Dynamic Scaling for Service Oriented Applications: Implications of Virtual Machine Placement on IaaS Clouds

Wes Lloyd[1,2], Shrideep Pallickara[1], Olaf David[1,2], 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, mazdak.arabi@colostate.edu

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

***Abstract*— Abstraction of physical hardware using infrastructure-as-a-service (IaaS) clouds leads to the simplistic view that resources are homogeneous and that infinite scaling is possible with linear increases in performance. Support for autonomic scaling of multi-tier service oriented applications requires determination of when, what, and where to scale. "When" is addressed by hotspot detection schemes using techniques including performance modeling and time series analysis. "What" relates to determining the quantity and size of new resources to provision. "Where" involves identification of the best location(s) to provision new resources. In this paper we investigate primarily "where" new infrastructure should be provisioned, and secondly "what" the infrastructure should be. Dynamic scaling of infrastructure for service oriented applications requires rapid response to changes in demand to meet application quality-of-service requirements. We investigate the performance and resource cost implications of VM placement when dynamically scaling server infrastructure of service oriented applications. We evaluate dynamic scaling in the context of providing modeling-as-a-service for two environmental science models.**

Keywords Autonomic computing; IaaS; Virtualization; Multi-Tenancy; Resource Management and Performance;

I. INTRODUCTION

Supporting dynamic scalability for service oriented applications introduces resource management challenges that must be addressed by Infrastructure-as-a-Service (IaaS) clouds. These challenges can be broken down into three primary concerns: (1) Determining WHEN infrastructure should be provisioned? (2) Determining WHAT infrastructure should be provisioned? And (3) Determining WHERE infrastructure should be provisioned?

WHEN server infrastructure should scale to cope with demand is informed by hotspot detection [5]. Determining when to scale is complicated by the latency of virtual machine (VM) launches. In some cases, the time required to provision and launch new VMs exceeds the duration of demand spikes! By predicting future demand server infrastructure can be pre-provisioned in anticipation. Load prediction can be difficult particularly for applications with stochastic load behavior. Care must be exercised as poor predictions can result in overprovisioning and higher hosting costs.

WHAT server infrastructure should be provisioned concerns the size (*vertical scaling*) and quantity (*horizontal scaling*) of new VM allocations. Vertical scaling involves modifying resource allocations of existing VMs. Altering VM CPU core, memory, disk, and network bandwidth allocations may alleviate poor performance. When vertical scaling is unavailable or insufficient to address demand horizontal scaling is often used. New VMs are launched and workload of stressed application tiers is balanced across a pool. The key challenge lies in determining how many VMs should be provisioned, and with what resource allocations?

WHERE server resources should be provisioned is abstracted by the virtual infrastructure manager (VIM). Representing VMs as tuples and using them to pack physical machines (PMs) can be thought of as an instance of the multidimensional bin-packing problem that has been shown to be NP-hard. Two basic VM placement schedulers common to private cloud VIMs which support launching VMs on local data centers include greedy and round-robin. Greedy allocation deploys all VMs to a single PM first. When the host's resources are exhausted another PM is selected and the process is repeated. Greedy allocation packs resources tightly, enabling maximum energy savings without regard to VM/application performance. Round-robin placement distributes VMs to each PM in succession, balancing the VM hosting load across the cluster. Round-robin placement typically provides better VM performance by reducing resource contention at the expense of higher energy requirements. Using round-robin placement, all PMs in the cluster receive a portion of the VM hosting load, eliminating potential for idle machines to operate in power saving modes [6].

A. Research Contributions

Previous research has explored alternative methods of VM placement to improve load balancing, server utilization, and energy savings [6] [7] [8] [9] [10] [11] [12]. In this paper we investigate the consequences of VM placement when dynamically scaling infrastructure for service oriented applications. Specifically, we investigate the effects of VM placement on software services scalability. To investigate implications of VM placement on scalability we developed the Least-Busy VM placement scheduler, a load-aware VM placement scheduler. To identify available resource capacity

across physical hosts an aggregated resource utilization metric known as Busy-Metric, is introduced. Resource utilization data from all VMs and their physical hosts is used to calculate spent resource capacity. Busy-Metric scores are used by Least-Busy to provide load balanced placement of new VMs across PMs. To our knowledge this is the first study that investigates how scalability of service oriented applications is impacted by IaaS cloud VM placement.

