Public Cloud Virtual Machine Co-residency: Prediction and Implications

Madhuri Suresh Sharma

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science

University of Washington

2022

Reading Committee:

Wes J. Lloyd

Matthew E. Tolentino

Program Authorized to Offer Degree:

Computer Science and Systems

University of Washington

Abstract

Public Cloud Virtual Machine Co-residency: Prediction and Implications

Madhuri Suresh Sharma

Chair of the Supervisory Committee:
Assistant Professor Wes J. Lloyd
School of Engineering and Technology

Infrastructure-as-a-Service (IaaS) cloud platforms provide virtual machines (VMs) on demand to users hosted on the public cloud using shared or private physical servers. Reserving VMs on private dedicated hosts is expensive compared to renting the same on public shared servers, thus public shared servers are a widely preferred option for application deployment by cloud consumers. Despite considerable efforts to improve performance of VMs on the public cloud, significant performance variation is still possible when a large number VMs share a single physical server. The information about VM co-residency is abstracted by the cloud provider and remains unknown to the cloud consumer. Due to this abstraction cloud consumers tend to make less informed decisions and choices when creating VMs on public clouds. This research

evaluates performance degradation due to resource contention and utilizes memory benchmarks as performance metrics to predict VM co-residency evaluated across different Amazon Elastic Compute Cloud (EC2) VM placement groups. We conducted performance experiments leveraging memory benchmarks, to study performance implications for memory intensive workloads executing on co-resident VMs. The benchmarking results obtained were used to train machine learning models to predict VM co-residency. The VM co-residency predictions are evaluated by launching VMs using EC2 placement groups, a feature which allows users to influence physical placement of VMs on EC2.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my advisor, Prof. Wes J. Lloyd for his invaluable guidance, constant motivation and support throughout my research and Master's study. I am grateful to Prof. Matthew E. Tolentino for his guidance during my research. I am equally thankful to all the professors and my seniors at the university for their invaluable advice and support during the course of my study.

I am extremely grateful to my husband, Sandeep for his love, care and support in every single step of my Master's study. I extend my gratitude to my parents for their unstinted support and motivation. This endeavor wouldn't have been possible without every minute adjustments made by my little daughter, special thanks to the little one.

Dedicated to my,

dear husband, Sandeep
and
lovely daughter, Samruddhi

Chapter 1. INTRODUCTION

1.1 VIRTUAL MACHINE CO-RESIDENCY

Infrastructure-as-a-Service cloud computing platforms provide virtual machines (VMs) on-demand to support deployment of software applications on the cloud. IaaS platforms encompass the entire deployment architecture inclusive of storage, compute, and networking, which is then managed and maintained by the IaaS cloud provider. The operating system, libraries, application, and application workloads, however, are managed by the cloud consumers. Cloud providers provide VMs to the consumers to support deployment of enterprise workloads along with a host of other services including load balancing, monitoring, billing, security, etc. Major IaaS platforms include Amazon Web Services Elastic Compute Cloud (AWS EC2), Google Compute Engine, Microsoft Azure Virtual Machines, and Rackspace to name a few.

Advancements in technology have enabled hardware virtualization of VM components including CPU, memory, network, and storage I/O. These developments have mitigated virtualization performance overhead, but not the resource contention caused by co-located VMs [1]. IaaS platforms such as AWS EC2 offer VMs hosted on public shared servers and also private dedicated servers, known as EC2 dedicated hosts on AWS [2]. Due to cost limitations and consumer demand, most users opt to host the majority of application infrastructure using public shared servers rather than private dedicated hosts. The low cost of public IaaS cloud platforms that share physical servers promotes the popularity of these platforms leading to increased user adoption. This popularity leads to increased demand and ultimately resource contention on public shared IaaS servers. Significant measures have been taken to reduce performance overhead from virtualization

for hosting applications on public clouds. Despite these efforts, performance variation is still common when co-located VMs share the same underlying physical servers. As the CPU density of modern cloud servers increases due to Moore's Law, the number of VMs sharing servers has increased, exacerbating this problem. From the 3rd to the 6th generation, the number of virtual CPU cores shared among VMs for the general-purpose AWS EC2 VMs (i.e., m3, m4, m5, m6 families) has increased by 480% from 40 to 192 increasing VM co-residency [3][4]. However, the number of VMs co-residing on shared public cloud servers is unknown as the cloud provider conceals these configuration details from users.

1.2 Research Motivation

To address the increasing co-residency of VMs on public IaaS platforms, this research addresses the problems of detecting resource contention and co-resident VMs. The cloud provider does not reveal the exact placement details of consumer's VMs, nor the exact number of co-resident VMs sharing a physical server. There have been several investigative attempts to determine VM co-residency and predictions of VMs on the public cloud [5], [6],[7] [8]. Most of these efforts focus specifically on predicting whether two individual VMs co-reside, rather than predicting the number of VMs that share a single physical server. Additionally, these efforts do not consider co-residency prediction using placement strategies. These shortcomings provided us motivation to address these problems by extending our existing research to improve VM co-residency predictions and to enhance our understanding on running memory benchmarks to predict VM co-residency.

In this research investigation, we leveraged the existing Python benchmarking suite from [6] to measure the memory performance of VMs deployed on private dedicated hosts and in the public cloud. We have extended the existing benchmarking suite to incorporate three memory stress

benchmarks, CacheBench, STREAM, and pmbench, microbenchmarks available from the Phoronix Test Suite [9] to investigate performance implications of memory contention from VM co-residency. The memory benchmarks are then utilized to train new machine learning models to predict the number of co-located VMs. Training data is obtained using the EC2 dedicated hosts where VM placement enables carefully controlling the number of VM placements. We focused our efforts on using dedicated hosts with a high number of shared vCPU cores, specifically the m5d.large (96 shared vCPUs) to enable comparison of our results with earlier work in the research group. Hosts with high numbers of vCPUs have been shown to exhibit large performance degradation from VM co-residency as shown in Figure 1.1 from [6]. Our models are then used to predict the number of co-located VMs in the public cloud by launching pools of spot VMs on the EC2 public cloud (e.g., pool size ~50 VMs). In addition, EC2 placement groups are investigated for their ability to influence VM placements.

1.3 VM CO-RESIDENCY MODEL

Han et al. investigated resource contention on EC2 by using the aforementioned Python benchmarking suite combining benchmarks that stress the CPU, disk, and network [6]. This suite automates parallel benchmark execution across VMs on Amazon EC2. The results were utilized to train machine learning models to predict the number of co-located VMs. Table 1.1 describes benchmarks included in the original test suite and the primary resource they stressed.

Table 1.1 Existing Benchmarks and Resources They Stress

Benchmarks	Primary Stressed Resources
Sysbench	CPU
Y-cruncher	CPU
Pgbench	CPU + Disk I/O
Iperf	Network I/O

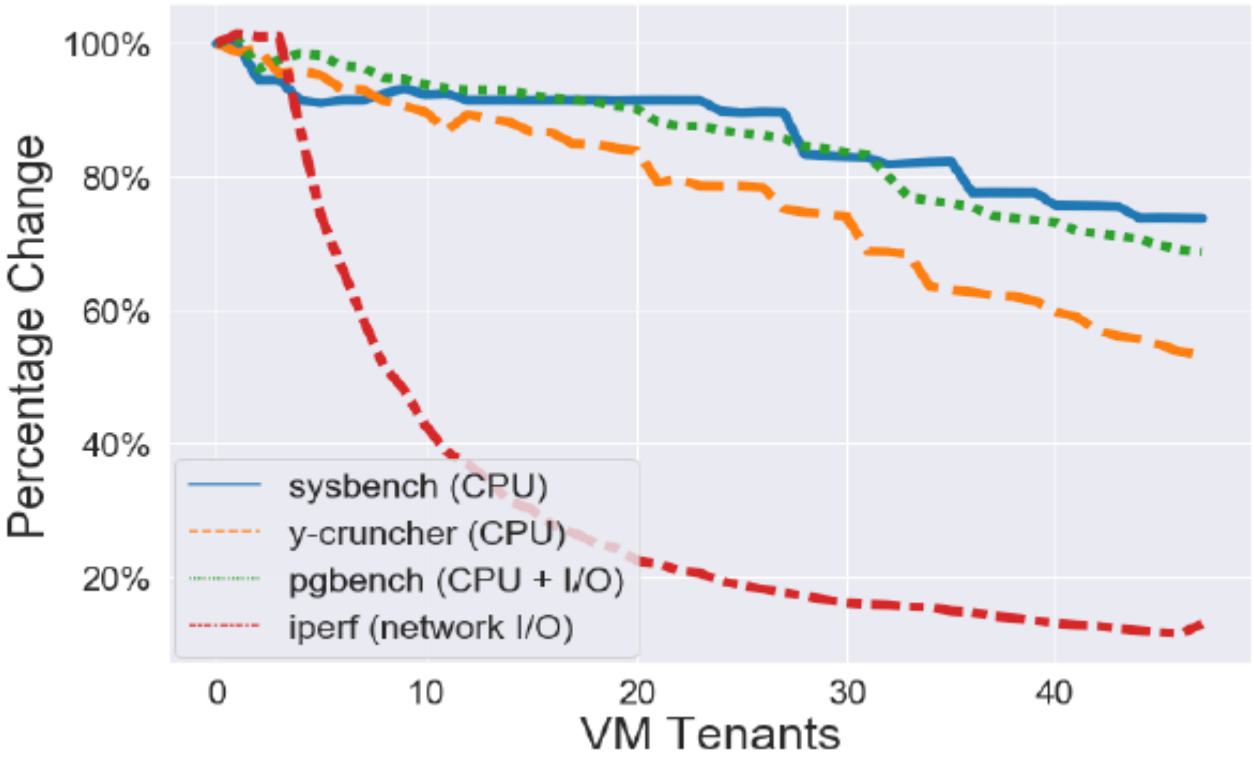

Figure 1.1 Normalized Performance Degradation from VM Co-location on m5 EC2 dedicated host with 96 virtual CPU cores (from Han, et al. [6])

This research investigates the utility of memory stress benchmarks to support VM co-residency detection. Table 1.2 describes the memory microbenchmarks that have been incorporated in the Python test suite.

