problem. The traditional bin packing problem states that each bin has a size (V), and items (a_1,...,a_n) are packed into the bins. For our problem there are a minimum of five bins describing dimensions of: CPU utilization, disk write throughput, disk read throughput, network traffic sent, and network traffic received. To treat component composition as a bin packing problem both the resource capacities of our bins (PMs) as well as the resource consumption of our items (components) must be quantified. Determining resource utilization and capacities is

challenging particularly for stochastic applications and heterogeneous hardware where resource consumption and performance of resources varies.

Several approaches exist for autonomic provisioning and configuration of VMs for IaaS clouds. Xu et al. have identified two classes of approaches in [30]: *multivariate optimization* (performance modeling), and *feedback control*. Multi-variate optimization approaches have a specific optimization objective, typically improving performance, which is achieved by developing performance models which consider multiple system variables. Feedback control approaches based on process control theory attempt to improve configurations by iteratively making changes and observing outcomes. Formal approaches to autonomic resource provisioning attempted include: *integer linear programming* [33] [43], *knowledge/case-based reasoning* [37] [38], and *constraint programming* [40]. Integer linear programming techniques attempt to optimize a linear objective function. Case based reasoning approaches store past experiences in a knowledge base for later retrieval which is used to solve future problems by applying past solutions or inferring new solutions from previous related problems. Constraint programming is a form of declarative programming which captures relations between variables as constraints which describe properties of possible solutions. Feedback control approaches have been built using reinforcement learning [30], support vector machines [11], neural networks [31] [32], and a fitness function [42]. Performance models have been built using regression techniques [29], artificial neural networks [13] [30], and support vector machines [11]. Hybrid approaches which combine the use of performance modeling with feedback control include: [11] [30] [31] [32].

Feedback control approaches apply control system theory to actively tune resources to best meet pre-stated service level agreements (SLAs). Feedback control systems do not determine optimal configurations as they often consider a smaller subset of the exploration space as they use actual system observations to train models. This may result in inefficient control response, particularly upon system initialization. Multivariate optimization approaches model system performance with larger or complete training data sets enabling a much larger portion of the exploration space to be considered. Performance models require initialization with training datasets which can be difficult and time consuming to collect. Models with inadequate training data sets may be inaccurate and ineffective for providing resource control. Time to collect and analyze training datasets results in a trade-off between *model accuracy vs. availability*. Additionally performance models with a large number of independent variables or a sufficiently large exploration space exhibit the *accuracy vs. complexity* trade-off. Difficulty of collecting training data for models with a large search space, and a large number of variables leads to increased model development effort possibly forcing a tradeoff with model accuracy to keep model building tractable. Hybrid autonomic resource provisioning approaches combine the use of performance models with feedback control system approaches with an aim to provide better control decisions more rapidly. These systems use training datasets to inform

control decisions immediately upon initialization which are further improved as the system operates and more data is collected. Hybrid approaches often use simplified performance models which trade-off accuracy for speed of computation and initialization.

Wood et al. developed Sandpiper, a black-box and gray-box resource manager for VMs [29]. Sandpiper was designed to oversee server partitioning and was not designed specifically for IaaS. "Hotspots" are detected when provisioned architecture fails to meet service demand. Their approach was limited to vertical scaling which includes increasing available resources to VMs, and VM migration to a less busy PMs as needed. They did not perform horizontal scaling by launching additional VMs and load balancing. Sandpiper acts as a control system which attempts to meet a predetermined SLA. Xu et al. developed a resource learning approach for autonomic infrastructure management [30]. Both application agents and VM agents were used to monitor performance. A state/action table was built to record performance quality changes resulting from state/action events. A neural network model was later added to predict reward values to help improve performance after 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 [13]. From their observations they developed both a regression model and an artificial neural network to model performance. Their approach did not perform resource control, but focused on performance modeling to predict the performance implications of task placements. Niehorster et al. developed an autonomic resource provisioning system using support vector machines (SVMs) for IaaS clouds [11]. 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.

A number of formal approaches for autonomic resource management appear in the literature and commonly they've been built and tested with simulations only and not tested with physical clouds. Lama and Zhou proposed the use of self-adaptive neural network based fuzzy controllers in [31] [32] and was limited to controlling the number of VMs. Addis et al. model resource control as a mixed integer non-linear programming problem and apply two main features of classical local search exploration and refinement in [33]. Maurer et al. propose using a knowledge management system which uses case based reasoning to minimize SLA violations, achieve high resource utilization, conserve time and energy and minimize resource reallocation (migration) [37] [38]. Van et al. treat virtual resource management as a constraint classification problem and employ the choco constraint solver [40]. Their approach is model agnostic as individual applications must provide their own performance model(s). Li et al. propose a linear integer programming model for scheduling and migration of VMs for multi-cloud environments which considers the costs of VM migration and cloud provider pricing [43]. Bonvin et al. propose a virtual economy which considers the economic fitness of the utility provided by various application component deployments to cloud infrastructure [42]. Server

agents implement the economic model on each cloud node to help ensure fault tolerance and adherence to SLAs. Unlike above mentioned approaches Bonvin et al. evaluated their approach using applications running on physical servers, but their approach did not consider server virtualization but simply managed the allocation/deallocation of services across a cluster of physical servers.

2.1. Research Gaps

Table 2 provides a comparison of autonomic infrastructure management approaches described in [11] [13] [29] [30]. These approaches are compared because of the similarity to our proposed approach(es) described later in this research proposal. Each of the reviewed approaches has built a performance model and validated it benchmarking physical hardware in contrast to theoretical approaches which were validated using only simulation [31] [32] [33] [37] [38] [40] [43]. The complexity of cloud based systems makes validation using only simple economics-like simulations of questionable value. Table 2 shows features modeled, controlled, and/or considered by each of the approaches. Analyzing these approaches helps identify gaps in existing research. None of the approaches reviewed address composition of application components, as components were always deployed separately using full VM service isolation. Service isolation enables easier horizontal scaling of resources for specific application components but requires the largest number of VMs and may not provide better performance versus more concise deployments [45] [46]. Several issues were considered by only one approach including: horizontal scaling of VMs, tuning application specific parameters, determination of optimal configurations, and live VM migration. None of the approaches in Table 2 consider many independent variables in performance models and generally focus on a few select variables purported as the crux of their research contributions while ignoring implications of other variables. Approaches reviewed did not consider performance implications of virtualization hypervisor type (XEN, KVM, etc.) or disk I/O throughput, and only one approach considered implications of network I/O throughput and VM placement across PMs. Learning approaches which tend towards the use of simplified performance models may fail to capture the cause of improved performance when too many variables have been omitted from models. Other issues not addressed include the use of heterogeneous environments when PMs have varying capabilities, resource contention from interference between application components, and interference from non-application VMs.

Research should establish the most important variables for impacting application performance on IaaS clouds to form a base set of parameters for future performance models. Performance models may be further improved by the development and application of heuristics which capture performance behavior and characteristics specific to IaaS clouds. New performance models should be developed which better capture effects of virtualization,

component and VM location, and characteristics of physical resource sharing to support ideal load balancing of disk I/O, network I/O and CPU resources.

Table 2

Autonomic Infrastructure Management Feature Comparison

Feature controlled / modeled	Wood et al. [29]	Xu et al. [30]	Kousiouris et al. [13]	Niehorster et al. [11]
CPU scheduler credit		X	X	
VM memory allocation	X	X		X
Hypervisor type (KVM/XEN)				
Disk I/O Throughput				
Network I/O Throughput	X			
Location of application VMs			X	
Service composition of VM images				
Scaling # of VMs per application component				X
Application specific parameters				X
SLA/SLO enforcement	X	X		X
Determine optimal configuration			X	
Live VM migration	X			
Live tuning of VM configuration	X	X		X
# of CPU cores	X	X		X
memory	X	X		X
Performance modeling		X	X	X
Multi-tier application support	X			X

Existing gaps in research result from the infancy of cloud computing and the difficulty of performance modeling while resulting from a large problem space(s) with many potential independent variables. Benchmarking application resource utilization is difficult as isolating resource utilization data using public clouds with heterogeneous PMs which may potentially host multiple unrelated applications is difficult. Collecting resource utilization data involves overhead and can skew the accuracy of the data. Use of isolated testing environments such as private IaaS clouds can support testing but private IaaS virtual infrastructure management (VIM) software is still evolving and presently exhibits variable performance, incomplete feature sets, and inconsistent product stability [44]. Virtualization further complicates IaaS research, in

that the effects of virtualization are often misunderstood as the underlying implementation of virtualization hypervisors are often misunderstood.