B. Research Questions

This paper investigates the following research questions:

RQ-1: What performance implications result from VM placement location when dynamically scaling service oriented applications? How important is VM placement for scaling in response to increasing service demand?

RQ-2: How do resource costs (# of VMs) vary when dynamically scaling service oriented applications as a result of VM placement location?

II. Background and Related Work

Amazon's public cloud implements the Elastic Compute Cloud (EC2) application programming interface (API) enabling programmatic control of resource elasticity. The EC2 API is supported by many open source cloud VIMs. Service-oriented applications harness the EC2 API to enable scalability using private and/or public cloud resources. Private clouds provide the base infrastructure while demand bursts are serviced with public cloud resources (hybrid cloud). Private cloud VIMs providing an implementation of the EC2 API include: Apache CloudStack [1], Eucalyptus [2], OpenNebula [3], and OpenStack [4].

All private cloud VIMs provide similar mechanisms for provisioning VMs on demand. Eucalyptus supports both greedy and round robin VM placement schemes [2]. VM deployment can be localized to specific clusters or subnets using EC2 security groups and availability zones. Apache CloudStack provides "fill first" VM placement, equivalent to greedy allocation, and "disperse" mode, equivalent to round-robin [1]. Additionally custom allocators support implementation of new VM scheduling schemes. OpenStack provides two primary VM schedulers known as fill-first and spread-first. Fill-first, equivalent to greedy placement, packs VMs tightly onto PMs. Spread-first distributes VMs across PMs in round-robin fashion, but schedules VMs on PMs having the highest number of available CPU cores and memory first. OpenStack supports filters which enable VMs to be co-located or separated as desired to achieve advantageous deployments for applications. OpenNebula provides both a "packing" policy, equivalent to greedy placement, and a "striping" policy equivalent to round-robin [3] [13]. Additionally custom "rank" expressions are supported which calculate hosting preference scores for each PM. When a VM launch request is received, the PM with the highest score is delegated as host. Scores are recalculated for each VM launch request. Eight system variables can be used in custom rank expressions, none of which are resource utilization statistics. Supported variables include: hostname, total CPUs, free CPUs, used CPUs, total memory, free memory, used memory, and hypervisor type.

Of the VM schedulers offered by cloud infrastructure managers none consider the load characteristics of the VM hosts. Only capacity parameters such as # of CPUs, available memory and disk space are considered to ensure VM allocations have sufficient resources to run. To better support dynamic scaling of service-oriented application infrastructure, VM schedulers should consider resource utilization across physical resources to improve application performance and cluster load balancing.

Previous research on dynamic scaling in the cloud has investigated WHEN to scale including work on autonomic control approaches and hotspot detection schemes [7] [14] [15] [16] [17]. These and other efforts additionally focus on WHAT to scale in terms of vertical and horizontal scaling [18] [19]. Investigations on WHERE to scale related to VM scheduling have largely focused on task/service placement [20] [21] or supporting VM live migration for load balancing [7] [11] [12] or energy savings via VM consolidation across physical hosts [7] [8] [9] [10] [12].

III. Dynamic Application Scaling

A. VM-Scaler

To investigate implications of VM placement for dynamic scaling we developed VM-Scaler, a REST/JSON-based web services application. VM-Scaler harnesses the Amazon EC2 API to support application scaling and cloud management and currently supports Amazon's public elastic compute cloud (EC2), and Eucalyptus versions 3.1 and 3.3. VM-Scaler provides cloud control while abstracting the underlying IaaS cloud and is extensible to any EC2 compatible VIM. VM-Scaler provides a platform for conducting IaaS cloud research by supporting experimentation with hotspot detection schemes, VM management/placement, and job scheduling/ proxy services.

