Mitigating Resource Contention and Heterogeneity in Public Clouds for Scientific Modeling Services

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

Shrideep Pallickara[1], Olaf David[1,2], Mazdak Arabi[2]
[1]Department of Computer Science
[2]Dept. of Civil and Env. Engineering
Colorado St. Univ., Ft. Collins, USA
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. Hosting scientific modeling services using IaaS clouds requires awareness of application resource requirements and careful management of cloud-based infrastructure. In this paper, we present multiple methods to improve public cloud infrastructure management to support hosting scientific model services. We investigate public cloud VM-host heterogeneity and noisy neighbor detection to inform VM trial-and-better selection to favor worker VMs with better placements in public clouds. We present a cpuSteal noisy neighbor detection method (NN-Detect) which harnesses the cpuSteal CPU metric to identify worker VMs with resource contention from noisy neighbors. We evaluate potential performance improvements provided from leveraging these techniques in support of providing modeling-as-a-service for two environmental science models.**

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

I. Introduction

Public cloud environments abstract the physical hardware implementation of resources providing users with only limited details regarding the actual physical hardware implementations. This abstraction hides resource heterogeneity and restricts users' ability to make well informed deployment decisions based on real knowledge about the state of the system. Resource heterogeneity includes differences in the physical hardware used to implement identically named resources, and also performance heterogeneity resulting from contention for shared CPU, disk, and network resources. This lack of information forces users to rely on the good will of cloud hosts to make resource deployments. But to what degree can we assume that cloud hosts will protect users from potential performance degradation and increased hosting costs resulting from heterogeneous resources?

Infrastructure-as-a-Service (IaaS) cloud resource management 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?

Determining WHAT server infrastructure to provision for application hosting concerns two dimensions: type of resources (vertical sizing), and quantity (horizontal sizing) of virtual machines (VM) resources. Vertical sizing involves selecting the appropriate resource configuration "type" considering: the required number of CPU cores, memory allocation, disk capacity, and network bandwidth. On public clouds such as Amazon EC2, specific VM resource configurations are known as "instance types". Each instance type, for example m3.large or c3.xlarge, describes a given VM configuration with a fixed amount of RAM, hard disk space, and network bandwidth. Horizontal sizing involves determining the appropriate quantity of VMs for workload hosting and then load balancing the workload across this VM pool. Instance types such as the Amazon EC2 m1.large VM have several physical implementations and research has demonstrated a 30% performance variance for web application hosting [1] [2].

WHERE server resources are provisioned within a specific datacenter is abstracted by IaaS clouds. Representing VMs as tuples and using them to pack physical machines (PMs) can be thought of as an example of the multidimensional bin-packing problem shown to be NP-hard [3]. In public cloud settings, VM placement is abstracted and difficulty to discern [4]. Previous efforts using heuristics to infer VM co-residency by launching probe VMs for exploration can be expensive and only partially effective at determining VM locations [5] [17] [18] [19] [20]. Public clouds do support features including Virtual Private Clouds (VPCs) and "placement groups" to ensure locality of VM placements for application hosting. These features help ensure data center and rack locality for application VMs. – but they do ***not*** ensure resource quality. Though application resource placements are localized, the physical hosts may be overprovisioned, or host "noisy neighbor" VMs, resource hungry VMs which consume an unusual share of CPU, disk, or network resources. This resource contention has been shown to degrade performance of scientific applications hosted in public clouds [4] [6].

In this paper, we investigate the performance implications of public cloud infrastructure abstraction for hosting scientific modeling web services to provide scientific "modeling-as-a-service" for ubiquitous clients. Specifically, we investigate the implications of *WHAT* infrastructure *gets* provisioned (resource implementation), and *WHERE* this infrastructure is provisioned (resource state). We first quantify performance degradation resulting from public cloud resource type heterogeneity (*WHAT*), and second from resource contention (*WHERE*). We benchmark the performance by hosting two US Department of Agriculture environmental modeling web service applications on Amazon's public EC2 cloud. Secondly, we propose two approaches which help mitigate this performance degradation through intelligent public cloud resource provisioning.

A. *Research Questions*

We investigate the following research questions:

RQ-1: How common is public cloud VM-type implementation heterogeneity?

RQ-2: What performance implications result from this heterogeneity for hosting web services application workloads?

RQ-3: How effective is cpuSteal at identifying worker VMs with high resource contention due to multi-tenancy (e.g. noisy neighbor VMs) in a public cloud?

RQ-4: What are the performance implications of hosting scientific modeling workloads on worker VMs with consistently high cpuSteal measurements in a public cloud?

CpuSteal provides a processor metric that helps quantify overprovisioning of the physical hardware. When a VM CPU core is ready to execute but the physical host CPU core is unavailable because it is performing other work, a CpuSteal tick is registered. Herein we describe our investigation on the feasibility of harnessing this CPU metric to identify resource contention from overprovisioned hosts in public clouds.

B. *Contributions*

This paper reports on our case study which develops and applies two approaches to improve performance of service oriented workloads hosted using public clouds. The primary contributions of this paper include:

1. We investigate the prevalence of resource contention among VMs on a shared host using the cpuSteal CPU metric. We measure cpuSteal for several hours while running compute intensive workloads across eight-different VM types from Amazon's EC2 cloud. We develop an approach called Noisy-Neighbor-Detect (NN-Detect) which harnesses the cpuSteal metric to identify resource contention from noisy neighbor VMs running in a shared cluster. Using this approach we identify VMs which exhibit degraded performance from resource contention, and quantify the potential impact on application performance for hosting USDA scientific model-as-a-service workloads.