3. Research Goals

This research proposal proposes to develop and improve autonomic resource provisioning and infrastructure management to better support hosting multi-tier applications using IaaS clouds. The aim is to develop heuristics which capture system behavior and devise a hybrid approach which combines performance modeling and feedback control. The focus is on two separate problems supporting deployment of multi-tier applications to IaaS clouds:

Problem 1:
Support autonomic determination of application component compositions to support multi-tier application deployment to VMs

Problem 2:
Automatically scale and configure virtual infrastructure to meet varying application service demand

For problem 1, approach(es) will be devised which automatically determine composition(s) of components that provide ideal application throughput. Rather than determine the optimal configuration, which may be infeasible in reasonable time for applications with too many components, our approach will combine performance modeling with heuristics as an alternative to brute force testing to determine good (but perhaps not optimal) component compositions. Once a composition has been chosen, VM images will be generated and an initial application deployment will be made using one VM instance per image. The initial application deployment (one VM per VM image) will support a base load. Problem 2 focuses on the research and development of a feedback control system to automatically scale infrastructure based on application service load. For both problems metrics will be developed and applied to evaluate quality and effectiveness of research system(s) and approach(es). Autonomic infrastructure management will strive to minimize resources including the number and size of VMs (memory, disk space, # processors), while maximizing application service throughput (requests serviced / time). Applications supported are assumed to be non-stochastic with relatively consistent performance behavior. These problems are related in that both require performance model(s) to be built and later applied to predict potential infrastructure management decisions. Performance models and lessons learned from investigation of research problem 1 are directly applicable to research problem 2.

Service request velocity is the number of requests per unit time:

Service request velocity= (# requests / time)

Service demand acceleration is the positive rate of change in service request velocity per unit time:

Service demand acceleration = (Δ Velocity / time)

Service demand deceleration is the negative rate of change in service request velocity per unit time.

Our aim is to develop approach(es) to autonomic resource provisioning which aim to (1) support high *service demand acceleration*, and (2) balance VM resource utilization in terms of CPU time, Disk I/O, and Network I/O. Good autonomic infrastructure management systems will support high service demand acceleration without dropping requests while completing each service request in less than a predetermined time to support a predetermined service level agreement (SLA). System(s) supporting high service demand acceleration can be said to have high responsiveness. Supporting a high service demand deceleration is generally not considered a challenge as deallocation of virtual resources is not time consuming relative to acquiring new virtual resources.

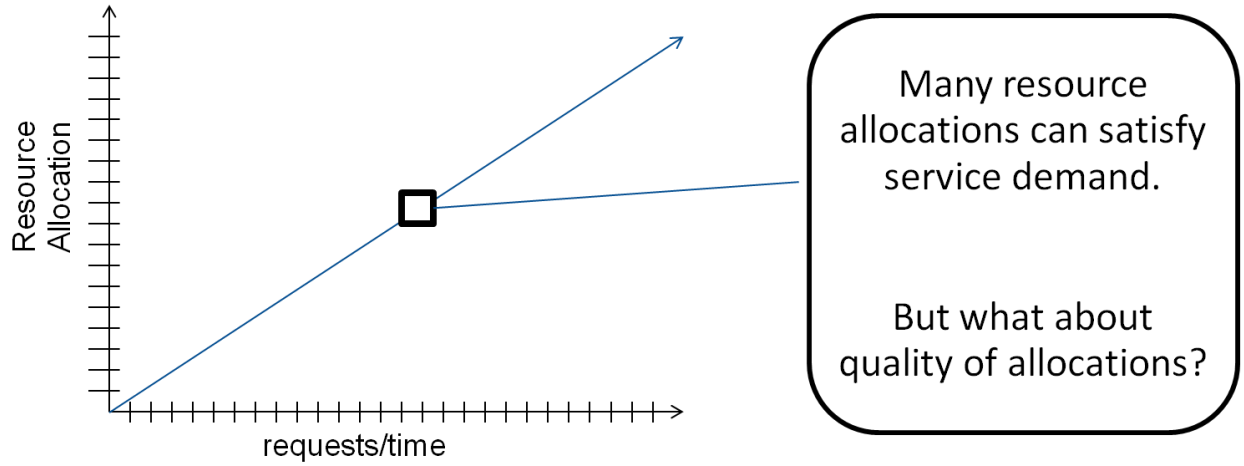


Figure 1 - Autonomic Resource Provisioning Quality

Many possible resource allocations meet service demand (requests/time). In Figure 1, the graph represents the minimum quantity of resources to meet service demand. Points above the line represent resource over-allocations incurring a higher cost, and points under the line represent under-allocations which have lower cost but fail to respond to all service

requests. **The goal is to identify resource allocations which exhibit the highest performance with the least number of resources (VMs and VM resources: memory, # cores, disk space).** The curve appears similar to a supply and demand curve from economics. The similarity between cloud-based resource provisioning and economics is recognized in [42]. In practice scaling resources rarely achieves linear performance improvement as shown in the graph because the law of diminishing returns predominates when scaling bottlenecks are not consistently surmounted.

4. Previous Findings

Previous findings from preliminary work have helped guide the development of this research proposal. Previously two variants of the Revised Universal Soil Loss Equation – Version 2 (RUSLE2), an erosion model were deployed as a cloud-based web service to a private IaaS cloud environment. RUSLE2 contains both empirical and process-based science that predicts rill and interrill soil erosion from rainfall and runoff. RUSLE2 was developed 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 migration because its architecture consisting of multiple components: an application server, geospatial/relational database, file server, logging server, and distributed cache serves as a good surrogate for multi-tier client/server applications.

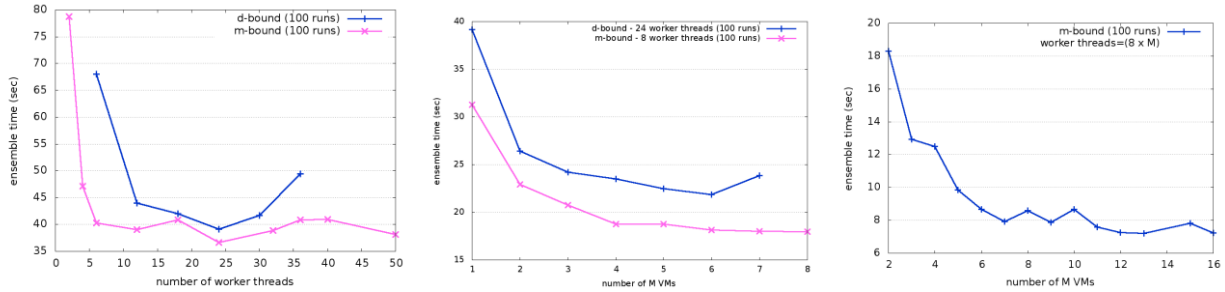
RUSLE2 was originally developed as a Windows-based Microsoft Visual C++ desktop application. To facilitate functioning as a web service a modeling engine known as the RomeShell was added to RUSLE2. The Object Modeling System 3.0 (OMS 3.0) framework using WINE [18] provides middleware to facilitate model to web service inter-operation. The RUSLE2 web service was implemented as a JAX-RS RESTful JSON-based service hosted by Apache Tomcat [16].