Upon initialization VM-Scaler probes the host cloud and collects metadata including location and state information for all PMs and VMs. An agent installed on all VMs/PMs sends resource utilization statistics to VM-Scaler at fixed intervals. Collected resource utilization statistics are described in [22][23]. This extends our previous work investigating the use of resource utilization statistics for guiding cloud application deployment.

VM-Scaler supports horizontal scaling of application infrastructure by provisioning VMs when application hotspots are detected. One or more VMs can be launched in parallel in response to application demand. To initiate scaling, a service request is sent to VM-Scaler to begin monitoring a specific application tier. VM-Scaler monitors the tier and launches additional VMs when hotspots are detected. VM-Scaler handles launch failures, automatically reconfigures the proxy server, and provides application specific configuration before adding new VMs to a tier's working set. Tier-based scaling in VM-Scaler is conceptually similar to Amazon auto-scaling groups [24].

Three configurable timing parameters are provided to support autonomic scaling: `min_time_to_scale_again`,

`min_time_to_scale_after_failure`, and `max_VM_launch_time`. `Min_time_to_scale_again` provides a time buffer before scaling again, allowing time to consider the impact of recent resource additions. This parameter helps to eliminate the ping-pong effect described in [25] and is equivalent to Amazon Scaling Group cool-down periods [24]. `Max_VM_launch_time` provides a maximum time limit before terminating launches that appear to have stalled. This supports handling launch failures by reissuing stalled launch requests. `Min_time_to_scale_after_failure` provides an alternate wait time when VM launch failures occur.

B. Busy-Metric

The Busy-Metric ranks resource utilization of the physical host machines by calculating total CPU time, disk sector reads/writes, network bytes sent/received for all VMs and PMs. Each component is normalized to 1 by dividing by observed approximate maximums for each resource utilization statistic. CPU time is double weighted to assign more importance to free CPU capacity.

A VM capacity parameter is included to prevent too many VMs from being allocated to a single host. Busy-metric scores of the physical host increase linearly for each additional VM hosted at a rate described using equation 3. The rate increases faster for hosts with fewer CPU cores. Incorporating this parameter enables Busy-metric to favor hosts having the fewest guest VMs. When PMs host fewer guests the degree of hypervisor level context switching required to multiplex resources is reduced. This practice should help reduce virtualization overhead.

Agents installed on all VMs and PMs are configured to send VM-Scaler resource utilization data every 15 seconds. One second averages using the last minute of data samples were used to calculate the Busy-Metric. Observed values for each parameter are divided by approximate one second maximum capacities of the physical hardware determined through testing.

For example:

$$cputime_n = \frac{cputime_{obs\_1sec}}{cputime_{\max\_1sec}} \quad (1)$$

Our Busy-Metric is expressed as:

$$\frac{(2 \cdot cputime_n) + dsr_n + dsw_n + nbr_n + nbs_n + (\frac{2 \cdot Hosted\_VMs}{PM_{cores}})}{7} \quad (2)$$

Each additional VM hosted linearly increases the value of the Busy-Metric by:

$$e^{(\ln PM_{cores} - 1.2528)} \quad (3)$$

The Busy-Metric provides an approach to rank available capacity of physical host machines. Our goal has been to develop a general metric to support VM scheduling based on the total shared load on PMs. CPU time is double weighted because our environmental science models are primarily CPU bound applications. **Many busy metric variations are possible. Our goal has not been to develop the perfect metric, but to investigate implications of VM placement for dynamic scaling.**

IV. Experimental Investigation

A. Experimental Setup

To investigate research questions presented in section 1 we test dynamic scaling for two environmental models: the Revised Universal Soil Loss Equation – Version 2 (RUSLE2) [26], and the Wind Erosion Prediction System (WEPS) [27]. RUSLE2 and WEPS are the US Department of Agriculture–Natural Resource Conservation Service standard models for soil erosion used by over 3,000 county level field offices across the United States. RUSLE2 and WEPS are used to provide soil erosion modeling services to end users. RUSLE2 was developed primarily to guide natural resources conservation planning, inventory erosion rates, and estimate sediment delivery. The Wind Erosion Prediction System (WEPS) is a daily simulation model which outputs average soil loss and deposition values for selected areas and periods of time to predict soil erosion due to wind.