2. We investigate the prevalence of host heterogeneity for 12 VM-types spanning 3 generations in 2 regions on Amazon's EC2 cloud. We apply the trial-and-better approach described in [1] to create VM pools with fixed VM implementations with a given CPU type. We quantify the implications for average service execution time for USDA model-as-a-service workloads from heterogeneous VM type implementations.

II. Background and Related Work

A. *Scientific Modeling on Public Clouds*

Ostermann et al. provided an early assessment of public clouds for scientific modeling in [7]. They assessed the ability of 1st generation Amazon EC2 VMs (e.g. m1.*-c1.*) to host HPC-based scientific applications. They identified that EC2 performance, particularly network latency, required an order of magnitude improvement to be practical and suggested that scientific applications should be tuned for operation in virtualized environments. Other early efforts highlighted similar challenges regarding performance for HPC applications [8] [9] [10]. To increase ubiquity of scientific models and applications, and to support dissemination of research, many organizations now build and host scientific web services [11] [12]. The Cloud Services Integration Platform (CSIP), a Java-based programming framework, has been developed by Colorado State University in cooperation with the US Department of Agriculture. CSIP supports development and deployment of web services for a number of USDA scientific modeling applications implemented in languages such as C, FORTRAN, Python, and others. Infrastructure-as-a-Service cloud computing offers an ideal platform for CSIP services which mix legacy code with modern framework instrumentation to provide modeling-as-a-service, on demand, to diverse client applications [13] [14].

B. *Public Cloud Resource Contention Detection*

Schad et al. [6] demonstrated the unpredictability of Amazon EC2 VM performance caused by contention for physical machine resources and provisioning variation of VMs. More recently researchers have developed techniques to identify resource contention from coresident VMs running on shared hardware [15] [16] [17] [18] [19] [20].

Mukherjee et al. developed a software-based resource contention probe based on techniques not specific to hardware or OS specific counters to provide a more portable technique [17]. They monitored TCP/IP connection rates and memory contention from cache misses, but their work was restricted to a small private cluster. Novakovic et al. developed a multi-level approach for detecting and mitigating resource contention in private clusters [18]. They used an initial lightweight warning detector to infer potential contention which would trigger a more complex inference analyzer to perform in depth analysis. Their inference analyzer predicts whether VM migration can alleviate resource contention by predicting system load after migration, as well as where VMs should migrate. Zhang et al. and Varadarajan et al. both exploit side channels to help determine coresidency of VMs, work done from the perspective of demonstrating how cloud systems are potentially insecure. Zhang et al. developed a single and multi-probe classifier which was quite successful at detecting co-located VMs [19]. However, their approach required kernel modifications, extensive benchmarking and training, and doesn't work in a virtual private cloud (VPC) configuration where VMs run on isolated VLANs behind a firewall. Varadarajan et al. used a side channel requiring less analysis, by performing atomic addition across a CPU cache-line boundary, to detect co-located VMs on the Google Compute Engine, Amazon EC2, and Microsoft Azure public clouds [20]. Though novel and effective at detecting VM coresidency a potential security concern, their approach does not detect resource contention among coresident VMs.

Liu offers a novel approach to characterize CPU utilization across a public cloud by analyzing CPU temperature [16]. Liu's approach averages thermal measurements of the CPU from small VMs which are context switched across the physical host's CPU cores for extended periods to approximate CPU die temperature which correlates with CPU utilization. Using this approach Liu observed CPU utilization using 20 m1.small VMs on Amazon EC2 in 2011 for 1 week and estimated average CPU utilization to be around 7.3%. Liu did not consider if temperature can help detect resource contention and application performance degradation.

A few efforts have looked at using cpuSteal to detect contention among VMs [15] [21]. Casale et al. considered the use of cpuSteal to look for contention between hyperthreads of the same CPU-core using private servers [21]. Ayodele et al. demonstrated a link between application performance and cpuSteal using SPEC CPU2006 benchmarks [15]. They verified this link on both a private cluster and with the m3.medium Amazon EC2 VM type. This effort did not further develop the cpuSteal-performance link into a mechanism to support contention avoidance and performance improvement.

C. Public Cloud Heterogeneity

Farley et al. demonstrated that Amazon EC2 instance types had heterogeneous hardware implementations in [2]. Their investigation focused on the m1.small instance type and demonstrated potential for cost savings by discarding VMs with lower performance CPUs. Ou et al. extended their work by demonstrating that heterogeneous implementations impact several Amazon and Rackspace VM types [1]. They found that the m1.large EC2 instance had four different hardware implementations (variant CPU types) and different Xen CPU sharing configurations. They demonstrated ~20% performance variation on operating system benchmarks for m1.large VM implementations. This technique enabled the development of a "trial-and-better" approach where the backing CPU type of VM instances upon launch are checked, and those with lower performing CPU types are terminated and relaunched. This technique provided cost savings through performance improvements when using m1.large EC2 instances for 10 or more hours.