Table 1.2 Memory Benchmarks Integrated in The Python benchmarking test suite

Benchmark	Microbenchmark Functionality
Pmbench	Profiles system paging performance by measuring memory access latencies during fault-intensive memory operations [10]
STREAM	Measures sustainable memory bandwidth and the corresponding computation rate for simple vector kernels [11]
CacheBench	A microbenchmark from LLCbench test suite, that tests the memory and bandwidth performance [12]

1.4 MOTIVATION BEHIND MEMORY STRESS BENCHMARKS

The idea behind choosing memory stress benchmarks is that memory stress is a potentially vulnerable characteristic where virtualization isolation may be imperfect creating an opportunity where we will be able to observe VM resource contention. The trend is that CPU isolation in virtualization systems (i.e., VMs) has become quite strong, making it harder to identify VM co-residency using CPU benchmarks. Memory stress benchmarks that exercise level 1, level 2, and level 3 CPU caches as well as the translation lookaside buffer (TLB) may be less resilient to the potential perturbation when a very large number of VM guests share a single physical host. This thesis investigates the utility of memory stress benchmarks to detect co-resident VMs in place of those from [6].

1.5 RESEARCH QUESTIONS

RQ-1: What are the performance implications of VM co-residency for memory benchmarks run in parallel across cloud VMs?

This project executed three memory benchmarks namely CacheBench, STREAM and pmbench, across VMs pools of up to 48 x m5d.large instances on an m5d EC2 dedicated host server. The results obtained have been used to analyze the memory implications of VM co-residency.

RQ-2: How effective are memory benchmarks and memory stress tests that are run in parallel across cloud VMs at inferring VM co-residency? What is their feature importance in models to predict VM-co residency?

Memory benchmarks are of specific interest in this research investigation, since the previous work in [6] focused on CPU, disk, and network bound benchmarks. This research has investigated performance effects of memory microbenchmarks when scaling the number of co-resident VMs. We compare available features derived from memory benchmarks to infer their feature importance in new VM co-residency prediction models.

RQ-3: Using the EC2 placement groups (e.g., spread, cluster, and partition) what degree of VM multi-tenancy is detectable?

EC2 offers general policies referred to as placement groups (i.e., spread, cluster, and partition) to influence where VMs are placed on the public cloud [13]. This research has leveraged EC2 placement groups to evaluate VM co-residency predictions. In particular, spread placement provides a guarantee of VM isolation by directing user VMs to be deployed to separate physical host servers. EC2 placement groups, however, do not isolate user VMs from other VMs that share the public cloud.

1.6 RESEARCH CONTRIBUTIONS

1. This project identifies performance implications for memory benchmarks run in parallel across co-resident VMs.
2. This project has integrated memory benchmarks into the Python test suite to support their use for the creation of new VM co-residency prediction models.
3. This project investigates the degree of VM co-residency using new VM co-residency models based on memory benchmarks to contrast with the existing models based on sysbench, pgbench, y-cruncher, and iPerf from [6]. Launching VMs using placement groups on EC2 can enable specific VM placement outcomes which can be harnessed to provide improved evaluations of our VM co-residency predictions [14].

Chapter 2. BACKGROUND AND RELATED WORK

2.1 VM Provisioning Variation and Resource Contention

Lloyd et al. [5] investigated public cloud hardware heterogeneity and detection of noisy neighbors to evaluate better selection of VMs and decide which VMs to retain in VM pools for hosting web service workloads in the public cloud. They quantified the performance degradation resulting due to resource heterogeneity and resource contention and benchmarked the performance of two environmental science modeling web service applications, RUSLE2 and WEPS, hosted on Amazon EC2 public cloud. They tested 12 VM types across 3 generations and found up to 25% host heterogeneity where VMs were created using alternate types of CPUs. They also developed a noisy neighbor detection approach called the NN-detect method, by leveraging cpuSteal metric. This method was tested and validated by running batch workloads on isolated VM pools consisting of VMs with high resource contention. As a result of this experiment, a performance degradation of around 25% for RUSLE2 and up to 18% for WEPS, was observed.

The research work of Ayodele et al. [7] is another evaluation of application performance due to resource contention. They investigated the impact of CPU allocation on application performance by calculating the interdependency of the CPU steal time and the application performance. They also introduced a Multi-tenant Performance Measurement and Interference Profiling system (MuPMIP), a Xen based Hypervisor cloud environment, that measures application performance at various interference points. By running benchmark workloads (microbenchmarks) in a multi-tenant environment workload slowdown time (performance degradation) is calculated. A mathematical relationship is deduced, workload slowdown time is given by the product of periodic interference loop and CPU steal time, workload slowdown time is represented and measured in

terms of CPU steal time. They have compared the benchmark workload performance on MuPMIP and Amazon EC2 m3. medium and have observed the performance impact to be 39.88% CPU steal time and 41.43% CPU steal time respectively. Their evaluation validates that the percentage of CPU steal time is directly linked to application performance and serves as an indicator for performance variation in the cloud.

Rehman et al. [15]investigated the effects of provisioning variation, a heterogeneity with respect to the number, size and inter VM locality on the cloud. They studied the impact of VM provisioning variation on application performance using suitable benchmarks and also demonstrated these effects on MapReduce workloads. Their research demonstrated provisioning variation for VM clusters provisioned on Qloud, an IaaS cloud and demonstrating a significant performance degradation of up to 5x for a MapReduce Sort application.

2.2 PERFORMANCE VARIATION

Schad et al. [16]studied performance variance of Cloud Infrastructure by utilizing CPU, I/O, and network benchmarks. They studied how performance variation on the cloud could impact a MapReduce application on a 50-node cluster. They analyzed that both large and small instances greatly suffered from performance variation, which was observed in their results which had, coefficient of variations (CV) of 24%, 20%, and 19% for CPU, I/O, and network performance respectively. Another key observation from their results is that the availability zones greatly influence performance variability.

Zhounghoung et al. [17] evaluate performance variation from hardware heterogeneity on Rackspace Cloud and Amazon EC2 by utilizing a set of microbenchmarks and application-level macro benchmarks. Three major observations the authors identified from this research are as

follows: the commonality of heterogeneous hardware in long-lasting cloud platforms, the fact that hardware heterogeneity contributes to performance variation on the cloud, and that varied CPU time shares and different virtual machine scheduling mechanisms intensified performance variation particularly for network related workloads. They compared the performance variation on Amazon EC2 and Rackspace, observing that the variation was around 20% for CPU performance to 268% for memory performance on EC2, and 15% for CPU, to 75% for disk on Rackspace with similar hardware. Finally, they proposed several cost saving approaches including, game theoretic analysis and Nash equilibrium from a cloud user's perspective. By implementing their trial-and-better approach on an application on EC2, they demonstrated a cost saving of up to 30%.

2.3 RESOURCE CONTENTION PREDICTION/STRATEGIES

Farley et al. [18]proposed a customer controlled VM placement gaming, a strategy enabling users to exploit performance heterogeneity and lower deployment costs. Their study was performed on Amazon EC2, where performance differences between identical instances were quite evident and thereby helping the researchers choose suitable targets for placement gaming. Their experiments were deployable without any assistance from the cloud provider. They developed a formal model of the strategy, which was tested and validated on EC2 instances, which yielded a significant performance improvement of 5% for a real-world CPU-bound job and 34% for a bandwidth-intensive job.

Yuquin qiu et al. [19]proposed a co-residency resistant VM deployment strategy and defined four thresholds to adjust the strategy for load balancing and security. They also introduced two deployment strategy evaluation metrics namely VM co-residency probability and user co-residency coverage probability. They validated their strategy by implementing it on two cloud

platforms namely OpenStack and CloudSim and observed that VM co-residency was reduced from 50% to 66.7%, and user co-residency from 50% to 66%, in comparison to the existing strategies.

2.4 VM Co-location Detection with Side Channels

Yu Si et al. [20] investigate a side channel attack on VMs and propose, VMs Co-residency Detection Scheme (VCDS) via cache-based side channel attacks, to get the location of a specified VM. They utilize a cubic spline interpolation-based load preprocessor, this enables VCDS to capture accurate and relevant raw measurements. VCDS probes accurate cache load changes produced by the affected VM, using a load-based predictor with a linear regression model. VCDS computes co-residency probability, based on the normal cloud model to quantitatively describe VM co-residency. Their experimental results show that compared to existing schemes, VCDS greatly improves the true detection rate despite interference of co-resident noisy VMs.

Yinqian Zhang et al. [21]proposed HomeAlone, a system that allows a tenant to verify a VM's exclusive use of a physical machine. It uses a side channel as a defensive detection tool. It analyzes the cache usage when friendly VM's coordinate with each other to avoid portions of cache, a tenant using HomeAlone can detect the activity of a malicious VM. The HomeAlone system includes classification techniques to analyze cache usage and guest operating system kernel modifications, to minimize the performance impact of genuine VMs sidestepping monitored portions of cache. HomeAlone is a standalone implementation on a XEN paravirtual machine and does not require further modifications to existing hypervisors or cooperation by cloud providers.

After a brief understanding of related and relevant research in the area of resource contention, it is evident that many studies have proposed explicit VMs that are probing side channel attacks or to evaluate performance variations, but in this research, we have leveraged common performance

benchmarks to train machine learning models to predict the VM co-residency. We have chosen memory benchmarks to characterize resource contention of VMs in the cloud. We predict the number of co-located VMs on CPU and memory dense hardware with 96 vCPUs which can host up to 48 co-located 2 vCPU virtual machines. Our model can suggest the health of the cloud environment with respect to memory contention, as well as VM co-residency. Our benchmarking approach takes ~125s to execute and obtain data from all the three memory benchmarks for model inferencing. The strategies proposed in this research can help in assessing and replacing underperforming VMs before running intense experiments with distributed workloads. This in turn improves the efficiency of the results and also lowers the experimental costs.