RUSLE2 was originally developed as a Windows-based Microsoft Visual C++ desktop application. RUSLE2 is deployed as a REST/JSON based web service hosted by Apache Tomcat [28]. WEPS was originally developed as a desktop Windows application using Fortran95 and Java. WEPS has been ported to Linux to operate as a REST/JSON based web service. Both applications are deployed as part of the USDA's Cloud Services Innovation Platform [29].

TABLE I. RUSLE2/WEPS APPLICATION COMPONENTS

Component		RUSLE2	WEPS
$\mathcal{M}$	Model	Apache Tomcat 6.0.20, Wine 1.0.1, RUSLE2, OMS3 [31] [32]	Apache Tomcat 6.0.20, WEPS
$\mathcal{D}$	Database	Postgresql-8.4, PostGIS 1.4, soils data (1.7 million shapes), management data (98k shapes), climate data (31k shapes), 4.6 GB total for Tennessee	Postgresql-8.4, PostGIS 1.4, soils data (4.3 million shapes), climate/wind data (850 shapes), 17GB total, western US data.
$\mathcal{F}$	File server	nginx 0.7.62 file server, 57k XML files (305MB), parameterizes RUSLE2 model runs.	nginx 0.7.62 file server, 291k files (1.4 GB), parameterizes WEPS model runs.
$\mathcal{L}$	Logger	Codebeamer 5.5, Apache Tomcat (32-bit), Ia-32libs	Redis 2.2.12 distributed cache server

RUSLE2 and WEPS provide good candidates to prototype service oriented application scaling. Their architecture consisting of a web application server, geospatial relational database, file server, and logging server is analogous to many service oriented applications. Components of the models are described in Table I. A PM ran the HAProxy load balancer to redirect modeling requests to the active pool of $\mathcal{M}$ VMs. HAProxy is a high performance load balancer that supports proxying TCP and HTTP socket-based network traffic [30].

B. Hardware Configuration

We conducted scaling tests using a Eucalyptus 3.1.2 IaaS private cloud deployed across nine SUN X6270 blade servers interconnected by a Giga-bit VLAN. Each blade server had dual Intel Xeon X5560-quad core 2.8 GHz CPUs, 24GB ram, and dual 15000rpm HDDs. The host operating system was Ubuntu 12.04 Linux (3.2.0-29) 64-bit server. The XEN hypervisor version 4.1.2 provided VMs in paravirtual mode. VM guests ran Ubuntu Linux 9.10 (2.6.31) 64-bit server. Six blade servers were used as Eucalyptus node-controllers to host VMs, one blade server hosted the Eucalyptus cloud-controller, cluster-controller, walrus server, and storage-controller services. Eucalyptus managed mode networking was used to support network isolation of VMs using private VLANs. A separate blade server was used to generate the modeling work load. Another blade server acted as a client for file transfers to create background network activity for shared load testing.

Random test generation was used to generate 10,000 unique RUSLE2 and 1,000 WEPS test cases. RUSLE2 tests used geospatial data from the state of Tennessee. WEPS tests used data primarily from Kansas and Colorado where soil erosion due to wind is a large environmental concern. For scaling tests, individual WEPS model runs were terminated after 10 minutes. This was necessary because some randomly generated WEPS runs required more than 30 minutes to execute.

C. Test Configurations

To simulate shared cluster load present in a public cloud we generated artificial load on the six PMs which hosted VMs. Table II describes our shared cluster load and the corresponding PM Busy-Metric scores prior to executing any tests. Our goal was to simulate potential public cloud load conditions where users compete for server resources. Custom scripts generated load activity. CPU load was created for a specified number of cores by performing continuous math computations. Disk load was created by continuously reading, writing, or copying a text file. To force the system to continuously reread the file, cache clearing as described previously was performed. To create network load a VM image file was constantly transferred to/from a non-cloud blade server. Sftp's "-l" flag was used to control the transfer bandwidth.