III. Science Applications

To study public cloud performance implications from noisy neighbor VMs and resource type heterogeneity, we harnessed two environmental modeling services implemented using the Cloud Services Integration Platform (CSIP) framework [13][14]. CSIP provides a common REST/JSON Java-based programming framework supporting the development of web services hosted using web containers such as Apache Tomcat [29]. CSIP has been developed by Colorado State University and the US Department of Agriculture (USDA) to support environmental modeling web service development.

We investigated implications for scaling workloads for two soil erosion models: the Revised Universal Soil Loss Equation – Version 2 (RUSLE2) [22], and the Wind Erosion Prediction System (WEPS) [23]. RUSLE2 focuses on modeling soil erosion due to rainfall and runoff, while WEPS predicts soil erosion due to wind. Both model services represent diverse applications with varying computational requirements. RUSLE2 and WEPS represent the US Department of Agriculture–Natural Resource Conservation Service agency standard models used by over 3,000 county level field offices across the United States to predict soil erosion. RUSLE2 models soil erosion primarily by parameterizing the RUSLE equation, whereas WEPS is a daily process-based simulation model.

RUSLE2 was originally developed as a Windows-based Microsoft Visual C++ desktop application. WEPS was originally developed as a desktop Windows application using Fortran95 and Java. The RUSLE2 and WEPS services require legacy language support, a mixture of Linux and Windows support, as well as relational, geospatial, and NoSQL databases. RUSLE2 and WEPS model service components are described in Table I. IaaS cloud computing is key to providing the infrastructure required to provide RUSLE2 and WEPS as scalable web services.

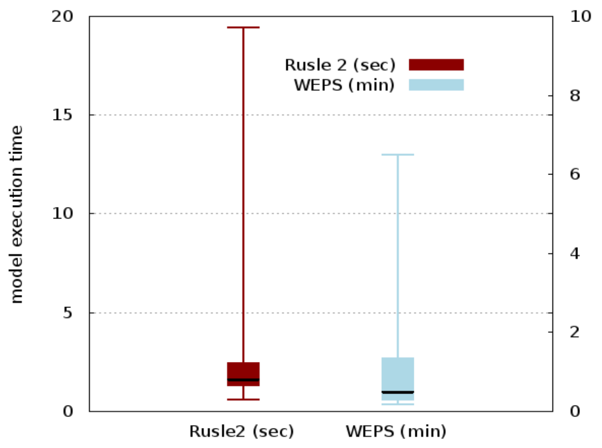

Fig. 1: RUSLE2 vs. WEPS
Model Execution Time Quartile Box Plot

RUSLE2 and WEPS have distinctly different workload profiles. A box plot depicts the average service execution times for RUSLE2 and WEPS in Figure 1. Average cpuUser, cpuKernel, and cpuIdle time for cloud hosted workloads are shown in Figure 2 with RUSLE2 (37.8% | 7.1% | 54.9%) vs. WEPS (83.8% | 4.2% | 12.0%). CpuIdle is time spent waiting for disk or network I/O operations to complete, or time spent performing context switches where the CPU does not actively execute instructions. The RUSLE2 model executes quickly as the model primarily parametrizes the RUSLE equation. Parameterization involves mostly I/O, while the model run is

Identify applicable sponsor/s here. If no sponsors, delete this text box (*sponsors*).

computationally simple. Conversely, WEPS is a CPU-bound process based simulation model which runs for at least 30-years using a daily time step. Their diversity makes RUSLE2 and WEPS good candidates to benchmark cloud infrastructure performance.

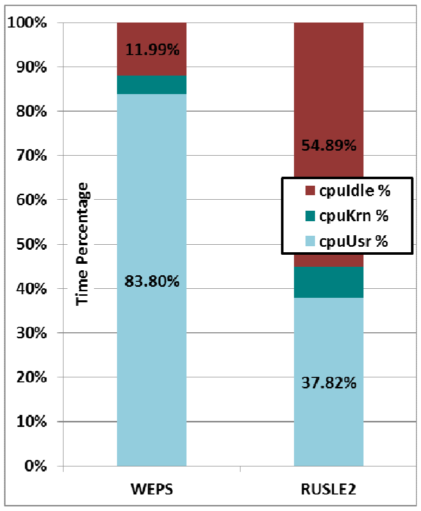

Fig. 2: RUSLE2 vs. WEPS CPU Profile
Comparison of: Kernel time, User mode time, Idle time

TABLE I. RUSLE2/WEPS APPLICATION COMPONENTS

	Component	RUSLE2	WEPS
M	Model	Apache Tomcat, Wine 1, RUSLE2 [22], CSIP [13]	Apache Tomcat, WEPS [23], CSIP [13]
D	Database	Postgresql, PostGIS, soils data (1.7 million shapes), management data (98k shapes), climate data (31k shapes), 4.6 GB total	Postgresql, PostGIS, soils data (4.3 million shapes), climate/wind data (850 shapes), 17GB total.
F	File server	nginx file server, 57k XML files (305MB), parameterizes RUSLE2 model runs.	nginx file server, 291k files (1.4 GB), parameterizes WEPS model runs.
L	Logger	Redis distributed cache server	Redis distributed cache server

To load balance model service requests we used HAProxy, a high performance load balancer [8], to redirect modeling requests across the active pool of M worker VMs. HAProxy, installed on a PM, provides public service endpoints.