Chapter 3. METHODOLOGY

3.1 BENCHMARKING INFRASTRUCTURE

3.1.1 *Python benchmarking test suite*

In this research investigation, we have leveraged the existing Python benchmarking test suite from [6], which initially featured a combination of benchmarks that stress CPU, disk, and network. For the purpose of this research, we have integrated additional memory stress benchmarks into the test suite, namely CacheBench [22], STREAM [11], and pmbench [10]. The Python benchmarking test suite automates parallel execution of benchmarks across VMs on Amazon EC2 using Cron. The memory benchmarks were scheduled to run in parallel across Amazon EC2 instances provisioned in the open cloud and on dedicated hosts. For RQ-1, the main intention of parallel execution of memory benchmarks was to observe maximum performance degradation on the allowed EC2 host, which was achieved by stressing identical resources at the same time. For RQ-2, by executing benchmarks in parallel across VMs, we investigated the utility of using memory benchmarks as independent variables in predictive models to estimate the number of co-located VMs. To obtain measurements for each of our memory microbenchmarks for model inferencing using our benchmarking suite requires approximately ~125 seconds.

3.1.2 *CacheBench*

CacheBench is a benchmark that parametrizes the performance of multiple levels of cache present on and off the processor in computer systems. It measures the raw bandwidth in megabytes per second. The goal of CacheBench is to establish peak performance, i.e., computation rates alongside ensuring optimal cache reuse. This benchmark also aims to verify the effectiveness of high levels

of compiler optimization of tuned and untuned codes. CacheBench is highly effective for application performance modeling and prediction for applications that have been substantially tuned for cache reuse [12]. CacheBench benchmark comprises eight microbenchmarks, where each of them repeatedly accesses data items on varying vector lengths. The access time is recorded. The total amount of data accessed in bytes is calculated by multiplying the vector length and the number of iterations. The bandwidth is calculated by dividing the accessed data (bytes) by the total access time (seconds). The final bandwidth is reported in megabytes per second. In addition, the average access time per data item is reported in nanoseconds. The eight microbenchmarks are: 1. Cache Read, 2. Cache Write, 3. Cache Read/Modify/Write, 4. Hand tuned Cache Read, 5. Hand tuned Cache Write, 6. Hand tuned Cache Read/Modify/Write, 7. memset() from the C library, and 8. memcpy() from the C library.

For the purpose of this research, upon analyzing the performance degradation curves of three microbenchmarks, cachebench write indicated a gradual and smooth performance degradation as the VMs scaled from 1 to 48, hence CacheBench write has been chosen and integrated in the Python benchmarking test suite for the purpose of benchmarking.

3.1.3 *STREAM*

STREAM is a synthetic benchmark program that measures sustainable memory bandwidth (MB/s) and the corresponding computation rate for simple vector kernels. With advancements in technology, CPU computational performance is rapidly increasing compared to the memory bandwidth of the systems, which is a major limitation for programs and applications. The STREAM benchmark is designed to work with data systems larger than the available cache on any system, to evaluate the performance of large vector style applications. STREAM is a useful

component of models that scale homogeneous throughput workloads (e.g., SPEC CPU “rate” benchmarks).

STREAM measures the performance of four long vector operations namely ‘Copy’, ‘Scale’, ’Add’, ’Triad’. These operations are the basic/building blocks of long vector operations. The array sizes are defined in a way such that they are larger than the cache of the machine that is tested and the code ensures that there is no data re-use.

For the purposes of this research, STREAM is integrated in the Python benchmarking test suite and the performance throughput of Copy, Scale, Add, and Triad is recorded and reported.

3.1.4 *Pmbench*

Pmbench [10]is a multi-platform synthetic microbenchmark that profiles system paging characteristics by measuring the latency of paging related memory access operations. Pmbench profiles system paging performance by collecting memory access latencies while systematically consuming large volumes of memory to induce paging activity. It measures the memory access time in microseconds and records the results in the form of a histogram of the measured latencies.

3.1.5 *Phoronix Test Suite*

The Phoronix Test Suite [9]is a comprehensive testing and automated benchmarking suite and consists of various microbenchmarks and test profiles. It easily automates the entire testing process, including dependency management, download, and installation of the test and aggregation of results. In this research investigation, we have used Phoronix Test Suite as a third-party tool to schedule and run pmbench. The test configurations and result parsing are implemented using a bash script integrated in the above-mentioned Python benchmarking test suite.

3.1.6 *Integration into Python benchmarking test suite*

The Python benchmarking test suite consists of Python 3 scripts to automate tests and shell scripts to execute Linux commands. The generated results were stored in CSV files, which were later analyzed using scripts written in the R programming language and Python in Jupyter notebooks. This project leveraged several Python libraries and packages to schedule tasks on EC2 instances in the cloud, parse the benchmarking results, and generate output in CSV format. Bash scripts were extensively used to configure benchmarks, schedule tests through the Phoronix test suite, and parse the benchmark results. Cron synchronized tests to run concurrently across EC2 instances. OS clocks were synchronized using the network time protocol (ntp). All VMs were provisioned prior to the execution of benchmarks using the Python benchmarking test suite.

3.2 BENCHMARKING INFRASTRUCTURE

Experiments to investigate resource contention implications of running memory benchmarks in parallel (RQ-1), and experiments to evaluate memory benchmarks as features in our VM co-residency models (RQ-2) were performed on AWS EC2 Dedicated Hosts. For RQ-3, we leveraged EC2 spot instances provisioned on the open cloud in the US East (Northern Virginia) region.

3.2.1 *Dedicated Hosts*

Amazon EC2 Dedicated Hosts offer private physical servers featuring full isolation from other users to support execution of customer workloads. End users have the flexibility and control on the dedicated host allocation and capacity utilization. In this research to investigate resource contention, we performed experiments by provisioning the maximum number of supported m5d.large EC2 instances allowed on an EC2 dedicated host.

3.2.2 *Amazon EC2*

Amazon Elastic Compute Cloud (EC2) is an Infrastructure-as-a-Service platform, which provides a wide range of instance types to choose from, depending on the customer needs like CPU, memory, storage, and networking capacity. EC2 instances come in varied sizes, offering elasticity and scalability of resources based on customer requirements [4].

We investigated Amazon EC2 m5d instances [23] the latest generation of general-purpose instances powered by Intel Xeon Platinum processors. This instance family provides a balance of compute, memory, and network resources. This instance family features a burst frequency up to 3.5 GHz using either of two Intel Xeon Scalable processors (Skylake 8175M or Cascade Lake 8259CL) with the Intel Advanced Vector Extension (AVX-512) instruction set. The physical host servers feature 96 vCPUs. The instances are powered by the AWS Nitro System, which combines dedicated hardware with a lightweight hypervisor. All the m5d.large instances have local NVMe-based SSD block level storage physically connected to the host server. This family of EC2 instances offers a network bandwidth of up to 10 Gbps and EBS bandwidth of up to 4750 Mbps.

AWS offers EC2 Spot instances which are available up to a 90% discount compared to on-demand pricing. Spot instances are highly effective in managing fault-tolerant applications, containerized workloads, and high-performance computing applications. They allow great flexibility in launching and maintaining applications.

3.2.3 *EC2 Placement Groups*

EC2 Placement Groups greatly influence the placement of instances in such a way that interdependent instances with similar workloads are placed strategically to avoid any correlated failures. There are three placement groups to choose from: Cluster (i.e., instances are packed closely in an availability zone), Spread (i.e., instances are spread in small groups across distinct underlying hardware), and Partition (i.e., provisions a user's instances by spreading across different logical partitions, alongside ensuring hardware distinction between the partitions). In this thesis investigation, we leverage placement groups to verify public cloud VM co-residency predictions. Cloud providers do not expose the physical location of VMs. EC2 placement groups offer the potential to control and suggest when public cloud VMs may or may not be co-located. For RQ-3, we leverage EC2 placement groups to support evaluation of VM co-residency model predictions made for pools of VMs launched on the public cloud. We performed experiments by launching pools of 50 VMs on the public cloud using a cluster placement group, partition placement group, and with no placement group. We also launched 35 VMs using the spread placement group. AWS limits users to creating a maximum of 7 VMs per availability zone using the spread placement group. For our testing, we were only able to launch VMs in 5 out of 6 us-east availability zones, limiting our test to 35 VMs.

Chapter 4. EXPERIMENTAL RESULTS

4.1 MEMORY RESOURCE CONTENTION

This thesis investigates the performance implications of running memory benchmarks in parallel across VMs and also studies their utility as predictors to infer VM co-residency in the public cloud. We leverage a Python benchmarking test suite to schedule benchmarks to run in parallel across all provisioned VMs on AWS. Three memory benchmarks have been added to the test suite for this project: CacheBench, STREAM, and pmbench. To investigate RQ-1 and RQ-2, we scheduled benchmarks to run across 48 co-located m5d.large EC2 instances on a dedicated host. We investigated performance implications of running memory benchmarks in parallel using a varying number of co-located VMs scaling from 48 to 1 on the dedicated host using two experiments. In the first experiment (experiment a), we ran benchmarks using a varying number of VMs on the host where VMs that did not run the benchmark remained idle. In the second experiment (experiment b) VMs that did not run the benchmark were shut down, so that all remaining VMs on the dedicated host ran the benchmark. The experimental results from the experiments are analyzed and discussed below.

4.1.1 *CacheBench Result Analysis*

CacheBench benchmark has three microbenchmarks namely CacheBench read, CacheBench write, and CacheBench read-modify-write. We evaluated the performance variation of CacheBench by running all three CacheBench microbenchmarks on a dedicated host with and without stopping idle VMs. It required approximately 50 minutes to complete the entire experiment scaling from 48 to 1 VM.

Figure 4.1 depicts the performance variation as measured by the CacheBench read microbenchmark in terms of percentage change across a pool of 48 VMs on a dedicated host, conducted by stopping idle VMs. Figure 4.2 and Figure 4.3 follow the same experimental approach and visualization but with the benchmarks, CacheBench write and CacheBench read-modify-write respectively. The average throughput is calculated at each tenancy level from 48 down to 1. The percentage change in performance is calculated by normalizing the values. This percentage change for 4 throughput rates on the y-axis has been plotted against the VM tenancy from 1 to 48 on the x-axis. CacheBench read results are depicted in Figure 4.1with performance degradation curves for the 4 throughput rates (e.g., 1KiB, 1 MiB, 1 GiB, and 4 GiB), whereas CacheBench writes are shown in Figure 4.2. For CacheBench read-modify-write a slight and steady degradation for 1 KiB and 1MiB is shown, but a steep curve indicates a rapid degradation in performance as the VM tenancy increased from 1 to 48. Using CacheBench, we observed a clear degradation in performance as the VM tenancy increased. The graphs clearly depict this variation where normalized performance is 100% for a single VM tenant and the degradation increases as VM tenancy increases to 48. The CacheBench benchmark appears to be an appropriate benchmark to measure and analyze memory resource contention, which is evident from the graphs (Figures 4.1, 4.2, and 4.3).