TABLE II. SHARED CLUSTER LOAD

Cloud Node	R2	WEPS	CPU	Disk	Network	Busy-Metric
PM-1	$\mathcal{M}$	$\mathcal{D}$	2 cores@25%			.083
PM-2	$\mathcal{D}\,\mathcal{L}$	$\mathcal{L}$	4 cores@25%		↑ @20%	.285
PM-3	$\mathcal{F}$		6 cores@25%			.240
PM-4		$\mathcal{M}\,\mathcal{F}$	5 cores@25%		↓ @20%	.240
PM-5			2 cores@25%			.082
PM-6			4 cores@25%			.156

Table III describes VM size, modeling request rates, and request rate increments for our scaling tests. An exponential distribution based random sleep function spaced individual model requests to simulate a Poisson distribution. This enabled requests to occur at random intervals while still achieving the desired constant request rate.

TABLE III. VM SCALING TESTS

VM size	Mem(MB)/ Disk(GB)	Rusle2 Req Rate	WEPS Req Rate	Increment RUSLE2	Increment WEPS
2-core	1024 / 3	.5-16/sec	.1-1/sec	.25/15s	.025/15s
4-core	2048 / 3	1-16/sec	.1-1/sec	.5/30s	.05/30s
8-core	4096 / 3	2-16/sec	.1-1/sec	1/min	.1/min

Scaling thresholds for hotspot detection were increased linearly for 2-core, 4-core, and 8-core VM tests. The intent was to use identical scaling thresholds relative to the number of VM CPU cores. Figure 1 provides a quartile plot of RUSLE2 vs. WEPS execution times. WEPS model runs consume nearly 100% of a CPU core for their duration averaging from 80-100 seconds compared to a few seconds for RUSLE2 runs. For RUSLE2 hotspot detection was performed by monitoring resource utilization of the initial worker VM as individual model execution times were short (2s average) and homogenous. Load was evenly balanced across worker VMs using HAproxy round-robin load balancing. This approach was insufficient for WEPS, as model runs had twice the variance and were much longer in duration. For WEPS hotspot detection we calculated average CPU time, CPU idle time, and # of context switches for the entire pool of worker VMs and launched additional worker VMs when averages exceeded the scaling thresholds.

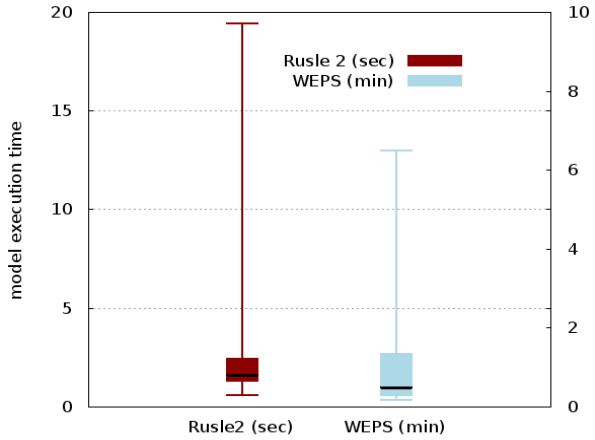

Figure 1. RUSLE2 vs. WEPS Model Execution Time Quartile Box Plot

TABLE IV. DYNAMIC SCALING TESTS

VM size	Rusle 2	WEPS
2-core VMs	20 scale tests	20 scale tests
4-core VMs	20 scale tests	20 scale tests
8-core VMs	20 scale tests	20 scale tests

V. EXPERIMENTAL RESULTS

Table IV summarizes RUSLE2 and WEPS scaling tests completed to support investigation of our research questions presented in section 1. Scaling tests were conducted twice, once using Least-Busy VM placement and again using round-robin VM placement. 60 test sets of 6500+ Rusle2 model runs, and 60 test sets of 300 WEPS model runs were

conducted for a total of over 800,000 model runs. For 2-core VM WEPS testing, sequential VM launches were insufficient to achieve good results. To add resources more rapidly three 2-core VMs were launched in parallel.