IV. THE VIRTUAL MACHINE SCALER

To investigate infrastructure management techniques and support hosting of scientific modeling web services we developed the Virtual Machine (VM) Scaler, a REST/JSON-based web services application [24]. 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 3/4 clouds. VM-Scaler provides cloud control while abstracting the underlying IaaS cloud and is extensible to any EC2 compatible cloud. VM-Scaler provides a platform for conducting IaaS cloud research by supporting experimentation with hotspot detection schemes, VM management/placement, job scheduling, and load balancing of service requests. VM-Scaler supports horizontal scaling of application infrastructure by provisioning VMs when application hotspots are detected, similar to Amazon auto-scaling groups [25].

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 [26][27]. Our development of VM-Scaler extends and enables this prior work investigating the use of resource utilization statistics to guide cloud application deployment.

VM-Scaler supports creation of virtual machine pools to support scaling and management of VMs hosting a specific tier of an application. Pools are defined using JSON files and can be created using dedicated or spot instances on Amazon EC2. VM-scaler supports managing a pool as a cohesive unit with functions to execute a command or deploy new data across all of the members of a pool in parallel. Pool management also includes enforcement of VM type heterogeneity. The forceCpuType attribute specifies the desired backing CPU type in Amazon's public cloud. When a pool is created, non-matching VMs are immediately terminated and replaced until an entire pool of matching VMs is created. This way it is possible to filter out heterogeneous VM types and ensure homogeneity of the backing CPU in a manner similar to the "trial-and-better" approach [1].

VM-Scaler supports profiling the resource requirements of distributed cloud workloads running across a set of VM-scaler managed VMs [28]. A resource utilization sensor is installed on each VM to send resource utilization data to the central VM-Scaler web service at an adjustable interval. By harnessing the Linux network time protocol (ntp) resource sampling can be synchronized across a pool to a resolution of 1-second. This supports collection of accurate workload profiles for workloads running across large VM pools. In this paper, we exercise this feature using pools with up to 60 VMs to collect cpuSteal statistics. VM-scaler workload profiling is similar to Amazon CloudWatch metrics except our implementation supports sampling at a 1-second resolution for free vs. 60-seconds with CloudWatch. VM-scaler provides raw profiling data for researcher consumption to analyze performance and build models.

V. HETEROGENEITY OF VM IMPLEMENTATIONS

A. Background and Methodology

Previous research has demonstrated that hardware implementations of public cloud VM types change over time [1] [2]. Today, Amazon still hosts the original first generation instance types initially offered in 2007. As the original server hardware has long since been retired, new hardware was tasked with providing the equivalent VM types. This hardware replacement has led to a variety of implementations in Amazon EC2, especially for older generation VM types. Several hardware implementations of the same VM type may be offered at once, each with different performance characteristics. When hosting scientific modeling workloads on public clouds we are interested in understanding the performance implications of VM type heterogeneity. Can we exploit this heterogeneity to improve performance of scientific modeling workloads?

To investigate implications of VM host heterogeneity, VM-Scaler provides host CPU type enforcement as described earlier using the "forceCpuType" attribute. CPU type enforcement incurs the additional expense of launching and terminating unmatching VM instances. In Amazon EC2, discarded VMs are billed for 1-hour of usage; we used EC2 spot instances to minimize these costs.

B. Experimental Setup

In [1], VM host heterogeneity is observed for m1.small, m1.large, and m1.xlarge VMs across all Amazon EC2 east subregions. To extend earlier work, we tested host heterogeneity for 12 VM types: 1st generation VM types (m1.medium, m1.large, m1.xlarge, c1.medium, c1.xlarge), 2nd generation types (m2.xlarge, m2.2xlarge, and m2.4xlarge), and 3rd generation types (c3.large, c3.xlarge c3.2xlarge, m3.large). To investigate the prevalence of VM-host heterogeneity (RQ-1) we first launched 50 VMs for each of the 12 VM types. When heterogeneity was detected, we launched 50 more VMs for a total of 100. Tests were performed using two Amazon regions: us-east-1c and us-east-1d. Host heterogeneity was determined by VM-Scaler by inspecting Linux's /proc/cpuinfo proc-file.

To quantify the effect of host heterogeneity on model service performance (RQ-2), we studied the two VM types exhibiting the highest degree of heterogeneity. We completed batch model service workloads using WEPS and RUSLE2 on pools of 5 VMs for each implementation type. We performed 10 trials of 100 WEPS runs, and 10 trials of 660 RUSLE2 runs using these implementation-specific pools to compute average model service execution time.

We did not investigate VM host heterogeneity for very small VMs which do not receive 100% of a full CPU core allocation such as the bursting VMs (e.g. t1.small) and 1-core VMs (e.g. m1.small, m3.medium). These VM types provide minimal throughput for service hosting.