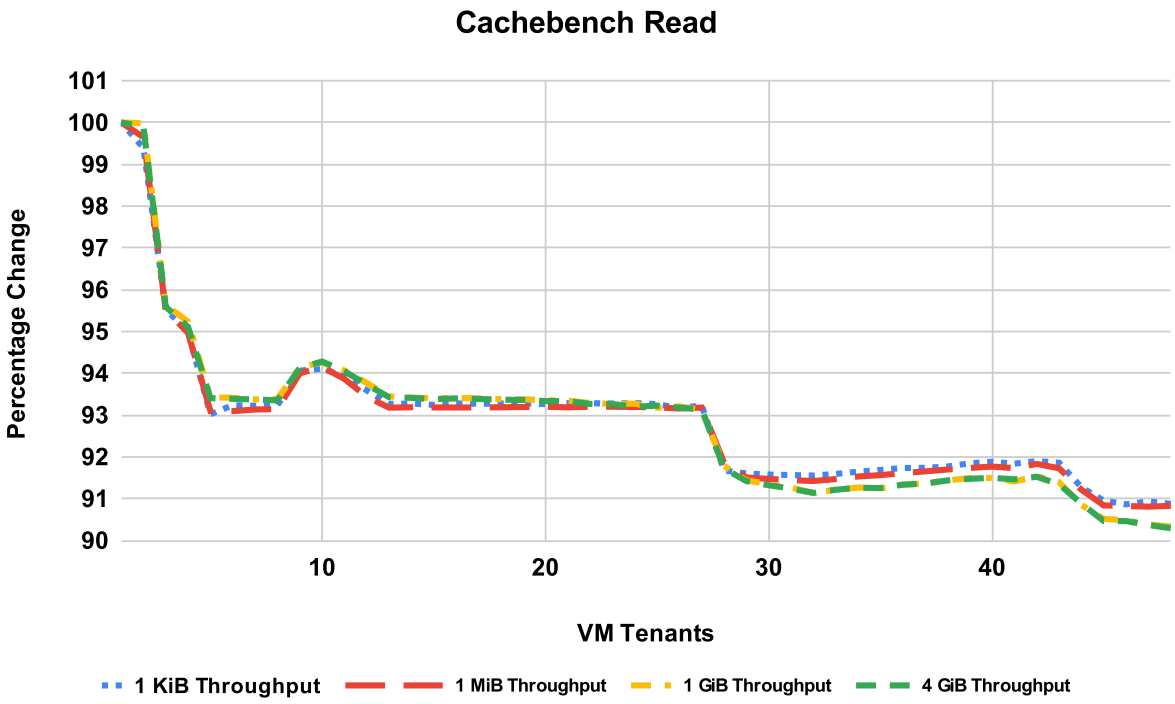

Figure 4.1 Normalized CacheBench Read benchmark
performance measured across 1 to 48 co-located m5d EC2 instances

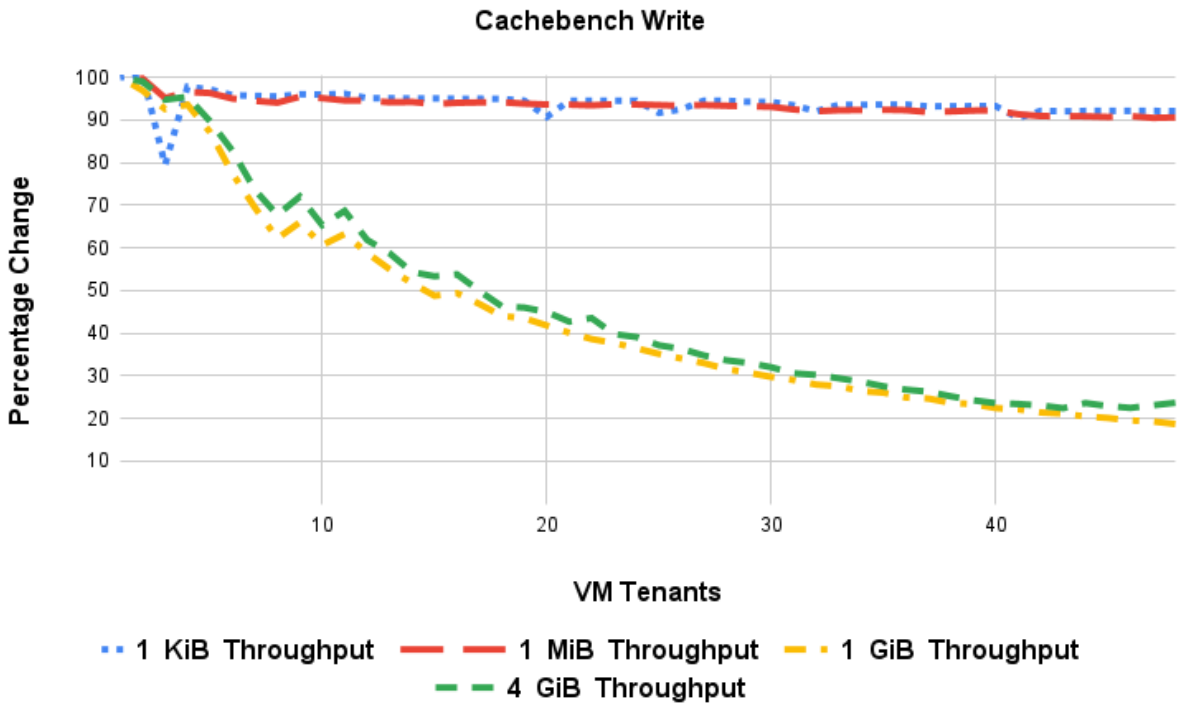

Figure 4.2 Normalized CacheBench Write benchmark performance measured across 1 to 48 co-located m5d EC2 instances

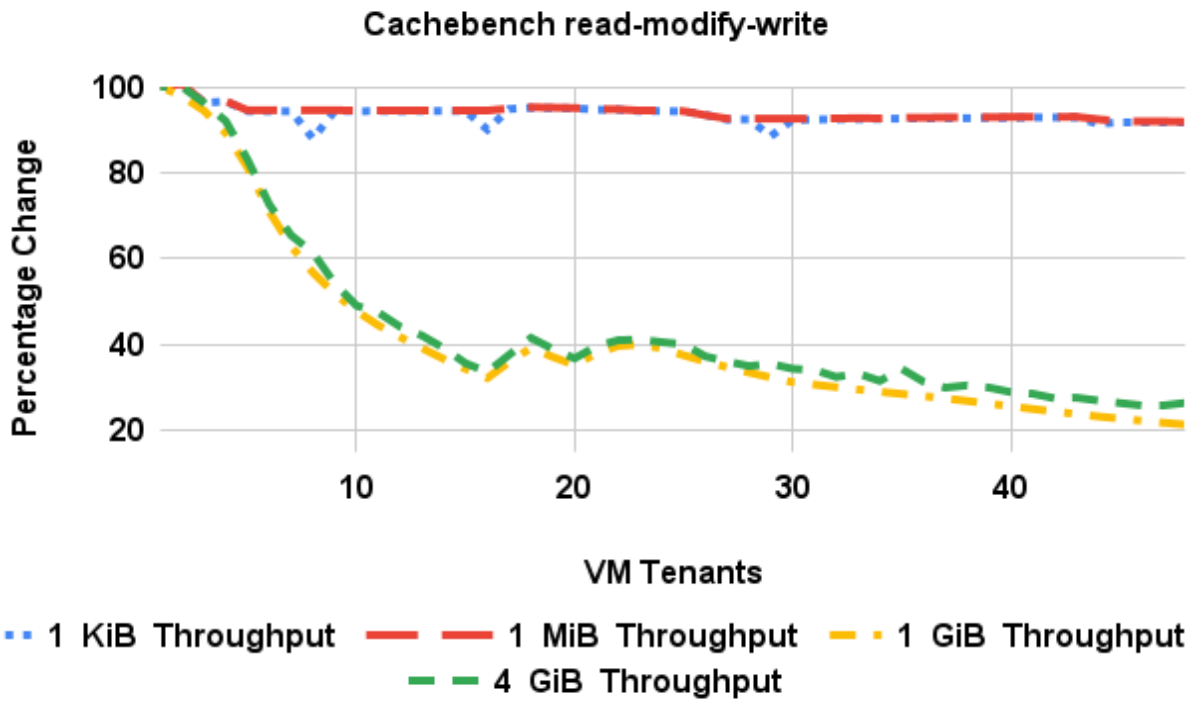

Figure 4.3 Normalized CacheBench Read-modify-write benchmark performance measured across 1 to 48 co-located m5d EC2 instances

4.1.2 *STREAM Result Analysis*

The STREAM benchmark experiment was conducted on a dedicated host with 48 x 2 vCPU m5d.large VMs. This benchmark took approximately ~1s to run on a single VM. To increase the stress and to obtain accurate results, for a single experiment STREAM was run 15 times in succession. On a dedicated host we then orchestrated STREAM to run scaling from 48 to 1 x m5d.large instances. This overall STREAM experiment required approximately 50 minutes to complete. The average computational rate (MB/s), minimum time (in seconds), maximum time (in seconds), and average time (in seconds) for the vector operations Copy, Scale, Add, and Triad were recorded and analyzed. Figure 4.4 shows a graphical representation of normalized average computation rates for the four vector operations plotted against VM tenancy. From the graph, it's evident that the performance degradation clearly increases as host VM tenancy scaled from 1 to 48. Running the STREAM benchmark by shutting down idle VMs improved performance by 2%, 4%, 7%, and 4% for STREAM Copy, Scale, Add, and Triad respectively. **The STREAM benchmark was useful in measuring and analyzing memory resource contention.**