TABLE V. SCALING TEST RESULTS

VM size	Rusle 2	WEPS
2-core VMs	*lb<rr* p=.014 df=18.2	*lb<rr* p=.162 *n.s.*[1]
4-core VMs	*lb<rr* p= 0.065 df = 22.7	*lb<rr* p= .035 df = 24.65
8-core VMs	*lb<rr* p=.017 df=24.5	*lb<rr* p=.00003 df=33.796

[1] (3) 2-core VMs launched per scaling event

Table V shows the statistical significance from t-tests indicating if performance means for scaling performance with Least-Busy were different from round-robin. Normalized performance improvements of Least-Busy VM placement relative to round-robin are shown in figure 2. Across all tests, RUSLE2 performance improved by 16% on average using Least-Busy VM placement, while WEPS performance improved 12%. Least-Busy exhibited its fastest differential performance for Rusle2 with 2-core tests (29.3% faster, 3.2 hrs cputime savings/scaling test), and WEPS for 8-core tests (19.1% faster, 1.6 hrs cputime savings/scaling test). Least-Busy VM placement enabled better model performance for all tests.

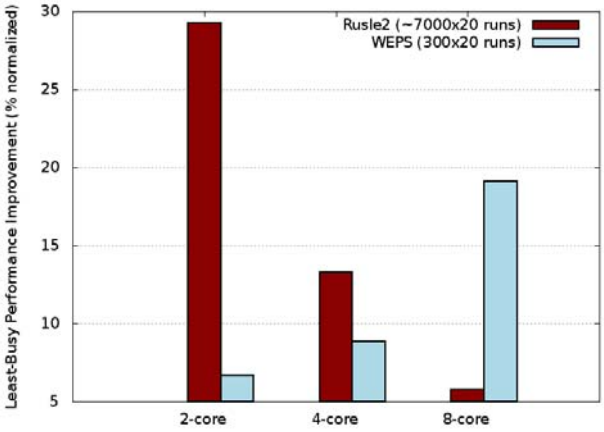

Figure 2. Least-Busy VM Placement
Application Performance Improvement (normalized)

Figure 3 shows the normalized resource cost savings in percentage of VM allocations. Least-Busy VM placement supported execution of the modeling workload with fewer resources. RUSLE2 required on average 3.2% fewer VMs and WEPS 2.2%. The most economical deployment used 2-core VMs for RUSLE2 (28.92 total cores avg) and 4-core VMs for WEPS (30.56 total cores avg). Least-Busy VM placement enabled hosting the same modeling workload with fewer VMs and total CPU cores. Less physical server capacity was required for hosting while faster modeling performance was achieved.

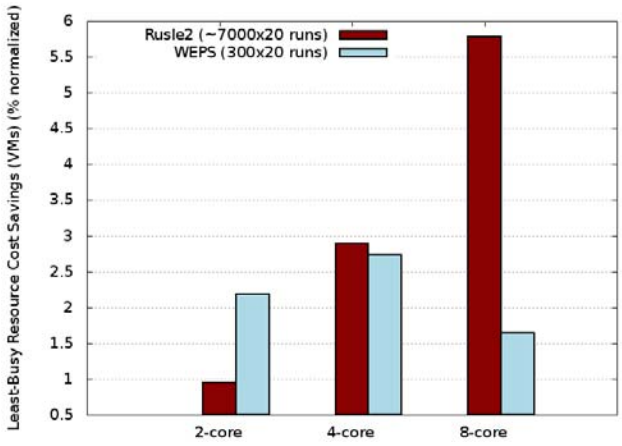

Figure 3. Least-Busy VM Placement
Resource Cost Savings (% of VMs- normalized)

Average VM launch times are shown in figure 4. Overall stress from hosting the WEPS workload was higher than RUSLE2 resulting in ~10% slower average VM launch times. For all tests Least-Busy VM launch times were faster except for 2-core WEPS tests. For these tests, three VMs were launched in parallel producing this performance degradation.