TABLE II. AMAZON VM HOST HETEROGENEITY

VM type	Region	Backing CPU	Backing CPU
m1.medium	us-east-1c	Intel E5-2650 v0 8c,95w,96%	Intel Xeon E5645 6c,80w,4%
m2.xlarge	us-east-1c	Intel Xeon X5550 4c, 95w, 48%	Intel Xeon E5-2665 v0 8c, 115w, 42%
m1.large	us-east-1d	Intel Xeon E5-2650 v0 8c,95w,74%	Intel Xeon E5-2651 v2 12c,105w,19%
m1.large	us-east-1d	Intel Xeon E5645 6c,80w,7%	--
m2.xlarge	us-east-1d	Intel Xeon E5-2665 v0 8c, 115w,78%	Intel Xeon X5550 4c, 95w, 22%

C. Experimental Results

We tested 12 VM types across 3 generations, and found significant VM host heterogeneity for 3 of the types (25%) from 2 generations while testing on two different Amazon regions (us-east-1c and us-east-1d). We discovered 7 implementations of these 3 VM types. On average, a given hardware configuration was found to support 43.3% of the VM instances for the m1.medium, m1.large, and m2.xlarge VM types as described in Table II. Five different Intel CPUs spanning two microarchitectures and four different sets of microcode were found to provide these implementations. We found that m1.large VMs were implemented with CPUs from two different Intel microarchitecture generations (Nehalem and Sandy Bridge) and three different microcode releases (Nahalem, Westmere, and Ivy Bridge). **This hardware heterogeneity is almost certain to provide variable performance under the label "m1.large".**

Our tests revealed implementation host heterogeneity for 1st and 2nd generation VMs. Where we did not observe VM-host heterogeneity this does not imply it doesn't exist, nor does it imply if it will exist in the future. As CPUs continue to evolve, and legacy hardware ages, we suspect to see host heterogeneity for 3rd, 4th and future VM generations. Migration to core dense, lower power CPUs in cloud data centers stands to save energy and lower datacenter costs for all public cloud providers.

Compared to Ou et al.'s results [1], we observed that some CPU types found in 2011 and 2012 have been replaced with lower power, core-dense CPUs. For example we observed m1.xlarge VM implementations using the 12-core Intel Xeon E5-2651 CPU. Higher core density CPUs should help save energy, and reduce resource contention, while enabling server real estate to expand. Previously m1.large VMs implemented using AMD Opteron's consumed as much as 42.5 watts per core! Implementations using the Intel Xeon E5-2651 v2 require only 8.75 watts per core. This amounts to just ~20% of the previous power requirement.

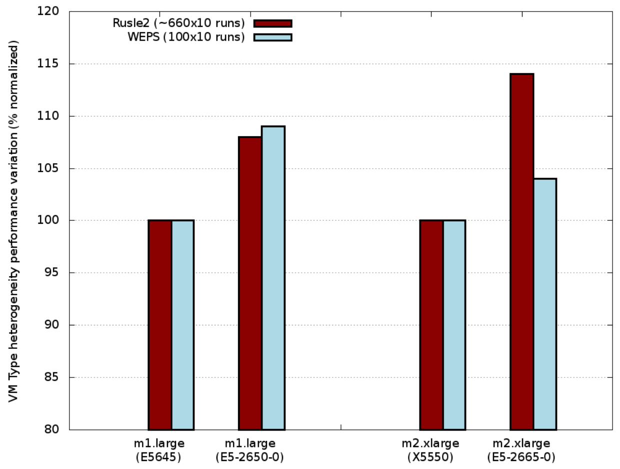

Fig. 2. Normalized Model Service Execution Time with Heterogeneous VM Implementations

Model service performance implications are shown in figure 2. For the m1.large VM implementations, model performance was slower when the VM was backed by the Intel Xeon E5-2650-v0. Average normalized execution time was 108% for RUSLE2, and 109% for WEPS versus VMs backed by the Intel Xeon E5645 CPU. For the m2.xlarge VM implementations, model performance was slower when the VM was backed with the Intel Xeon E5-2665-v0. Average normalized execution time was 114% for RUSLE2, and 104% for WEPS versus VMs backed by the older Intel Xeon X5550. Ironically, newer CPUs provided lower performance than their legacy counterparts in both cases.

VI. Identifying Resource Contention with CpuSteal

A. Background

Resource contention in a public cloud can lead to performance variability and degradation in a shared hosting environment [4] [6]. CpuSteal registers processor ticks when a VM's CPU core is ready to execute but the physical host CPU core is busy performing other work. The core may be unavailable because the hypervisor (e.g. Xen dom0) is executing native kernel mode instructions, or user mode instructions for other VMs. High cpuSteal time can be a symptom of over provisioning of the physical servers hosting VMs. On the Amazon EC2 public cloud, which uses a variant of the Xen hypervisor, we observe a number of factors that produce CpuSteal time. These include:

1. Processors are shared by too many VMs, and those VMs are busy.
2. The hypervisor kernel (Xen dom0) is occupying the CPU.
3. The VM's CPU time share allocation is less than 100% for one or more cores, though 100% is needed to execute a CPU intensive workload.

In the case of 3, we observe high cpuSteal time when executing workloads on Amazon EC2 VMs which under allocate CPU cores. A specific example is the m1.small and m3.medium VMs. We observed that the m3.medium VM type is allocated approximately 60% of a single core of the 10-core Xeon E5-2670 v2 CPU at 2.5 GHz. Consequently even an idle m3.medium VM produces cpuSteal. This observation explains the high cpuSteal values for m3.medium VMs studied in [15]. Because of this under allocation, all workloads executing at 100% on m3.medium VMs exhibit high cpuSteal because they must burst and use unallocated CPU time to reach 100%. These burst cycles are granted only if they are available, otherwise cpuSteal ticks are registered.