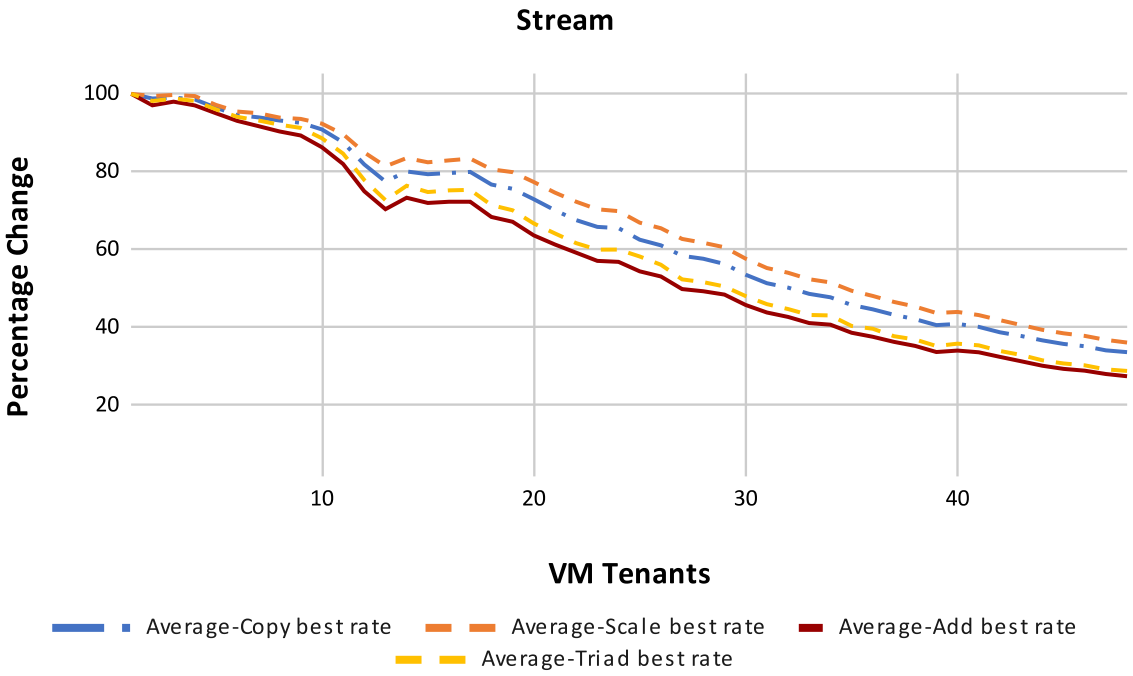

Figure 4.4 Normalized STREAM benchmark performance measured across 1 to 48 co-located m5d EC2 instance

4.1.3 *Pmbench Result Analysis*

The Pmbench experiment was performed on a dedicated host with 48 x 2 vCPU m5d.large VMs. The average page latency is calculated and reported by the pmbench benchmark. The benchmark required approximately 70s for single run on a VM. To orchestrate a pmbench scaling experiment on a dedicated host from 48 to 1x m5d.large instances required 100 minutes to complete. The normalized performance degradation of average page latency (microseconds) is calculated and plotted against VM tenancy and is shown in Figure 4.5. The graph depicts the normalized average page latency as the VM tenancy is increased from 1 to 48. Higher values indicate higher memory page latency and lower memory performance. The performance variation of 1 to 48 VMs by stopping idle VMs is 118% which shows higher memory page latency. Pmbench demonstrated resource contention at higher VM tenancy through higher memory access latency. **The Pmbench benchmark was an effective benchmark to observe and record memory resource contention on cloud platforms.**

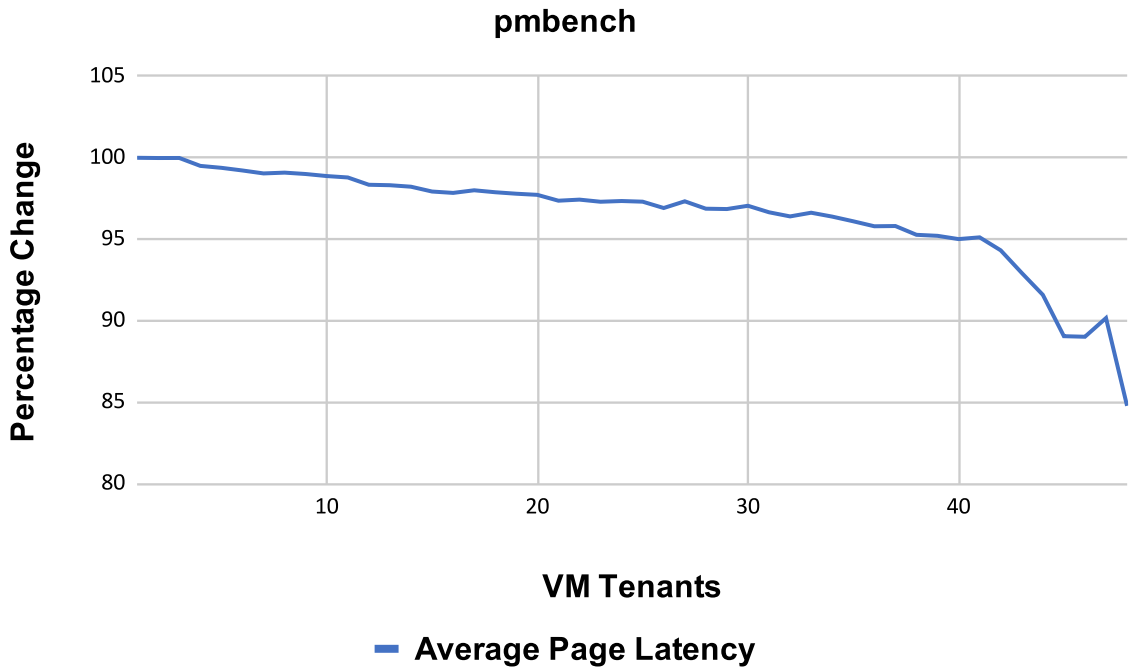

Figure 4.5 Normalized pmbench benchmark performance
measured across 1 to 48 co-located m5d EC2 instances
Note: for memory page access latency, higher values indicate lower memory performance

4.1.4 *CacheBench Performance Implications of Idle Virtual Machines*

Table 4.1 provides the observed performance variation for the CacheBench benchmark performed on a dedicated host when scaling from 48 to 1 VM running where VMs not running the benchmark were idle. Table 4.2 indicates the same experiment as earlier, but where idle VMs were stopped. Table 4.1 illustrates that the performance degradation of CacheBench read was very minute. Upon comparison of the performance of the other two CacheBench microbenchmarks, with idle VMs, CacheBench write produced a best-to-worst case performance difference of 81.34% for 1 GiB throughput and 79.23% for 4 GiB throughput. CacheBench read-modify-write produced a best-to-worst case performance difference of 77.91% for 1 GiB throughput and 73.97% for 4 GiB throughput.

By stopping the idle VMs, the performance degradation was slightly more and was also evident for 1 KiB and 1 MiB throughput tests. As shown in Table 4.2, CacheBench read had a slight dip of approximately 8-10%. CacheBench read produced a best-to-worst case difference of 9.67% for 1 GiB throughput and 9.71% for 4 GiB throughput. CacheBench write produced a best-to-worst case difference of 81.34% for 1 GiB throughput and 79.23% for 4 GiB throughput. CacheBench read-modify-write produced a best-to-worst case difference of 77.91% for 1 GiB throughput and 73.97% for 4 GiB throughput

Table 4.1 Performance Variation Results of CacheBench
Experiment Conducted on 48 VMs without Stopping Idle VMs

Without stopping Idle VMs (48 Runs)				
Benchmark	**1KiBThroughput**	**1MiBThroughput**	**1GiBThroughput**	**4GiBThroughput**
CacheBench read	100.68%	100.71%	99.92%	99.93%
CacheBench write	102.67%	101.95%	18.66%	20.77%
CacheBench read-modify-write	102.21%	102.64%	22.09%	26.03%

Table 4.2 Performance Variation Results of CacheBench
Experiment Conducted on 48 VMs by Stopping Idle VMs

Stopping Idle VMs (48 Runs)				
Benchmark	**1KiBThroughput**	**1MiBThroughput**	**1GiBThroughput**	**4GiBThroughput**
CacheBench read	90.87%	90.83%	90.33%	90.29%
CacheBench write	92.07%	90.60%	18.65%	23.64%
CacheBench read-modify-write	91.69%	91.84%	21.38%	26.43%

4.1.5 *STREAM Performance Implications of Idle Virtual Machines*

Table 4.3 indicates the performance variation results of the STREAM benchmark conducted on a dedicated host with 48 x 2vCPU m5d.large VMs, without stopping idle VMs. The VMs were scaled from 1 to 48 and the results tabulated. STREAM Copy, Scale, Add, and Triad produced a best-to-worst case throughput difference of 64%, 63%, 66%, and 67% respectively.

Table 4.4 indicates the performance variation results of the STREAM benchmark conducted on a dedicated host with 48 x 2 vCPU m5d.large VMs, by stopping idle VMs. The VMs are scaled down from 48 to 1. STREAM Copy, Scale, Add, and Triad produced a best-to-worst case throughput difference of 66%, 67%, 73%, and 71% respectively.

The performance delta increased by 2%, 4%, 7%, and 4% for Stream Copy, Scale, Add, and Triad respectively as the idle VMs are shutdown. These results indicate that running idle VMs which reserve memory added to the overall stress on the dedicated host leading to greater resource contention. Shutting down the idle VMs enabled a performance improvement for the STREAM benchmark.