The two variants of RUSLE2 are referred to as “d-bound” for the database bound variant and “m-bound” for the model bound variant. The variants are named based on the predominant application component acting as the application's performance bottleneck consuming the largest quantity of CPU time. These variants represent two types of common multi-tier applications relevant in practice: an application bound by the database tier, and an application bound by the middleware (model) tier. For the “d-bound” version of RUSLE2 two primary spatial queries were modified to perform a join on a nested query, while the “m-bound” variant was unmodified. This modification results in the “d-bound” application having a different application profile than the “m-bound” model. On average the “d-bound” application requires about 2.45x more CPU time than the “m-bound” variant. The “m-bound” application’s

performance is constrained more by I/O performance while the "d-bound" application's performance is predominantly CPU-bound.

In [14] RUSLE2 was deployed using to a Eucalyptus private IaaS cloud consisting of nine PMs, eight acting as VM hosts. RUSLE2 deployment consisted of four primary components. Execution times were benchmarked for both 100 and 1,000 model run ensemble tests using a variety of infrastructure configurations. Scaling infrastructure to support computing an increasing number of simultaneous model runs was not as simple as increasing the number of VMs. While scaling infrastructure, a number of bottlenecks occurred which required tuning of application parameters to surmount. Each of the graphs in Figure 2 shows performance gains until a specific performance bottleneck occurs, which is surmounted in the following graph.

Figure 2 - Bottlenecks While Scaling Infrastructure



(a) (b) (c)

The first graph (a) depicts observed performance increases while scaling the number of worker threads. The second graph (b) depicts performance increases as the number of VMs performing modeling computations was increased. The final graph (c) shows performance increases after tuning the number of worker threads in response to an increased number of $\mathcal{M}$ VMs for the "m-bound" model only. In a related test for the "d-bound" model the number of shared database connections required tuning when performance was bound by nested database queries.

In [45], how application resource utilization profiles were affected based on component compositions across VMs, VM provisioning variation across PMs, VM memory size allocations, and hypervisor type KVM/XEN were investigated. To capture VM resource utilization statistics a profiling script was developed which captures CPU time, disk sector reads and writes (disk sector=512 bytes), and network bytes sent/received. RUSLE2 was deployed using four VM images hosting one application component each. VMs types are identified in table 3. The $\mathcal{M}$ component provides model computation and web services using Apache Tomcat. The $\mathcal{D}$ component implements the geospatial database which provides geospatial lookups for climate, soil, and management data to parameterize model runs. Postgresql was used as a relational database and PostGIS extensions were used to support geospatial database functions [17] [18].

The file server service $\mathcal{F}$ was used by the RUSLE2 model as a large cache of static XML files which parameterize model runs. NGINX [19], a lightweight high performance web server provided access to a repository of static files on average ~5KB each. The logging service $\mathcal{L}$ provided historical tracking of modeling activity. The Codebeamer tracking facility was used to log model activity [20]. Codebeamer provides an extensive customizable GUI and reporting facility. A simple JAX-RS RESTful JSON-based web service was developed to encapsulate logging functions to decouple Codebeamer from the RUSLE2 web service and also to provide a logging queue to prevent logging delays from interfering with the RUSLE2 web service. Codebeamer used the Apache Tomcat web application server and Derby a non-network relational database. Codebeamer, is a 32-bit application and required the use of the Linux 32-bit compatibility libraries (ia32-libs) to run on 64-bit VMs. A physical server running the HAProxy load balancer implements the application's entry point and redirects modeling requests to the VM hosting the modeling engine. HAProxy is a dynamically configurable very fast load balancer which supports proxying both TCP and HTTP socket-based network traffic [21].

Brute force testing was completed of every possible component composition for the RUSLE2 application as shown in Table 4. In all tests each PM hosted only one VM. Figure 3 shows the variation in resource utilization profiles for the service compositions. For the 15 service compositions, resource utilization data for all components was totaled: CPU time, Disk sector reads, Disk sector writes, network bytes received, and network bytes sent. Data was normalized to the average resource consumption for each resource. The graph shows the absolute value of the deviation from average for each service composition. Larger boxes indicate a greater deviation from average resource utilization and smaller boxes indicate compositions which perform close to the average. The graph does not express positive/negative deviation from average but the magnitude of deviation.

Table 3

RUSLE2 VMs

	VM	Description
$\mathcal{M}$	Model	Apache Tomcat 6.0.20, Wine 1.0.1, RUSLE2, Object Modeling System (OMS 3.0)
$\mathcal{D}$	Database	Postgresql-8.4.7, and PostGIS 1.4.0-2. Spatial 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
$\mathcal{F}$	Fileserver	nginx 0.7.62 to serve XML files which parameterize the RUSLE2 model. 57,185 XML files consisting of 305MB.
$\mathcal{L}$	Logger	Codebeamer 5.5 running on Apache Tomcat w/ Apache Derby. Custom RESTful JSON-based logging wrapper web service.

Figure 3 – Resource Utilization Variation for Service Composition

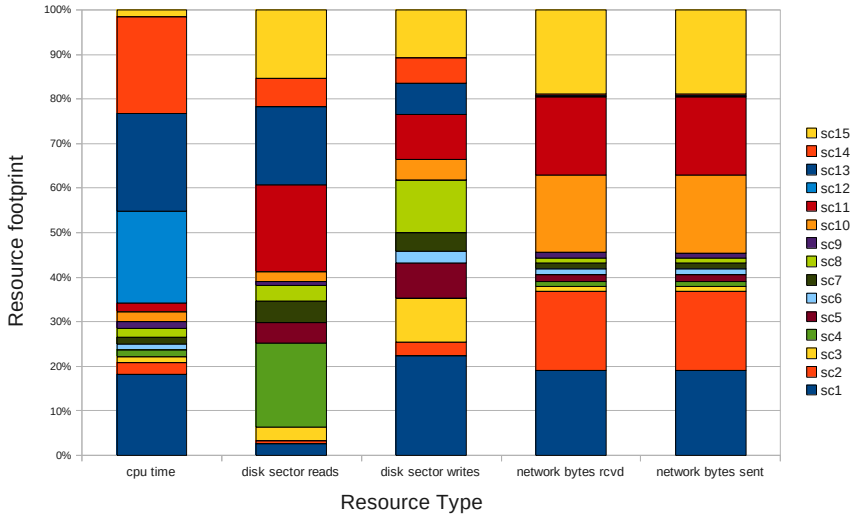


Table 4

VM Component Compositions

Config	PM 1	PM 2	PM 3	PM 4
SC1	$\mathcal{M}\,\mathcal{D}\,\mathcal{F}\,\mathcal{L}$			
SC2	$\mathcal{M}\,\mathcal{D}\,\mathcal{F}$	$\mathcal{L}$		
SC3	$\mathcal{M}\,\mathcal{D}$	$\mathcal{F}\,\mathcal{L}$		
SC4	$\mathcal{M}\,\mathcal{D}$	$\mathcal{F}$	$\mathcal{L}$	
SC5	$\mathcal{M}$	$\mathcal{D}\,\mathcal{F}\,\mathcal{L}$		
SC6	$\mathcal{M}$	$\mathcal{D}\,\mathcal{F}$	$\mathcal{L}$	
SC7	$\mathcal{M}$	$\mathcal{D}$	$\mathcal{F}$	$\mathcal{L}$
SC8	$\mathcal{M}$	$\mathcal{D}$	$\mathcal{F}\,\mathcal{L}$	
SC9	$\mathcal{M}$	$\mathcal{D}\,\mathcal{L}$	$\mathcal{F}$	
SC10	$\mathcal{M}\,\mathcal{F}$	$\mathcal{D}\,\mathcal{L}$		
SC11	$\mathcal{M}\,\mathcal{F}$	$\mathcal{D}$	$\mathcal{L}$	
SC12	$\mathcal{M}\,\mathcal{L}$	$\mathcal{D}\,\mathcal{F}$		
SC13	$\mathcal{M}\,\mathcal{L}$	$\mathcal{D}$	$\mathcal{F}$	
SC14	$\mathcal{M}\,\mathcal{D}\,\mathcal{L}$	$\mathcal{F}$		
SC15	$\mathcal{M}\,\mathcal{L}\,\mathcal{F}$	$\mathcal{D}$		

From component composition testing in [45] and [46] the top 3 performing compositions *SC2*, *SC6*, and *SC11* were identified. These compositions performed better than

isolating each application component on a separate physical node as in *SC7*. Testing was required to identify these top performing compositions as intuition was insufficient. The best compositions appear to isolate the $\mathcal{L}$ VM while keeping the $\mathcal{D}$ and $\mathcal{F}$ VM co-located, but $\mathcal{M}$ and $\mathcal{D}$ were separate only sometimes. Altering the VM memory size allocation was also found to result in changes to resource utilization footprints.