B. CpuSteal Noisy Neighbor Detection Method

Noisy neighbors are busy, co-located VMs that compete for resources that can adversely impact performance. We investigate the utility of using cpuSteal to detect resource contention from these "noisy neighbors" VMs using the Amazon EC2 public cloud. We use the CSIP-WEPS modeling service which features an average CPU utilization of 88% to detect Noisy Neighbors. We propose and investigate the use of the following "CpuSteal Noisy Neighbor Detection method" (NN-Detect):

Step 1. Execute a processor intensive batch workload across a pool of worker VMs several times. (WEPS)

Step 2. For each batch run, capture *cpuSteal* for each worker VM for the duration of the workload.

Step 3. Calculate the average and standard deviation of *cpuSteal* across the VM pool for the batch runs. ($cpuSteal_{avg}$).

Step 4. Identify the presence of noisy neighbor VMs by applying application agnostic and application specific thresholds to VM cpuSteal averages.

To detect when noisy neighbors share the same physical hosts, we screened for outliers using the application agnostic threshold:

$$\mathrm{cpuSteal}_{vm} \geq \mathrm{cpuSteal}_{avg} + (2 \bullet \mathrm{stdev})$$

To classify as having noisy neighbors we applied a second threshold:

$$\mathrm{cpuSteal}_{vm} \geq 10\ _{\mathrm{ticks}}$$

When a worker VM has fewer than 10 cpuSteal ticks this is considered too low to discern from noise. In these instances, this minor amount of resource contention, potentially from the hypervisor, is deemed insignificant and too low to degrade application performance. We validated these application agnostic thresholds by benchmarking representative workloads and qualitatively observing relationships between WEPS model performance and cpuSteal. Our approach using CpuSteal does not require the use of separate probe VMs as in [17] as workload execute and profiling occur simultaneously on the same VMs. We describe the results of our evaluation of NN-Detect using the WEPS model as the computational workload below.

C. Experimental Setup

To investigate the utility of cpuSteal for detecting resource contention from VM multi-tenancy (RQ-3) we evaluated NN-Detect using WEPS model service batch workloads as it is more CPU bound than RUSLE2 (figure 2). For step 1, we ran WEPS batch workloads on the VM pools described in table III. To enforce hardware homogeneity of our test environment we used the "forceCpuType" attribute of VM pools supported by VM-Scaler to guarantee every VM would be implemented with the same backing CPU. We launched 50 VMs for each VM type listed in table III, except for the 8-core c1.xlarge (25 VMs), and the 1-core m1.medium and m3.medium (60 VMs). We repeated 4 batch runs of approximately 1,000 randomly generated WEPS runs. Model runs were distributed evenly using HAProxy round-robin load balancing across all VMs in the pool. The 4 batch runs required a total of approximately 5 hours to complete.

For step 2, we totaled individual VM cpuSteal ticks for each VM in the worker pools for each WEPS batch job. For step 3, we calculated the average and standard deviation of cpuSteal for every VM across batch runs. We then looked for patterns in cpuSteal behavior for each of the 9 different VM pools from Table III. For step 4, we applied the thresholds described above using data from step 3 to identify VMs with noisy neighbors. To measure the performance implications of noisy neighbors (RQ-4) we created sub-pools of 5 VMs using the existing VMs. One sub-pool consisted of 5 VMs where cpuSteal exceeded the noisy-neighbor thresholds (step 4), and the other was 5 randomly selected VMs that did not exceed thresholds. We then completed 10 batches of 100 WEPS runs, and 10 batches of 660 RUSLE2 runs to compare performance.

D. *Experimental Results*

To address (RQ-3) we compared individual VM cpuSteal values from each batch run and used linear regression to test if one batch's cpuSteal behavior could predict cpuSteal for future batch runs. Statistically we wanted to know how much variance is explained by comparing individual VM cpuSteal values across batch runs. The averaged R^2 values from these comparisons appear in table IV. R^2 is a measure of prediction quality that describes the percentage of variance explained by the regression. The important discovery here is that over 5-hours, **early observations of cpuSteal were helpful to predict the future presence of cpuSteal!** Using this approach we can detect resource contention patterns that persist. We observe predictable patterns of VM cpuSteal behavior using Amazon EC2 for 4 VM types (m1.large, m2.xlarge, m1.xlarge, m1.medium) at ($R^2 > .44$), and as high as R^2=.94 for m1.xlarge.

To evaluate (RQ-4) we next calculated average and standard deviation scores of cpuSteal for the VM pools to determine threshold values to identify worker VMs with noisy neighbors. When no worker VMs exceeded these thresholds for a given type (e.g. c3.large), cpuSteal was not useful at detecting potential performance degradation from noisy neighbors. Table III shows the "% Noisy Neighbors" identified by NN-Detect. VMs without noisy neighbors tended to be hosted on physical hosts backed by higher density 10 and 12-core CPUs. (e.g. c3.large, m3.large, m3.medium, c1.xlarge). Modern 10 and 12-core Intel Xeon CPUs all feature hyper threading and new servers often utilize 2 or 4 CPUs per server. By increasing the number of available hyper threads in a dual-CPU server from 16 (X5550) to 48 (E5-2651) the likelihood of resource contention from multi-tenancy should drop to ¼ of the original amount. Such advances in hardware should help mitigate public cloud resource contention until demand increases to consume this additional capacity.