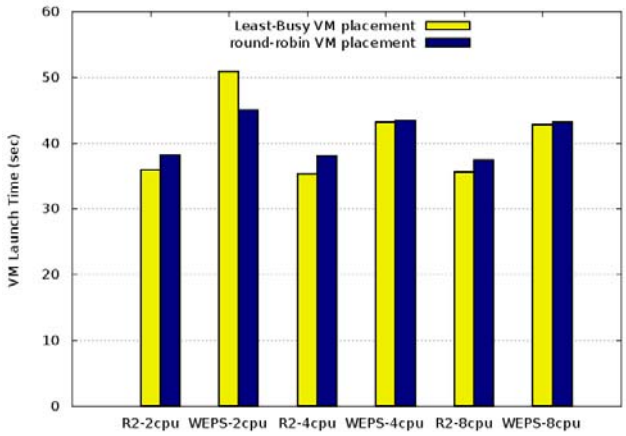

Figure 4. VM Launch Times (seconds):
Least-Busy and Round-Robin VM Placement

Hosting our modeling workload using VMs with fewer cores provided a greater challenge as more VMs had to be rapidly launched adding to system stress. For WEPS 2-core tests launching VMs in parallel increased overhead and degraded performance illustrating tradeoffs between cluster size, launch overhead, and application performance. For 2-core tests, nearly continuous sequential VM launches were required for RUSLE2 to cope with demand. WEPS model execution times nearly doubled with 20% of runs timing out after 10 minutes. Parallel VM launches helped our WEPS 2-core VM configuration achieve performance similar to 4-core configurations.

VI. CONCLUSIONS

Dynamic scaling performance of service oriented applications hosted using IaaS clouds depends on carefully distributing new VMs across physical hosts. Comparing our Least-Busy VM placement approach versus round-robin we observed performance improvements up to 29% (RUSLE2) and 19% (WEPS) with average improvements of 16% and 12% respectively. Dynamically scaling our service oriented applications with Least-Busy VM placement required 2-3% fewer VMs while achieving these observed performance improvements!

Abstraction of physical hardware using IaaS clouds leads to the simplistic view that resources are homogenous and scaling can infinitely provide linear increases in performance. Our results demonstrate that decisions regarding VM placement location have important performance and resource cost implications. Ad-hoc VM placement schemes appear inefficient at managing server infrastructure while providing resources for dynamic scaling**.** Our results demonstrate how private clouds can deliver potential performance and resource cost savings by addressing resource management inefficiencies.

ACKNOWLEDGEMENTS

This research is supported by a grant from the US National Science Foundation's Computer Systems Research Program (CNS-1253908).

REFERENCES

[1] CloudStack Admin. Guide, 2013, http://incubator.apache.org/cloudstack/docs/en-US/Apache_CloudStack/4.0.1-incubating/html/Admin_Guide

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

[3] Llorente, R et al, 2011. On the Management of Virtual Machines for Cloud Infrastructures (ch. 6), in Cloud Computing: Principles and Paradigms, J Wiley & Sons, Inc., Hoboken, NJ, USA.

[4] OpenStack Compute Admin. Manual-Essex (2012.1), 2013, http://docs.openstack.org/essex/openstack-compute/admin/content/index.html

[5] P. Saraipalli at al., Load Prediction and Hot Spot Detection Models for Autonomic Cloud Comp., Proc.4th IEEE/ACM Int.Conf. on Utility and Cloud Comp (UCC 2011), Melbourne, Australia, Dec 2011, pp. 397-402.

[6] A. Gandhi et al., Optimal power allocation in server farms, Proceedings of the 11th Int. Conf. on Measurement and Modeling of Comp. Systems (SIGMETRICS'09), Seattle, WA, USA, June 15-19 2009, pp. 157-168.

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

[8] M. Andreolini et al., Dynamic Load Management of Virtual Machines in Cloud Architectures, Springer Lecture Notes in Comp.Sci, Social-Informatic and Telecom. Engineering, vol. 34, 2010, pp. 201-214.