Service isolation, the practice of isolating each application component to run on a separate VM exhibited higher overhead leading to performance degradations for top performing configurations as shown in figure 4. The graphs show the percentage change in performance resulting from service isolation for the *SC2*, *SC6*, and *SC11* configurations. In most cases service isolation decreased model performance, and in all cases service isolation increased hosting costs requiring 4 VMs versus 2 VMs for *SC2*, and 3 VMs for *SC6* and *SC11*. **Industry has suggested service isolation as a best practice for deploying multi-tier applications to IaaS clouds which appears contrary to our results in terms of achieving both best performance and lowered hosting costs.**

Additional provisioning variation testing was conducted to complete testing of all possible methods to deploy the application in terms of component and VM location. Table 5 describes these component composition and VM deployment configurations. This testing validated if component composition testing using only one component per VM (table 4) was sufficient to find the best configurations. In total 46 configurations of the 15 service compositions from Table 4 were tested. Test results are shown in Figure 5. The graph shows the performance difference from average in milliseconds for each configuration. The *SC2*, *SC6*, *SC11* configurations and their provisioning variants were the best performing compositions. *SC6A* and *SC11A* performed better than equivalents *SC6* and *SC11* which used one VM to host each application component. Testing component composition performance with VMs deployed to isolated PMs was sufficient to find the best compositions, and co-locating VMs provided further performance improvements. **These results demonstrate that it is not necessary to exhaustively test all provisioning variations of component compositions to determine the best compositions.**

Figure 4 – Service Isolation Overhead

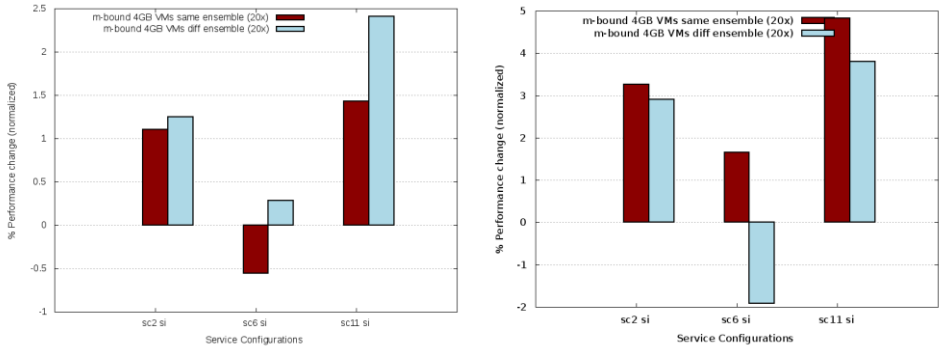
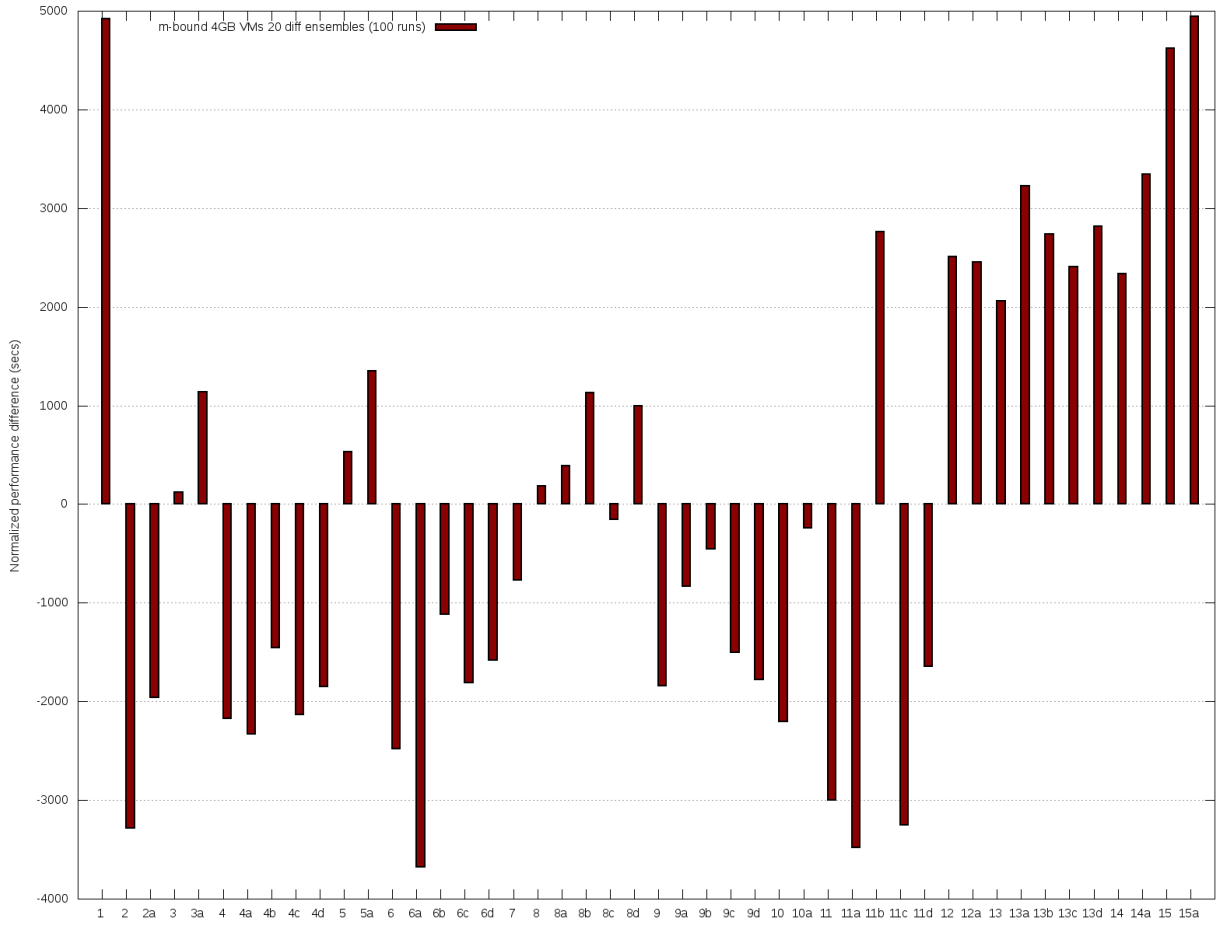


Table 5

Provisioning Variation VM Tests

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

Figure 5 – Provisioning Variation Performance Variation



5. Research Questions

The following research questions will be investigated:

RQ1. Which independent variables best help model application performance (throughput) to guide autonomic service composition?

To improve performance modeling of multi-tier applications deployed to IaaS clouds the utility of a number of independent variables not previously studied will be investigated. Variables describing resource utilization including statistics describing system load, CPU time, disk I/O and network I/O will be investigated. Both VM and PM statistics will be studied. Other independent variables to be considered include application component location relative to other application components, application VM location relative to other application VMs

(where VMs host one or more application components), VM configuration parameters including memory size allocation, # of CPU cores, and hypervisor type (KVM, XEN). The goal is to determine the most important independent variables for performance modeling to support building better performance model(s) for predicting performance of different possible configurations. Additionally multiple approaches to multivariate analysis will be investigated including but not limited to multiple linear regression (MLR), multivariate adaptive regression splines (MARS), and artificial neural networks (ANNs).

RQ2. Can VM resource classifications and behavioral rules aid performance predictions to help determine good service compositions?