We applied NN-detect thresholds to create two mini-pools of 5 VM each. One pool consisted of VMs with noisy neighbors identified using NN-detect, and the other pool were randomly chosen worker VMs without noisy neighbors. We compared the performance of these mini-pools by executing WEPS and RUSLE batch runs as described earlier. The observed performance degradation for model service execution time with noisy neighbor VMs is shown in table IV. Performance is normalized vs. VM pools without noisy neighbors to show the increase in execution time resulting from noisy neighbors. Across 4 VM-types we observe average performance degradation up to 18% for WEPS and 25% for RUSLE2 using m1.large VMs. Three VM types (m1.large, m2.xlarge, m1.medium) produced statistically significant ($p > .05$) performance degradation for the repeated batch runs. The average performance degradation for RUSLE2 and WEPS model services for execution on VM pools with noisy neighbors was 9% across all tests.

TABLE III. AMAZON EC2 CPUSTEAL ANALYSIS

VM type	Backing CPU	Average R^2 linear reg.	Average *cpuSteal* per core	% with Noisy Neighbors
		us-east-1c		
c3.large-2c	E5-2680v2/10c	.1753	2.35	0%
m3.large-2c	E5-2670v2/10c	-	1.58	0%
m1.large-2c	E5-2650v0/8c	.5568	15.24	10%
m2.xlarge-2c	X5550/4c	.4490	953.25	6%
m1.xlarge-4c	E5-2651v2/12c	.9431	29.01	4%
m3.medium-1c	E5-2670v2/10c	.0646	17683.2[1]	n/a
c1.xlarge-8c	E5-2651v2/12c	.3658	1.86	0%
		us-east-1d		
m1.medium-1c	E5-2650v0/8c	.4545	6.2	7%
m2.xlarge-2c	E5-2665v0/8c	.0911	3.14	0%

[1] LESS THAN 100% CPU CORE ALLOCATION PRODUCES SIGNIFICANT *CPUSTEAL*

TABLE IV. EC2 NOISY NEIGHBOR MODEL SERVICE PERFORMANCE DEGRADATION[1]

VM type	Region	WEPS	RUSLE2
m1.large E5-2650v0/8c	us-east-1c	117.68% df=9.866 $p=6.847 \cdot 10^{-8}$	125.42% df=9.003 p=.016
m2.xlarge X5550/4c	us-east-1c	107.3% df=19.159 p=.05232	102.76% df=25.34 $p=1.73 \cdot 10^{-11}$
m1.xlarge E5-2651v2/12c	us-east-1c	100.73% df=9.54 p=.1456	102.91% n.s.
m1.medium E5-2650v0/8c	us-east-1d	111.6% df=13.459 $p=6.25 \cdot 10^{-8}$	104.32% df=9.196 $p=1.173 \cdot 10^{-5}$

[1] NORMALIZED MODEL SERVICE PERFORMANCE FOR WORKER VMS IS 100%.

To summarize our key findings: (1) a complete set of WEPS batch runs required up to 5 hours to complete. Throughout this time, trends in cpuSteal remained consistent to produce statistically significant model service performance degradation shown in table IV. And, (2) in cases where cpuSteal did not identify resource contention from noisy neighbors, it likely did not exist. In these cases the host hardware tended to be more core-dense. To conclude, our results suggest that resource contention and application performance degradation is more likely in public clouds when VMs are hosted using hardware with fewer CPU cores.

VII. CONCLUSIONS

Public cloud environments abstract the physical hardware implementation of resources providing users with limited details regarding the actual physical cluster implementations. For public cloud hosting of model services, the trial-and-better approach can improve model service performance and lower hosting costs. We tested 12 VM-types provided by Amazon EC2 and found that 25% of the types had more than one hardware implementation. (RQ-1) By leveraging the trial-and-better approach in VM-scaler we demonstrated potential for a

14% RUSLE2 model performance improvement (m2.xlarge) and 9% WEPS performance improvement (m1.large) by harnessing VM-host heterogeneity (RQ-2).

We developed an approach to find noisy neighbor VMs in a public cloud which cause unwanted resource contention by harnessing the cpuSteal CPU metric (NN-detect). We tested 9VM types across 2 regions in Amazon EC2 and found that 4 VM types had VMs which were able to measure increased levels of cpuSteal which persisted over several hours (RQ-3). We applied our NN-detect method to isolate pools of VMs with and without resource contention from noisy neighbors to measure performance degradation. Running batch workloads on pools consisting of VMs with high resource contention produced model service performance degradation up to 18% for WEPS and 25% for RUSLE2 (RQ-4).

Abstraction of physical hardware using IaaS clouds leads to the simplistic view that cloud resources are homogenous and that scaling will infinitely provide linear increases in performance because all resources are identical. Our results demonstrate how trial and better evaluation of VMs in public clouds can help mitigate both resource contention and address hardware heterogeneity to deliver performance improvements. We quantify the extent of hardware heterogeneity and resource contention in Amazon EC2's public cloud, and quantify performance implications for service-oriented application workloads. Our results provide simple and promising techniques that can be leveraged to improve workload performance leading to potential for cost savings in public clouds.

REFERENCES

[1] Z. Ou, H. Zhuang, A. Lukyanenko, J. Nurminen, P. Hui, V. Mazalov, A. Yla-Jaaski, Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds, IEEE Trans. on Cloud Computing, vol. 1, No. 2, July-Dec 2013, pp. 201-214.

[2] B. Farley, et al., More for Your Money: Exploiting Performance Heterogeneity in Public Clouds, Proc. 3rd ACM Int. Symp on Cloud Computing (SoCC '12), San Jose, CA, USA, Oct 14-17, 2012, 14 p.