[9] A. Beloglazov, R. Buyya, Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in Cloud data centers, Concurrency and Computation: Prac.and Exp., v24, n13, Sept.2012, pp.1397-1420.

[10] A. Roytman et al., Algorithm design for performance aware VM consolidation. Tech. rep, Microsoft Research, 2013,MSR-TR-2013-28.

[11] M. Mishra, A. Sahoo, On Theory of VM Placement: Anomalies in Existing Methodologies and Their Mitigation Using a Novel Vector Based Approach, IEEE Conf. on Cloud Comp. (CLOUD 2011), Washington, D.C., USA, pp. 275-282.

[12] Z. Xiao, W. Song, Q. Chen, Dynamic Resource Allocaiton using Virtual Machines for Cloud Computing Environment, IEEE Trans. on Parallel and Distributed Systems, vol. 24, No. 6, June 2013, pp. 1107-1117.

[13] OpenNebula - Open Source Data Center Virtualization, 2013, http://opennebula.org/documentation:rel3.8:template# placement_section

[14] O. Niehörster et al., Autonomic resource management with support vector machines, Proc. 12th IEEE/ACM International Conference On Grid Computing (GRID'11), Lyon, France, Sept 2011, pp. 157-164.

[15] 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.

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

[17] P. Lama, X. Zhou, Autonomic provisioning with self-adaptive neural fuzzy control for end-to-end delay guarantee, Proc. 18th IEEE/ACM Int. Symp. on Modeling, Analysis and Sim. of Comp.and Telecom Sys (MASCOTS 2010), Miami Beach FL, Aug 2010, pp.151-160.

[18] D. Jayasinghe et al., Variations in performance and scalability when migrating n-tier applications to different clouds, Proc.4th IEEE Int. Conf. on Cloud Comp (Cloud'11), WashingtonDC,Jul, 2011, p 73-80.

[19] H. Van, F. Tran, J. Menaud, Autonomic Virtual Resource Management for Service Hosting Platforms, Proc. IEEE Workshop on SoftEng Challenges in Cloud Comp (ICSE CLOUD '09), Vancouver, Canada, May 2009, 8p.

[20] 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, J. of Sys. and Sftwre, vol. 84, 2011, pp. 1270-1291.

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

[22] W. Lloyd et al., Performance implications of multi-tier application deployments on IaaS clouds: Towards performance modeling, Future Generation Computer Systems, v.29, n.5, 2013, pp.1254-1264.

[23] W. Lloyd et al., Performance Modeling to Support Multi-Tier Application Deployment to IaaS Clouds, IEEE/ACM Int. Conf. on Utility and Cloud Computing (UCC 2012), Nov 5-8, 2012, 8p.

[24] AWS Documentation: Concepts – Auto Scaling , 2013, http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AS_Concepts.html

[25] A. Kejariwal, Techniques for Optimizing Cloud Footprint, Proc. 1st IEEE Int. Conf. on Cloud Eng (IC2E), Mar 25-27, 2013, pp. 258-268.

[26] U.S. Department of Agriculture - Agricultural Research Service, Revised Universal Soil Loss Equation Ver. 2 (RUSLE2), http://www.ars.usda.gov/SP2UserFiles/Place/64080510/RUSLE/RUSLE2_Science_Doc.pdf

[27] L. Hagen, "A wind erosion prediction system to meet user needs", J. of Soil and Water Conservation Mar/Apr 1991, v.46 (2), pp.105-111.

[28] Apache Tomcat - Welcome, 2011, http://tomcat.apache.org/

[29] W. Lloyd et al., The Cloud Services Innovation Platform - Enabling Service-Based Environmental Modelling Using IaaS Cloud Comp. iEMSs 2012 Int. Cong on Env.Modeling and Software, Germany, Jul 2012, 8 p.

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

[31] O. David et al., A software engineering perspective on environmental modeling framework design: The Object Modeling System, Environmental Modeling & Software, vol.39, Jan 2013, pp. 201-213.

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