A resource utilization classification scheme will be devised to classify application component resource utilization for CPU time, disk I/O, and network I/O. Application components will be benchmarked in isolation using separate VMs/PMs to determine their resource requirements for various service loads. These classifications will then be used to determine service compositions which balance PM resource utilization. The utility of using component resource classifications as a heuristic to improve component compositions will be investigated. A scheme will be developed which characterizes component behavior and dependencies based on assessing network traffic of components deployed in physical isolation. The existence and quantity of traffic flowing between components will be captured to express both the existence and importance of the component dependencies for the application. Component interactions (network traffic) will be expressed as behavioral rules for the application. The utility of using these behavioral rules as heuristics to improve component compositions will then be investigated.

The ultimate goal of developing and investigating both of these heuristics is to support the development of a component composition approach which determines good service compositions without requiring brute force testing because for large applications with many components exhaustive testing of all possible compositions is impractical.

RQ3. How rapidly can VMs be launched in response to service demand acceleration?

Before developing an autonomic resource provisioning system the scaling limitations imposed by the underlying virtualization and IaaS cloud middleware must be determined. How quickly application VMs can be launched should be quantified and workarounds developed to support higher service demand acceleration as needed. One potential workaround is to prelaunch VMs and suspend or pause them as needed. A tradeoff with pre-launching surplus VMs is higher hosting costs in exchange for greater responsiveness to demand acceleration.

RQ4. Does performance of service compositions change when scaled up?

Do the performance characteristics of service compositions remain the same when the number of VMs is scaled up? If only some VMs of an application are scaled up, how does this impact performance? Does the physical placement location of these additional VMs matter? A possible scenario which may degrade performance could be when multiple instances of a particular application VM are launched but all assigned to the same PM. This scenario could result in unwanted resource contention since each instance of the VM depends on the same resources and they have all been deployed to the same PM. Rules and policies may be needed to control the physical placement of VMs based on resource profiles as application VMs are scaled up to meet increased service demand.

RQ5. Which independent variables are most important for modeling application performance (throughput) in support of autonomic resource scaling?

An extension to RQ1, this research will investigate which independent variables are most important for modeling application performance when virtual infrastructure is scaled up. Independent variables to be considered include: Number of VMs (1 to n) per application VM, VM RAM allocation, VM core allocation, Location of VMs, and application specific parameters including: number of worker threads, number of database connections, and number of supported application server connections. The goal is to build a performance model capable of predicting performance without exhaustive testing of all possible infrastructure configurations.

5.1. Research Approach

To support this research the Cloud Services Innovation Platform (CSIP) has been developed in cooperation with the US Department of Agriculture - Natural Resources Conservation Service (USDA-NRCS) and the US Department of Agriculture – Agricultural Research Service (USDA-ARS). CSIP provides a generic modeling engine to support deployment of scientific models using IaaS Cloud based VMs to support multi-core parallel model computation. CSIP enables modeling-as-a-service by supporting execution of many simultaneous model runs in parallel for existing legacy models without re-architecting model code. The Object Modeling System version 3 framework within CSIP provides the basis for supporting parallel distributed modeling computation for individual computational steps within a model [16] [17]. 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. OMS provides numerous tools supporting data retrieval, GIS, graphical visualization,

statistical analysis and model calibration. Using a scalable pool of distributed worker VMs CSIP's architecture is positioned to extend beyond support of legacy models and support map-reduce style disaggregation of modeling computation [53]. Models with independent time and spatial stepping such as discrete event simulations and fully distributed watershed models can split computations based on discrete time and spatial units for later merging during a reduction phase. These models are poised as excellent candidates to harness the distributed parallel computation capability of distributed map-reduce.

CSIP supports individual model runs and ensemble runs which are groups of modeling requests bundled together. To invoke CSIP model services a client sends a JSON object including required parameters. Model results are computed and returned as a JSON object. Ensemble runs are processed by dividing groups of modeling requests into individual requests which are then resent to the web service, similar to the "map" function of MapReduce. A configurable number of worker threads concurrently executes individual runs of the ensemble, and upon completion results are combined (reduced) into a single JSON response object and returned.

To support CSIP two Eucalyptus 2.0 IaaS private clouds hosted by Colorado State University's Civil Engineering department have been implemented. Eucalyptus is an open source framework which provides an implementation of the IaaS architecture [25]. Eucalyptus supports two common cloud application programming interfaces (APIs) developed by Amazon, elastic compute cloud (EC2) and simple storage service (S3). EC2 is an API which enables management of virtual computing infrastructure. VMs can be launched, destroyed, modified, etc. as needed programmatically using the EC2 API. S3 is an API which supports a non-SQL, non-relational simple storage system acting as a cloud-based key-value data store. These private clouds were installed using the engineering college's local area network and were isolated to prevent outside access. One cloud consists of (9) SUN X6270 blade servers on the same chassis sharing a private 1 Giga-bit VLAN. Each blade server is equipped with dual Intel Xeon X5560-quad core 2.8 GHz CPUs, 24GB ram, and two 15000rpm hard disk drives of 145GB and 465GB capacity. A second cloud consists of a variety of surplus DELL Poweredge servers and commodity PCs. Both clouds employ a single server to host cloud services including the Eucalyptus cloud-controller (CC), cluster-controller (CLC), walrus (VM image) server, and storage-controller (SC). All other machines are configured as Eucalyptus node-controllers (NCs) to support hosting one or more VMs using either the XEN or KVM hypervisor [26] [27]. CentOS 5.7 Linux 64-bit is used as the host operating system for cloud nodes running the XEN hypervisor, and Ubuntu 10.10 Linux is the host operating system for cloud nodes running the KVM hypervisor. VM guests run Ubuntu Linux 64-bit server 9.10, 10.04, and 10.10. Planned upgrades to support future research include moving to Eucalyptus 3.1 and Ubuntu 12.04 which offers native dual hypervisor support (XEN and KVM) using the Linux 3.x kernel.

XEN supports para-virtualization which enables VMs to have nearly direct access to the host computer's physical disk and network devices to enable faster performance [5] [6] [26]. A disadvantage of XEN's para-virtualization is that all guest VMs must run a modified operating system kernel. The kernel-based virtual machine (KVM) hypervisor supports full virtualization of the underlying operating system allowing VMs to run any operating system without requiring special kernels. KVM has gained popularity with recent enhancements to Intel/AMD x86-based CPUs required for running KVM which provide special extensions to support full virtualization of guest operating systems without modification. These extensions allow device simulation overhead to be reduced to provide improved performance similar to XEN [10] [27].

To support research investigating deployment of multi-tier applications to cloud-based infrastructures several models developed and deployed using the CSIP architecture are available for study which serve as multi-tier application surrogates. These models include: the Revised Universal Soil Loss Equation – Version 2 (RUSLE2) [15], the Wind Erosion Prediction System (WEPS) [48], the Soil Tillage Intensity Rating (STIR), a sub-model of RUSLE2, and the Soil Conditioning Index (SCI) model [49]. Additionally one or more watershed models may be deployed using the CSIP infrastructure such as the Soil and Water Assessment Tool (SWAT) [50] and/or the AgroEcosystem-Watershed model (AgES-W) [51].

5.2. Research Experiments and Tasks

To investigate research questions detailed in section 5, the following tasks and experiments outlined in Table 6 are proposed. Experiment and tasks are described in detail below.

Table 6

Research Question Task and Experiments Relationships

Research Question	Experiments / Tasks
RQ1	Experiment 1
RQ2	Task 1, Task 2, Task 3, Experiment 2
RQ3	Experiment 3
RQ4	Experiment 4
RQ5	Task 4, Experiment 5

Experiment 1	Component Composition Exploratory Modeling
Goal	Investigate independent variables to determine their utility for performance modeling for singly provisioned component compositions. Performance model(s) should predict performance without exhaustively testing all possible component compositions. Emphasis will be on predicting performance throughput and ranking performance of possible component compositions.
Research Question	RQ1
Summary	Determine the best independent variables which help predict performance for a variety of service compositions based on benchmarks collected from testing application performance when each application component has been deployed in physical isolation. Variables: • Application profile resource utilization data: CPU time, disk I/O and network I/O • Resource utilization and load of VMs/PMs • VM placement location • VM configuration parameters: memory allocation, # of CPU cores, hypervisor type (KVM, XEN, …)
Metrics	Independent variables should have high R^2 values under multiple linear regression tests indicating their predictive value. Performance models should predict good service compositions determined using brute force testing.