Table 4.3 Performance Variation Results of STREAM
Experiment Conducted on 48 VMs Without Stopping Idle VMs

Without stopping Idle VMs (48 Runs)				
Benchmark	**Best Rate**	**Average Time**	**Minimum Time**	**Maximum Time**
STREAM Copy	36%	268%	254%	247%
STREAM Scale	37%	260%	244%	240%
STREAM Add	34%	285%	270%	261%
STREAM Triad	33%	296%	278%	272%

Table 4.4 Performance Variation results of STREAM
experiment conducted on 48 VMs by stopping idle VMs

Stopping Idle VMs (48 Runs)				
Benchmark	**Best Rate**	**Average Time**	**Minimum Time**	**Maximum Time**
STREAM Copy	34%	324%	307%	295%
STREAM Scale	36%	302%	285%	276%
STREAM Add	27%	405%	381%	366%
STREAM Triad	29%	385%	362%	357%

4.1.6 *pmbench Performance Implications of Idle Virtual Machines*

The Pmbench benchmark was executed on a dedicated host with 2 x 48 m5d.large VMs producing a best-to-worst case performance difference of 11% without stopping idle VMs, and 18% when stopping idle VMs. Upon stopping idle VMs, the performance delta increased by 7% indicating the presence of more memory access latency when idle VMs were present. We observed that by stopping idle VMs, memory access latency improved. This implies that the presence of idle VMs increased memory access latency as the physical host had to share physical memory across multiple guests leading to higher resource contention.

Table 4.5 Consolidated performance results of pmbench experiment conducted on 48 VMs

Benchmark	Without stopping Idle VMs (48 Runs)	Stopping Idle VMs (48 Runs)
pmbench	111%	118%

4.1.7 *Comparison and Analysis of Memory Resource Contention*

From the above analysis of graphs and comparison of performance of 48 VMs on a dedicated host with two different states of idle VMs, we are able to arrive at the conclusions for ***RQ-1:*** *What are the performance implications of VM co-residency for memory benchmarks run in parallel across cloud VMs?* **- the performance of memory benchmarks declines as the number of co-resident VMs increase from n = 1 to 48. This is the expected performance trend, which clearly implies resource contention due to presence of co-located VM's accessing the same hardware.**

Another interesting observation was that performance degradation was higher in the presence of idle VMs but was less in experiments where idle VMs were shutdown. CacheBench write and CacheBench read-modify-write provided clear indicators of performance degradation with increasing VM tenancy. STREAM also exhibited a large performance degradation. To summarize, all three benchmarks clearly exhibited performance degradation with increased VM tenancy. Considering the above results, CacheBench write, STREAM, and pmbench were then considered for our machine learning models to investigate RQ-2 to predict the VM co-residency.

4.2 Coefficient of Variation

The analysis of Coefficient of Variation (CV) addresses RQ-2*: How effective are memory benchmarks and memory stress tests that are run in parallel across cloud VMs at inferring VM co-residency? What is their feature importance in models to predict VM-co residency?* An interesting observation in the dataset reveals that the throughput values are oscillating between two different groups of values. The coefficient of variation (CV) for the benchmark metrics, was calculated by dividing the standard deviation of the throughput values by the mean of the values. The resulting quotient is multiplied by 100 to give the CV a percentage. Figures 4.6, 4.7, 4.8, 4.9, 4.10, and 4.11 show the CV of throughput values for the CacheBench write, pmbench, STREAM Copy, STREAM Scale, STREAM Add, and STREAM Triad respectively. While CV of 1-3% for a benchmark running on a dedicated server is reasonable, we observed high CV values in excess of 20% for high throughput CacheBench tests as well as for STREAM Copy, STREAM Scale, STREAM Add, and STREAM Triad. Upon closer inspection of the data, for these tests with greater than 10 co-located VMs, we observed strong bimodality in raw data where benchmark

performance appeared to diverge into two separate classes with values centered around two different means. Our graphs show that memory benchmarks are volatile and demonstrate a bimodal performance behavior when many co-located VMs compete for physical memory and cache resources. The existence of this bimodal data reduced the efficacy of these predictors for use in machine learning models resulting in predictions that weren't as accurate as desired. The 1 GiB and 4 GiB CacheBench write throughput curves in Figure 4.6, STREAM Copy in Figure 4.8, STREAM Scale in Figure 4.9, STREAM Add in Figure 4.10, and STREAM Triad in Figure 4.11 clearly indicate a higher degree of variation which is evident from the curves in the graphs. Upon closer inspection, the 1 KiB and 1 MiB throughput CV for CacheBench Write in Figure 4.6, and the average page latency CV for pmbench shown in Figure 4.7 were the most stable indicating less variation. These memory benchmarks did not appear to exhibit bimodal behavior. Hence, we leveraged the 1 KiB and 1 MiB throughput values from CacheBench and average page latency from pmbench as features for the machine learning model for RQ-2.

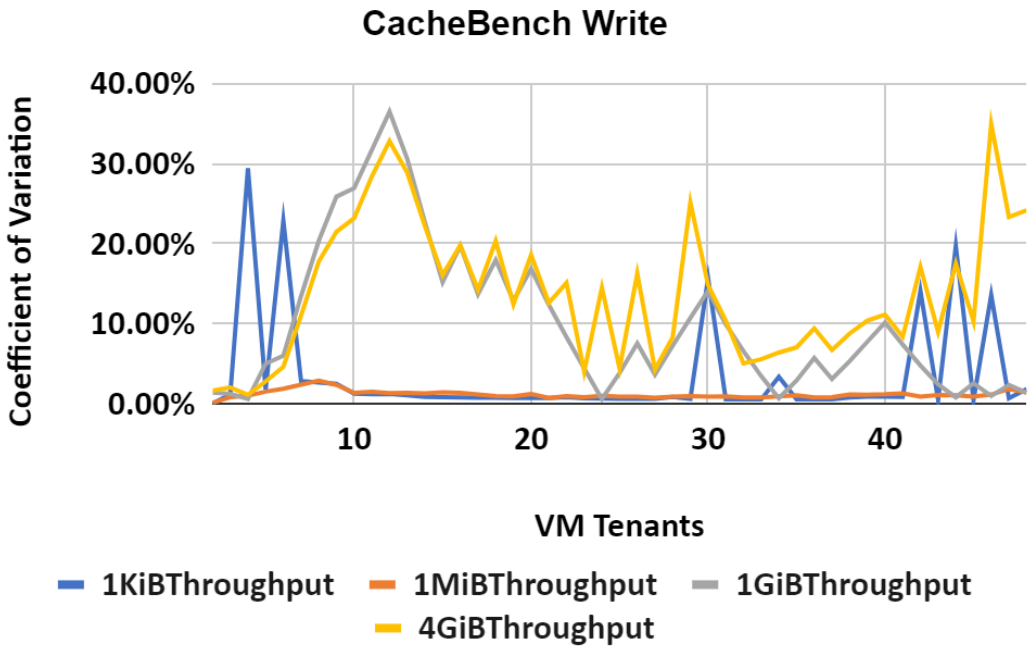

Figure 4.6 Coefficient of Variation of CacheBench write dataset with increasing VM tenancy

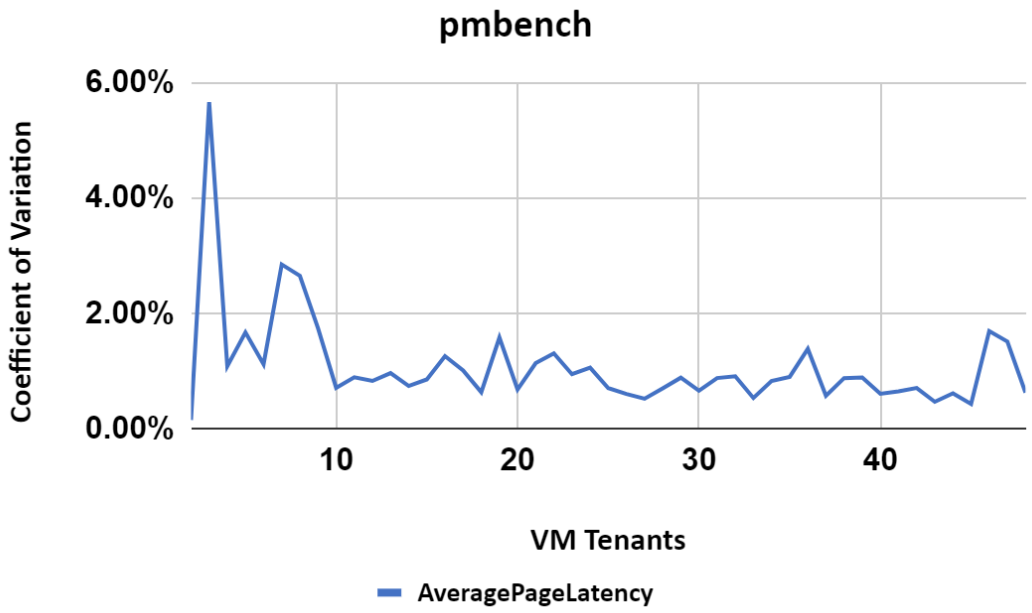

Figure 4.7 Coefficient of Variation of pmbench dataset with increasing VM tenancy

The high CV seen on the left-hand side of Figure 4.7 is because the standard deviation and average are calculated over a very small number of samples. With just 2 VMs, there are only 2 pmbench performance values available to calculate the average and standard deviation. With 3 VMs, there are just 3 pmbench performance values, and so on. This higher CV rapidly levels off as the number of pmbench tests increases. CV remains below 2% when the number of VMs is 10 or higher.