[3] 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.

[4] M. Rehman, M. Sakr, Initial Findings for Provisioning Variation in Cloud Computing, Proc. of the IEEE 2nd Intl. Conf. on Cloud Computing Technology and Science (CloudCom '10), Indianapolis, IN, USA, Nov 30 - Dec 3, 2010, pp. 473-479.

[5] T. Ristenpart, E. Tromer, H. Shacham and S. Savage, "Hey, You, Get Off My Cloud! Exploring Information Leakage in Third- Party Compute Clouds, Proc. of the 16th ACM conf. on computer and communication security (CCS '09), Chicago, IL, USA, Nov. 9-13, 2009, pp. 199-212.

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

[7] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, A Performance Analysis of EC2 Cloud Computing Services for Scientific Computing, Proc of the 1st Int. Conf on Cloud Comuting (CloudComp '09), Munich, Germany, Oct. 19-21, 2009, pp. 115-131.

[8] E. Walker, Benchmarking Amazon EC2 for High-Performance Scientific Computing, USENIX; login: vol. 33, no.5, pp. 18-23., 2008.

[9] K. Jackson et al., Performance Analysis of High Performance Computing Applications on the Amazon Web Services Cloud, Proc. of the 2nd IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom '10), Indianapolis, IN, USA, Nov 30-Dec 3, 2010, oo. 159-168.

[10] Y. Zhai, Cloud versus In House Cluster: Evaluating Amazon Cluster Compute Instances for Running MPI Applications, Proc. State of Practice Reports (SC '11), Seattle, WA, USA, Nov 12-18, 2011, 10 p.

[11] P. Saripalli, C. Oldenburg, B. Walters, N. Radheshyam, Implementation and Usability Evaluation of a Cloud Platform for Scientific Computing as a Service (SCaaS), Proc of the 4th IEEE Int. Conf on Utility and Cloud Computing (UCC 2011), Melbourne, Australia, Nov 5-8, 2011, pp. 345-354.

[12] Villamizar, M., Castro, H., Mendez, D., E-Clouds: A SaaS Marketplace for Scientific Computing, Proc. of the 5th IEEE/ACM Int. Conf. on Utility and Cloud Computing (UCC 2012), Chicago, IL Nov 5-8, 2012, 8p.

[13] W. Lloyd et al., The Cloud Services Innovation Platform- Enabling Service-Based Environmental Modeling Using IaaS Cloud Comp. (iEMSs 2012)Int.Cong on Env.Model Softw., Germany, Jul 2012, 8 p.

[14] O. David et al., Model as a Service (MaaS) using the Cloud Services Innovation Platform (CSIP), iEMSs 2014 Int. Cong on Env.Modeling and Software, San Diego, CA, USA, Jun 2014, 8 p

[15] A. Ayodele, J. Rao, T. Boult, Performance Measurement and Interference Profiling in Multi-tenant Clouds, Proc. of the IEEE Conf. on Cloud Computing (Cloud 2015), New York, NY, 2015, pp. 941-949.

[16] H. Liu, A Measurement Study of Server Utilization in Public Clouds, Proc. 9th IEEE International Conference on Cloud and Green Computing (CAG'11), Sydney, Australia, Dec 2011, pp.435-442.

[17] J. Mukherjee, D. Krishnamurthy, J. Rolia, C. Hyser, Resource Contention Detection and Management for Consolidated Workloads, Proc. of the IFIP/IEEE Int. Symposium on Integrated Network Management (IM 2013), Ghent, Belgium, 2013, pp. 294-302.

[18] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, R. Bianchini, DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments, 2013 USENIX Annual Technical Conference (USENIX ATC '13), San Jose, CA, pp. 219-230.

[19] Y. Zhang, A. Juels, A., Oprea, M. Reiter, HomeAlone: Co-Residency Detection in the Cloud via Side-Channel Analysis, 2011 IEEE Symposium on Security and Privacy (SP 2011), Oakland, CA, pp. 313-328.

[20] V. Varadarajan, Y. Zhang, T. Ristenpart, N. Swift, A Placement Vulnerability Study in Multi-Tenant Public Clouds, 24th USENIX Security Symposium, Washington DC, USA, pp. 913-928.

[21] G. Casale, C. Ragusa, P. Parpas, A Feasibility Study of Host-Level Contention Detection by Guest Virtual Machines, Proc. of the 5th IEEE Int. Conf. on Cloud Computing Technology & Science (CloudCom 2013), Bristol, UK, 2013, pp. 152-157.

[22] 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

[23] 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.

[24] W. Lloyd et al., The Virtual Machine (VM) Scaler: An Infrastructure Manager Supporting Environmental Modeling on IaaS Clouds, Proc. iEMSs 2014 International Congress on Environmental Modeling and Software, San Diego, CA, USA, June 16-19, 2014, 8 p.

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

[26] 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.

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

[28] W. Lloyd, S. Pallickara, O. David, M. Arabi, K. Rojas, T. Wible, J. Ditty, Demystifying the Clouds: Harnessing Resource Utilization Models for Cost Effective Infrastructure Alternatives, IEEE Transactions on Cloud Computing, 14pp, IEEE: To appear, 2017, available online at: http://dx.doi.org/10.1109/TCC.2015.2430339.

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