Recently, initial work has been completed for Experiment 1 and a paper summarizing the results has been submitted to the 2012 IEEE/ACM Utility and Cloud Computing Conference (UCC) [47].

Task 1	Application Resource Classification
Goal	Develop scheme to classify magnitude of resource utilization for CPU time, disk I/O, and network I/O of application components deployed in isolation on separate VMs.
Research Question	RQ2
Summary	Develop classification scheme to express relative resource utilization for application components. Express resource utilization using numeric ratio scale normalized from 0 to 1, where 1 represents the maximum available resources as benchmarked for CPU time, disk I/O, network I/O for a typical VM (eg. 8 virtual cores, 4 GB ram, 10 GB disk hosted by one of our private clouds) An application component's resource utilization can be expressed as: CPU time = .725 Disk I/O = .212 Network I/O = .100

Task 2	Devise Scheme to Capture Application Behavioral Rules
Goal	Develop scheme to capture and express application component interactions. Scheme will provide behavioral guidelines to investigate independent variables to determine their utility for performance modeling for singly provisioned component compositions. Performance model(s) should predict performance without exhaustively testing all possible compositions.
Research Question	RQ2
Summary	Deploying all components in physical isolation will increase network traffic. This network traffic will be assessed and the dependencies and the magnitude of dependencies captured which occur between application components. Magnitude of dependencies for up-link and down-link communication are expressed using a numeric ratio scale normalized from 0 to 1, where 1 represents the maximum dependency possible (network saturation) between a component running on one VM interacting with a dependent component on another VM. The maximum is established using benchmarks to determine the maximum network throughput supported between 2 VMs. This approach assumes that dependencies between components will use TCP socket-based based communication for all interaction. Behavior could be stated as follows: A → B .115 B → C .35 C → A .511

Task 3	Autonomic Component Composition
Goal	Using the resource classification scheme and behavior rules developed in tasks 1 and 2, develop method(s) to automatically determine good VM image component compositions for multi-tier applications. Approach(es) should not rely on brute force exhaustive testing of all possible compositions but be informed by heuristics developed in tasks 1 & 2.
Research Question	RQ2
Summary	Develop an autonomic component composition scheme combining the resource utilization classification and behavioral rule heuristics of tasks 1 & 2 to provide a method for determining good service composition(s) without exhaustive testing of all possible compositions. Exhaustive testing may be used to validate approach efficacy.

Experiment 2	Validate Autonomic Service Composition
Goal	Validate component composition approaches developed by task 3. Quantify the quality of the compositions, and the composition effort required. Compare service composition approach(es) developed against other approaches including adhoc (random) composition, and approaches which exhaustively test all possible compositions (where feasible). Compare with other approaches from literature if available.
Research Question	RQ2
Summary	Evaluate quality of autonomic service composition approaches developed in task 3. Quantify the quality of compositions generated by the approach(es). Quantify the quality of approach(es) deriving the compositions.
Metrics	Service composition approach(es) are validated using the following metrics: Composition Speed: This is the average time required to determine component compositions. Good approaches will determine service compositions quickly. Resource Packing: This is the average number of VMs required for component compositions. Good approaches will minimize the number of required VMs for the application deployment. Composition Quality: This is the average service throughput of component compositions. Good compositions will support high service throughput. Computation Cost: This is the average quantity of CPU time, disk I/O, network I/O time required to make the component composition determinations. Good approaches will have the smallest computational footprints.

Experiment 3	Investigate Virtual Infrastructure Management Limitations
Goal	Benchmark performance of launching a variety of VM sizes
Research Question	RQ3
Summary	Benchmark the capabilities of Virtual Infrastructure Management (VIM) to determine the physical limitations for scaling. Determine how rapidly VMs can be launched based on their size and present load of PMs. Develop workarounds for performance bottlenecks as needed. Consider costs of prelaunching additional VMs and placing them in stand-by to rapidly respond to increased service demand.
Metrics	Benchmark VM launch times as a factor of VM disk size. Small (5 GB disk) Medium (10 GB disk) Large (20 GB disk)

Experiment 4	Service Composition Scaling Test
Goal	Determine if the best component compositions determined by task 3 remain the best compositions when scaled to multiple VMs per system component and subjected to high load.
Research Question	RQ4
Summary	Quantify performance of the best component compositions from task 3 to determine if their performance remains favorable versus other possible compositions when scaled up. Consider ramifications of the physical placement of additional application VMs. What is important to know about how the placement of these additional VMs impacts capacity? Check for unexpected behavior.
Metrics	Compare service response time and service throughput of different component compositions when scaled to multiple VMs.

Task 4	Develop Autonomic Resource Provisioning System
Goal	Develop an autonomic resource provisioning system
Research Question	RQ5
Summary	Develop a dynamic resource provisioning system which scales infrastructure resources based on service demand. The system requires hot spot detection to determine when allocated resources are insufficient to meet service demand. Hotspot detection can be based on detection of dropped service requests due to overloading. Hotspot detection schemes may also attempt to predict resource shortfalls before they occur based on analysis of usage patterns. Scaling and load balancing: Upon hotspot detection resources should be scaled up and load balancers reconfigured. The system will not use brute force testing to determine how to scale infrastructure but will use performance model(s) informed by heuristics developed in tasks 1 and 2. Two or more approaches to performance modeling will be attempted including a multiple linear regression model and an artificial neural network.

Experiment 5	Evaluate Autonomic Resource Provisioning System
Goal	Validate autonomic resource scaling approach(es)
Research Question	RQ4
Summary	This experiment will validate the effectiveness of the autonomic resource provisioning system. Where possible our approach will be contrasted with existing approaches.
Metrics	Service Demand Acceleration Responsiveness: Determine the maximum load acceleration supported without dropping requests. % of service requests completed: Determine the ratio of completed requests to total requests. Adaption time: Determine the time required to adapt (min:sec) infrastructure before there are no dropped requests for various load scenarios Recovery time: Determine the recovery time from failure (min:sec) before there are no dropped requests

5.3. Benchmarks / Evaluation Metrics

Metrics will be used to quantify the quality of component composition and autonomic resource provisioning schemes developed by this research. Time-based performance metrics will be used to quantify service response time and service throughput of multi-tier applications.

5.3.1. Service Performance Metrics

The time scale of service performance metrics will be dependent on the service of interest. The time scale will typically be in seconds or minutes. *Service response time* is the execution time required from when the service is invoked until it responds with a result. *Service throughput* is the number of service responses produced per unit of time. Service throughput will be calculated using ensemble tests which combine together service requests into a single request. The ensemble execution time is the time required to execute all service requests in the ensemble. Service throughput is computed by normalizing to a common time unit (e.g. seconds, minutes) the number of service requests completed. For this research the RUSLE2 and STIR models will express performance in model runs per second, and the WEPS and SCI models will express performance in model runs per minute.

5.3.2. Autonomic Resource Provisioning Quality Metrics

Responsiveness, will capture the maximum supported load acceleration supported by an autonomic resource provisioning system without dropping application service requests. When service request(s) are dropped this is indicative of a hot spot indicating additional resources are required to meet current demand. The *completion rate* captures the percentage of service requests completed relative to the total number of requests issued. The *adaption time* reflects the time required to adapt a multi-tier application's infrastructure in response to a change in service demand. The *recovery time* is the time required for the system to recover from the failure of one or more VMs.