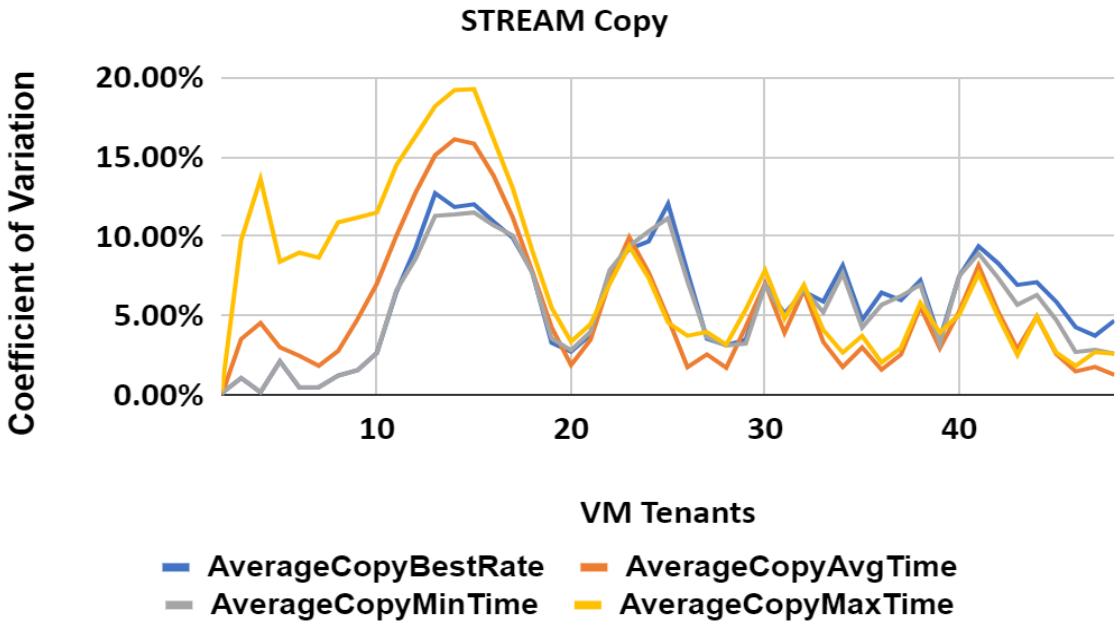

Figure 4.8 Coefficient of Variation of STREAM copy dataset with increasing VM tenancy

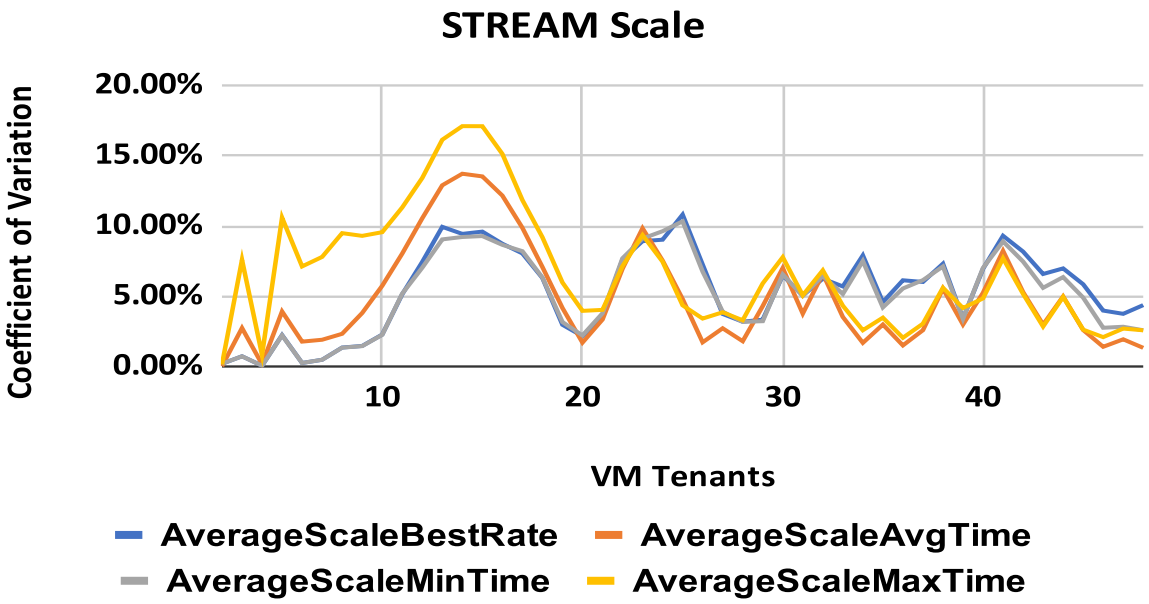

Figure 4.9 Coefficient of Variation of STREAM Scale dataset with increasing VM tenancy

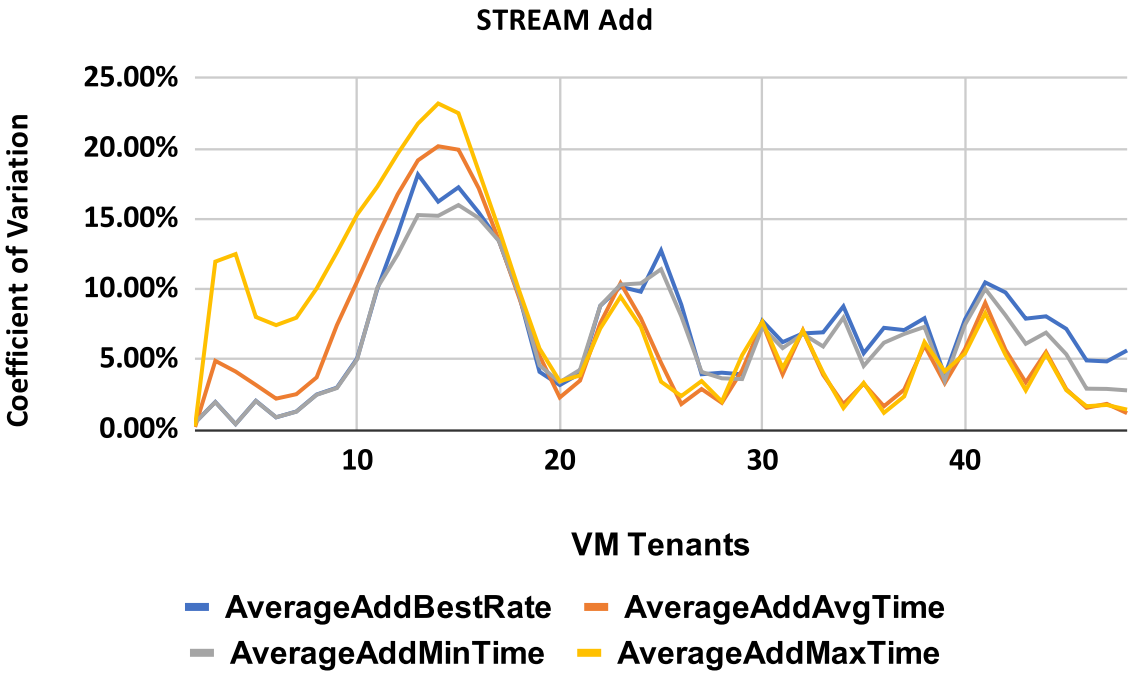

Figure 4.10 Coefficient of Variation of STREAM Add dataset with increasing VM tenancy

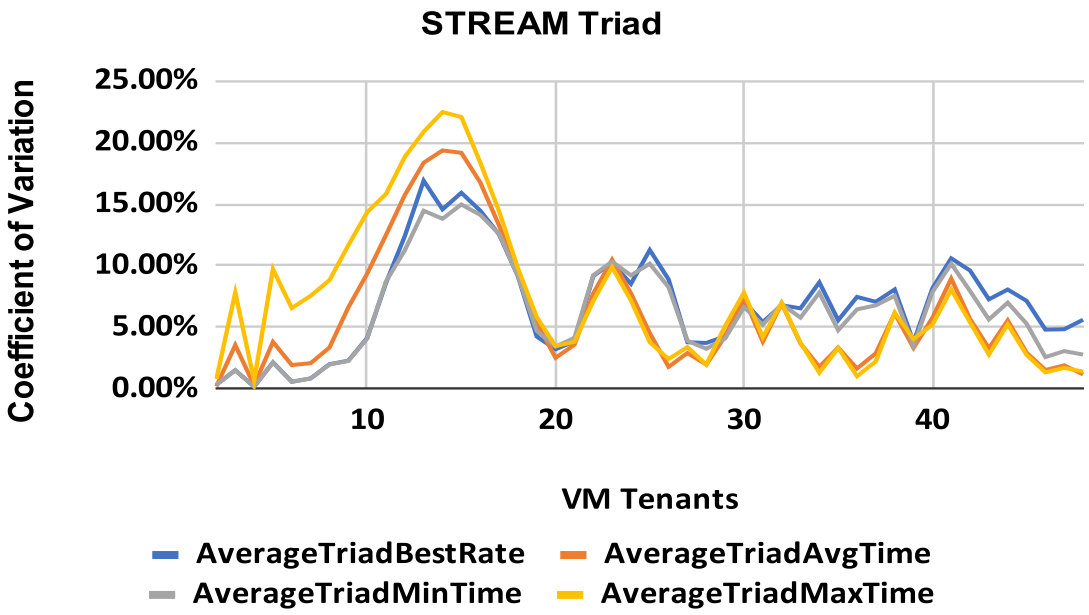

Figure 4.11 Coefficient of Variation of STREAM Triad dataset with increasing VM tenancy

4.3 MODELS -VIRTUAL MACHINE CO-RESIDENCY PREDICTION

Leveraging the metrics obtained from the benchmarking experiments, we selected performance metrics from CacheBench write, STREAM, and pmbench as features to train random forest models to predict VM co-residency. The performance metrics from the benchmarking experiments were obtained by running benchmarks on a dedicated host to characterize performance with a varying number of VMs from 1 to 48. The benchmark runs were scheduled at equal intervals of 1- 2 minutes apart according to the specific benchmark's runtime to ensure individual runs could complete before the next test cycle would begin. For each test cycle the number of VMs was scaled down from 48 to 1 in steps of 1. VMs not running a benchmark were paused to eliminate any overhead from having an idle machine share the physical host. Later all VMs were restarted to retrieve benchmarking results. The runs spanning from a single run on 1 VM to 48 runs across 48 VMs produced a dataset consisting of 1,176 rows of data in CSV format. This dataset was then used to train random forest models. The training dataset was split into 5 combinations of training and testing datasets i.e., 70-30, 72-25, 80-20, 85-15, and 90-10. Initially all available features were included as predictors in the models. An evaluation of the first five models is shown using metrics obtained and tabulated in Table 4.8 A second set of test runs was performed on the dedicated host to produce a separate CSV data file with 1,176 rows for use as a validation data set. The training and testing dataset were independent of each other and were obtained from separate benchmark runs performed on different ec2 dedicated hosts. We trained another random forest model using 100% of the training dataset and then used the separate validation dataset to evaluate this model as shown in Table 4.6 in the row, "Separate test and train csv**".**

Upon analyzing the random forest model metrics, we observed that the “Separate test and train csv” model that was trained using 100% of the test data, and evaluated using a separate set of validation data had R-Squared that was much lower than models trained and evaluated using various combinations of the training CSV (e.g., 815/15, 80/20, etc.). This is to be expected, because the validation dataset was unseen data, not used in training of the model. We then carefully analyzed independent features from the dataset, and discovered bimodal characteristics of certain features, this was explained above in section 4.2. We then removed all the features with bimodal data and high CV to produce a new random forest model with just three features. This random forest model was trained with just three features (i.e., CacheBench 1 KiB throughput, CacheBench 1 Mib throughput, and pmbench average page latency). This model was used to predict the number of co-located VMs. The three-feature model was trained using the training dataset and then evaluated using the separate validation dataset. The metrics from the evaluation of the three feature models are shown in Table 4.6. For the new model, the R-Squared value improved by .182 in comparison to the model which included all available memory benchmarks as features.