5.3.3. Service Composition Quality Metrics

Composition speed is the computation time required to determine application component compositions used for deploying components across a set of VMs. *Resource packing density* measures the compactness of an application's component composition. Resource packing density is the ratio of the number of application components to the number of VMs used for application deployment (components : VMs). Packing density greater than 1 indicates that at least one VMs is multiplexed to host more than one application component. A very high packing density represents an efficient deployment in terms of hosting costs as the total number of VMs has been kept low. A packing density of 1 represents total service isolation. *Composition performance* quantifies the average service throughput obtained by an application component composition. *Average application service throughput* is established by executing a standardized set of tests which are reused throughout experimentation to benchmark service performance. The *composition footprint* measures an application's total resource utilization profile by totaling together all CPU time, disk I/O, and network I/O for all VMs of the component composition. Having a high or low composition footprint does not necessarily correlate with good application performance. Autonomic component composition schemes should aim to minimize the composition speed, maximize resource packing density, while maximizing service throughput.

6. Expected Contributions

This research proposal aims to research and investigate: (1) autonomic approach(es) for determining ideal component compositions to guide deployment of multi-tier application(s) to VMs, and (2) autonomic resource provisioning approach(es) to manage and scale infrastructure to meet service demand for hosting multi-tier applications using IaaS clouds. This research aims to develop performance model(s) of multi-tier applications hosted by virtualized infrastructure

which consider application resource utilization profiles, component compositions across VMs, and VM physical placements. The effectiveness of harnessing application profile data including various CPU time, disk I/O, and network I/O heuristics will be quantified for use in models which aim to predict performance of application deployments. Two primary heuristics will be developed and investigated to help improve autonomic service composition and resource provisioning. A *resource characterization heuristic* will be developed to quantify application component resource consumption of CPU time, disk I/O throughput, and network I/O throughput. The utility for supporting load balancing of cloud resources using this heuristic will be investigated. A *component behavior heuristic* will be developed and investigated which aims to capture and quantify the degree of component dependencies. Implications for aiding determination of ideal component composition(s), and performing autonomic infrastructure management using performance models aided by both heuristics will be investigated. Limitations of scaling virtual infrastructure will be quantified and workarounds investigated to support autonomic resource provisioning. Finally, metrics will be developed to evaluate both our autonomic service composition and resource provisioning approaches developed by this research. These metrics are considered research contributions as they are applicable to evaluating similar autonomic service composition and resource provisioning systems.

Presently multi-tier applications are often deployed to IaaS clouds using service isolation of application components by deploying components to separate VMs hosted in an ad hoc manner across PMs. When service isolation is not used, applications are deployed using adhoc deployments of components to VMs which may or may not provide ideal utilization of physical resources. IaaS clouds are not intelligent in how VMs are deployed and how application infrastructure is supported. This research proposes to develop approaches which (1) improve the initial composition of application components deployed to VMs, and (2) improve management of cloud infrastructure by considering application resource requirements and profile characteristics enabling better sharing and utilization of physical resources.

6.1. Related research

Autonomic service composition and resource provisioning for hosting multi-tier applications using IaaS clouds is a compound problem which crosscuts many existing areas of distributed systems research. Related research problems which are NOT the primary focus of this work are described below. These research problems can be as considered related research and potential future work, but are generally considered as non-goals of this work.

6.1.1. Heterogeneous physical infrastructure

This research does not focus on the implications of IaaS clouds consisting of heterogeneous physical infrastructure. Having physical hosts with unequal resource capacity adds another dimension to the resource provisioning problem. Support for heterogeneous hardware for PMs acting as VM hosts will be developed only as needed but is not a focus of this research.

6.1.2. Stochastic applications

This research focuses on autonomic service composition and resource provisioning to support hosting a single non-stochastic multi-tier application which exhibits stable resource utilization characteristics. The primary focus is to support application deployment and infrastructure management for service-based applications which provide modeling or computational engines as a service to end users. Applications with non-deterministic stochastic behavior are not the focus of this work.

6.1.3. External Interference

Public IaaS clouds and private IaaS clouds hosting multiple applications may experience interference when multiple applications simultaneously share the same PMs hosting application infrastructure. Interference from external applications, particularly stochastic applications, can cause unpredictable behavior. The research will focus on service composition and resource provisioning for a single multi-tier application. Future work could include extending our approach to support autonomic resource provisioning for multiple co-located non-stochastic applications. At the virtualization level, the ability for hypervisors to host multiple VMs while minimizing interference is an active research topic [8] [9] [10].

6.1.4. Fault tolerance

This research does not focus exclusively on fault tolerance of the virtual infrastructure hosting multi-tier applications. Fault tolerance is considered an autonomic resource provisioning system feature, but is not a primary focus of this research. Fault tolerance support and investigation of fault tolerance research questions related to autonomic resource provisioning systems is considered as future or related work.

6.1.5. Hot-spot detection

This research does not focus specifically on development of novel hot spot detection schemes. Hot spot detection scheme(s) are required for autonomic resource provisioning and appropriate methods will be chosen and developed as needed.

7. Summary

This proposal focuses on research and investigation of challenges related to developing autonomic component composition and resource provisioning systems to support deployment of multi-tier applications to IaaS clouds. Five primary research questions have been identified and five experiments supported by four development tasks have been detailed to support this research. At the conclusion of this research one or more approaches to the two primary problems will be developed and demonstrated using available multi-tier environmental science applications. This research will (1) develop performance models and heuristics which support prediction of ideal compositions of application components for deployment across VMs, and (2) develop performance models and heuristics which support autonomic resource provisioning to scale virtual infrastructure autonomously based on application service demand. This research aims to develop approaches for intelligent infrastructure management which improve existing IaaS cloud technology by moving IaaS virtual infrastructure management from simple management of pools of VMs, to intelligent systems which balance resource utilization of CPU, disk, and network resources to improve application performance and utilization of physical infrastructure.

References

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

[2] Rehman, M., Sakr, M., “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] Schad, J., Dittrich, J., Quiane-Ruiz, J., “Runtime Measurements in the Cloud: Observing, Analyzing, and Reducing Variance”, Proc. of 36th Intl. Conf. on Very Large DataBases (VLDB 2010), Singapore, China, Sept 13-17, 2010, pp. 460-471.

[4] Zaharia, M., Konwinski, A., Joesph, A., Katz, R., Stoica, I., “Improving MapReduce Performance in Heterogeneous Environments”, Proc. 8th USENIX Conf. Operating systems design and implementation (OSDI '08), San Diego, CA, USA, Dec 8-10, 2008, pp. 29-42.

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

[6] Armstrong, D., Djemame, K., “Performance Issues In Clouds: An Evaluation of Virtual Image Propagation and I/O Paravirtualization”, The Computer Journal, June 2011, vol. 54, iss. 6, pp. 836-849.

[7] Jayasinghe, D., Malkowski, S., Wang, Q., Li, J., Xiong, P., Pu, C. “Variations in Performance and Scalability when Migrating n-Tier Applications to Different Clouds”, Proc. 4th IEEE Intl. Conf on Cloud Computing (Cloud 2011), Washington D.C., USA, July, 2011, pp. 73-80.

[8] Matthews, J., Hu, W., Hapuarachchi, M., Deshane, T., Dimatos, D., Hamilton, G., McCabe, M., Owens, J., “Quantifying the Performance Isolation Properties of Virtualization Systems”, Proc. ACM Workshop on Experimental Computer Science (ExpCS ’07), New York, NY, USA, 2007, Article 6.

[9] Somani, G., Chaudhary, S., “Application Performance Isolation in Virtualization”, Proc. 2nd IEEE Intl. Conf on Cloud Computing (Cloud 2009), Bangalore, Indian, Sept 2009, pp. 41-48.

[10] Raj, H., Nathuji, R., Singh, A., England, P., “Resource Management for Isolation Enhanced Cloud Services”, Proc. ACM Cloud Computing Security Workshop (CCSW ’09), Chicago, IL, USA, Nov, 2009, pp. 77-84.

[11] Niehörster, O., Krieger, A., Simon, J., Brinkmann, A., “Autonomic Resource Management with Support Vector Machines”, Proc. 12th IEEE/ACM Intl. Conf. On Grid Computing (GRID 2011), Lyon, France, Sept 21-23, 2011, pp. 157-164.

[12] Tordsson, J., Montero, R., Moreno-Vozmediano, R., Llorente, I., “Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers”, Future Generation Computer Systems, vol. 28, 2012, pp. 358-367.

[13] Kousiouris, G., Cucinotta, T., Varvarigou, T., “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.

[14] Lloyd, W., Pallickara, S., David, O., Lyon, J., Arabi, M., Rojas, K., “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.

[15] 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

[16] Ahuja, L., Ascough, J., David, O., “Developing natural resource modeling using the object modeling system: feasibility and challenges”, Advances in Geosciences, vol. 4, 2005, pp. 29-36.

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

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

[19] Apache Tomcat – Welcome, 2011, http://tomcat.apache.org/
[20] PostGIS, 2011, http://postgis.refractions.net/
[21] PostgreSQL: The world's most advanced open source database, http://www.postgresql.org/
[22] nginx news, 2011, http://nginx.org/
[23] Welcome to CodeBeamer, 2011, https://codebeamer.com/ cb/user/
[24] HAProxy – The Reliable, High Performance TCP/HTTP Load Balancer, http://haproxy.1wt.eu/
[25] Nurmi, D. 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.
[26] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A., "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.
[27] Kivity, A., Kamay, Y., Laor, D., Lublin, U., Liguori, A., "kvm: the Linux Virtual Machine Monitor", Proc. 2007 Ottawa Linux Symposium (OLS 2007), Ottawa, Canada, June 27-30, 2007, pp. 225-230.
[28] Myers, R., "Classical and Modern Regression with Applications", 2nd Edition, PWS-KENT Publishing Company, Boston, MA, 1994.
[29] Wood, T., Shenoy, P., Venkataaramani, A., Yousif, M., "Sandpiper: Black-box and gray-box resource management for virtual machines", Computer Networks, vol. 53, 2009, pp. 2923-2938.
[30] Xu, C., Rao, J., Bu, X., "URL: A unified reinforcement learning approach for autonomic cloud management", Journal of Parallel and Distributed Computing, vol. 72, 2012, pp. 95-105.
[31] Lama, P., Zhou, X., "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.
[32] Lama, P., Zhou, X., "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.
[33] Addis, B., Ardagna, D., Panicucci, B., Milano, P., Zhang, L., "Autonomic Management of Cloud Service Centers with Availability Guarantees", Proc. 3rd IEEE Int. Conf. On Cloud Computing (CLOUD 2010), Miami FL, USA, July 5-10, 2010, pp. 220-227.
[34] Casale, G., Kraft, S., Krishnamurthy, D., "A Model of Storage I/O Performance Interference in Virtualized Systems", Proc. 31st Int. Conf. On Distributed Computing Systems Workshops (ICDCSW 2011), Minneapolis MN, USA, June 20-24, 2011, pp. 34-39.
[35] Casalicchio, E., Silvestri, L., "Architectures for Autonomic Service Management in Cloud-Based Systems", Proc. 16th IEEE Symposium on Computers and Communication (ISCC '11), Kerkyra, Greece, June 28-July 1, 2011, pp. 161-166.
[36] Voith, T., Oberle, K., Stein, M., "Quality of service provisioning for distributed data center inter-connectivity enabled by network virtualization", Future Generation Computer Systems, vol. 28, 2012, pp. 554-562.

[37] Maurer, M., Brandic, I., Sakellariou, R., "Enacting SLAs in Clouds Using Rules", Springer Lecture Notes in Computer Science, vol. 6852, p. I, 2011, pp. 455-466.
[38] Maurer, M., Brandic, I., Sakellariou, R., "Simulating Autonomic SLA Enactment in Clouds Using Case Based Reasoning", Lecture Notes in Computer Science, vol. 6481, 2010, pp. 25-36.
[39] Andreolini, M., Casolari, S., Colajanni, M., Messori, M., "Dynamic Load Management of Virtual Machines in Cloud Architectures", Proc. 1st Int. Conf. On Cloud Computing (ICST CLOUDCOMP 2009), Munich, Germany, Oct 19-21, 2009, 14 p.
[40] Van, H., Tran, F., Menaud, J., "Autonomic Virtual Resource Management for Service Hosting Platforms", Proc. IEEE Workshop on Software Engineering Challenges in Cloud Computing (ICSE CLOUD '09), Vancouver, B.C. Canada, May 23, 2009, 8 p.
[41] Iqbal, W., Dailey, M., Carrera, D., "SLA-Driven Dynamic Resource Management for Multi-tier Web Applications in a Cloud", Proc. 10th IEEE/ACM Int. Conf. On Cluster, Cloud, and Grid Computing (CCGRID 2010), Melbourne, Australia, May 17-20, 2010, pp. 832-837.
[42] Bonvin, N., Papaioannou, T., Aberer, K., "Autonomic SLA-driven Provisioning for Cloud Applications", Proc. IEEE/ACM Int. Symposium on Cluster, Cloud, and Grid Computing (CCGRID 2011), Newport Beach, CA, USA, 2011, pp. 434-443.
[43] Li, W., Tordsson, J., Elmroth, E., "Modeling for Dynamic Cloud Scheduling via Migration of Virtual Machines", Proc 3rd IEEE Int. Conf. On Cloud Computing Technology and Science (CloudCom '11), Athens, Greece, 2011, pp. 163-171.
[44] Sempolinski, P., Thain, D., "A Comparison and Critique of Eucalyptus, OpenNebula and Nimbus", Proc 2nd IEEE Int. Conf on Cloud Computing Technology and Science (CloudCom '10), Indianapolis, IN, USA, Nov 30 – Dec 3, 2010, pp. 417-426.
[45] Lloyd, W., Pallickara, S., David, O., Lyon, j., Arabi, M., Rojas, K., "Performance Implications of Multi-Tier Application Service Configurations on Infrastructure-as-a-Service Clouds", submitted to Elsevier Future Generation Computer Systems, Feb. 15 2012, under review.
[46] Lloyd, W., Pallickara, S., David, O., Lyon, j., Arabi, M., Rojas, K., "Service Isolation and Compositions for Multi-tier Application Deployment: An Investigation on Implications for Infrastructure-as-a-Service Clouds", submitted to ACM Symp. on Cloud Computing (SOCC 2012), Oct. 14-17, 2012.
[47] Lloyd, W., Pallickara, S., David, O., Lyon, j., Arabi, M., Rojas, K., "Performance Modeling to Support Multi-Tier Application Deployment to Infrastructure-as-a-Service Clouds", submitted to IEEE/ACM Int. Conf. on Utility and Cloud Computing (UCC 2012), Nov 5-8, 2012.
[48] Hagen, L., "A wind erosion prediction system to meet user needs", Journal of Soil and Water Conservation March/April 1991, vol. 46, iss. 2, pp. 105-111.
[49] United State Department of Agriculture – Natural Resources Conservation Service (USDA-NRCS), 2003, "Interpreting the Soil Conditioning Index: A tool for measuring soil organic matter trends", http://soils.usda.gov/SQI/management/files/sq_atn_16.pdf
[50] Gassman, P. Reyes, M., Green, C., Arnold, J., "The Soil And Water Assessment Tool: Historical Development, Applications, and Future Research Directions", Transactions of the American Society of Agricultural and Biological Engineers (ASABE 2007), vol. 50, iss. 4, pp. 1211-1250.

[51] Ascough, J., David, O., Krause, P., Heathman, G., Kralisch, S., Larose, M., Ahuja, L., Kipka, H., “Development and application of a modular watershed-scale hydrologic model using the object modeling system: runoff response evaluation”, Transactions of the American Society of Agricultural and Biological Engineers (ASABE 2012), vol. 55, iss. 1, pp. 117-135.

[52] Chen, W., Deng, E., Du, R., Stanley, R., Yan, C., “Crossing and Nesting of Matching and Partitions”, Transactions of the American Mathematical Society, vol. 359, No. 4, April 2007, pp. 1555-1575.

[53] Dean, J., Ghemawat, S., “MapReduce: Simplified Data Processing on Large Clusters”, In Communications of the ACM, vol. 51, Iss. 1., Jan 2008, pp. 107-113.