Table 4.6 Consolidated metrics from Random Forest Model trained using all the features

Evaluation metrics → / Training/Test Data Split ↓	R squared on the testing set	Root Mean Squared Error (RMSE)	Mean Absolute Error (MAE)	Min prediction	Max prediction	Average Prediction	% Variance Explained
70-30	0.934	2.884	1.957	1.055	45.662	15.354	94.68
75-25	0.948	2.562	1.796	0.986	45.160	15.492	94.27
80-20	0.956	2.475	1.721	0.795	45.968	15.415	94.38
85-15	0.951	2.455	1.740	0.996	45.729	14.914	94.7
90-10	0.943	2.704	1.849	0.815	44.125	15.893	94.74
Separate training and test data	0.591	8.530	7.490	0.090	45.301	12.445	98.89

Table 4.7 Consolidated metrics from Random Forest Model trained using three best features

Evaluation metrics → / Dataset ↓	R squared on the testing set	Root Mean Squared Error (RMSE)	Mean Absolute Error (MAE)	Min prediction	Max prediction	Average Prediction	% Var Explained
3 Feature Model	0.773	5.659	4.449	0.244	45.580	14.242	87.18

4.4 EC2 PLACEMENT GROUP VM TENANCY PREDICTIONS IN THE PUBLIC CLOUD

For RQ-2 we trained the random forest model described in Table 4.9 using three memory benchmarks with the lowest CV. This model was then used to predict the VM tenancy for the open cloud experiments in support of RQ-3. We launched pools of m5d.large spot EC2 instances using three different placement groups (i.e., spread, partition, and cluster) and also with no placement group in the US East (N. Virginia) region. VMs for partition and cluster VMs were launched in the 'd' availability zone (us-east-1d) relative to a personal AWS account. We note that availability zone designations on Amazon EC2 are floating and not fixed, so that not all users experience the same letter to zone mappings. This design approach is used to prevent the most commonly selected zone (i.e., us-east-1a) from being overused. The following launch configurations were used perform the open cloud experiment: a pool of 50 x m5d.large instances were allocated without any placement group, 50 instances were allocated using a cluster placement group, 50 instances were allocated within a single partition of a partition placement group, and 35 instances were allocated across 5 availability zones using a spread placement group. The spread placement group on Amazon EC2 ensures that VMs are created on distinctly different server racks. This not only guarantees server (node) isolation, but also rack isolation. Due to having a limited amount of hardware to meet this constraint, spread placement limits users to launching a maximum of 7 VMs per availability zone.

We then executed our memory benchmarks across these VMs in parallel to collect data for model inferencing. Each of our three benchmarks was executed 5 times for each of the placement group configurations. The values obtained were used to perform model inferencing using the three-feature random forest model to generate VM co-residency predictions.

We calculated the average VM co-residency by performing model inferencing five times for each of the benchmark runs collected on the VMs. We rounded VM co-residency predictions to the nearest whole integer. These rounded prediction values represent the predicted number of co-located VMs. For our VM pools, we then generate histograms to illustrate the distribution of VM co-residency predictions. Our histograms show the frequency/number of VMs with an identical co-residency prediction (y-axis) vs. the VM tenancy value (x-axis). Figures 4.12, 4.13, 4.14, 4.15 provide the histograms depicting the distribution of VM co-residency predictions for no placement group, the cluster placement group, the spread placement group, and the partition placement group, respectively. Table 4.8 provides the average VM co-residency predictions for each placement group.

Table 4.8 Average Predicted m5d.large VM Co-residency
by Placement Strategy on AWS EC US-EAST-1

Placement Group	**Pool Size (# of VMs)**	**Average VM Co-residency**
No Placement Group	50	25.034
Cluster	50	29.052
Spread	35	21.047
Partition	50	21.279

Figure 4.12 provides a histogram of VM co-residency predictions for no placement group. The minimum VM co-tenancy prediction was 1 and the maximum prediction was 44. The VM occurrences cluster more around 18 and 37.

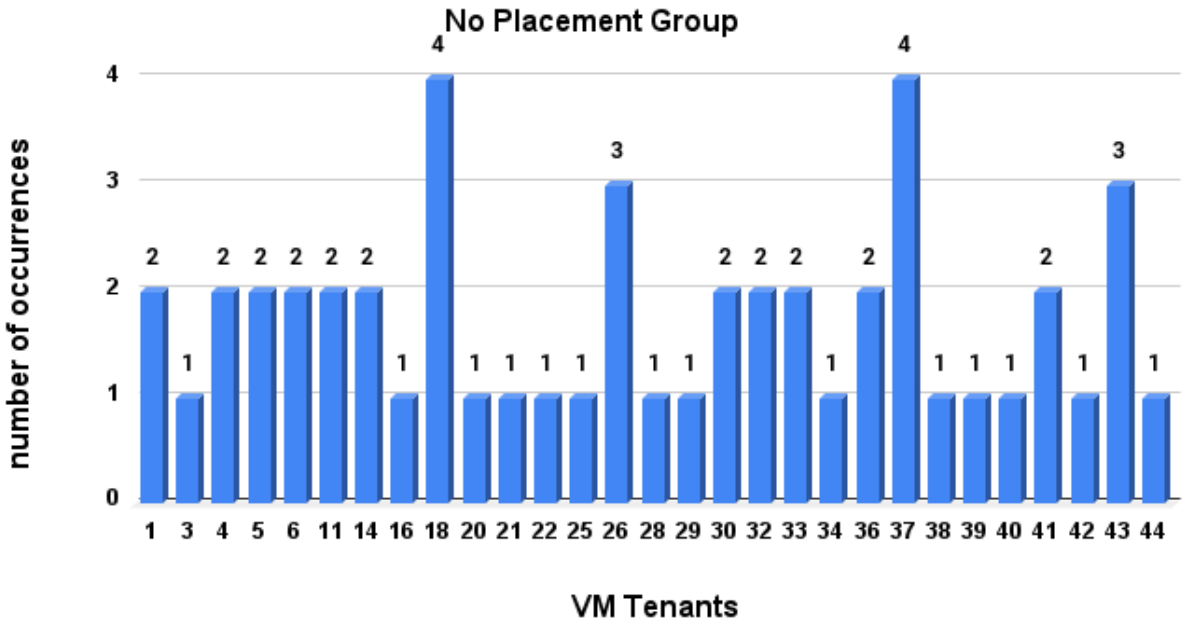

Figure 4.12 Predicted number of co-located VMs in a pool of 50 x m5d.large instances provisioned in us-east-1 with no placement group

Figure 4.13 provides a histogram of VM co-residency predictions for our cluster placement group testing. The minimum VM co-tenancy prediction was 3 and the maximum prediction was 44. The VM occurrences cluster more around 18, 35,38, 41 and 43.

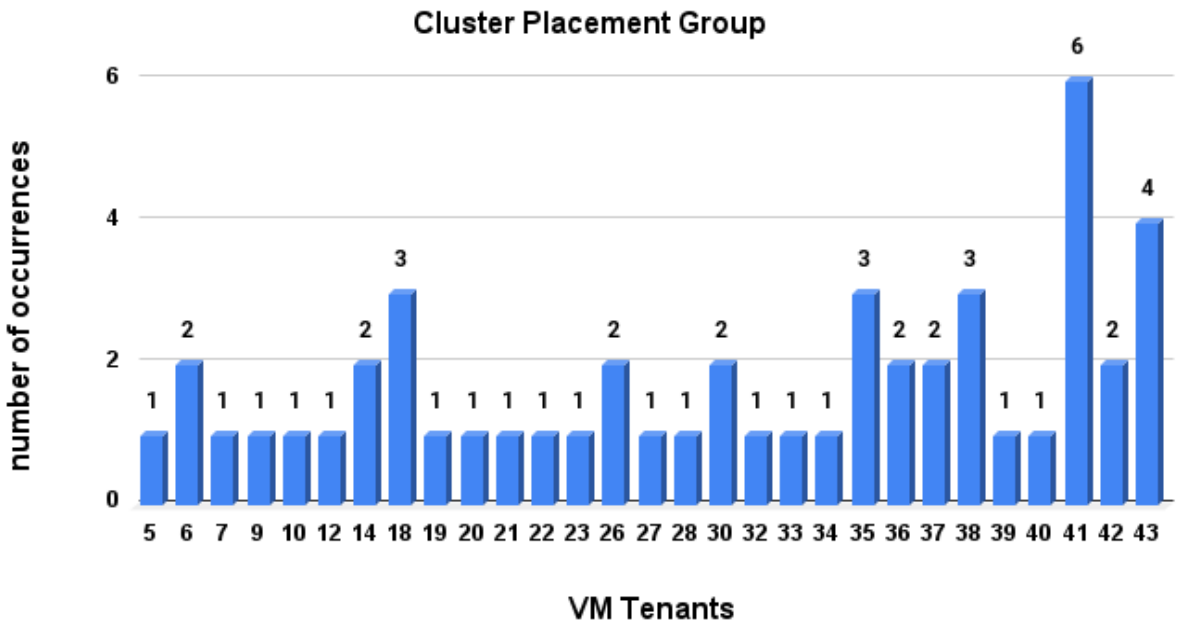

Figure 4.13 Predicted number of co-located VMs in open cloud in a pool of 50 x m5d.large instances provisioned on us-east-1 using the cluster placement group

Figure 4.14 provides a histogram of VM co-residency predictions for the spread placement group. The minimum VM co-tenancy prediction was 1 and the maximum prediction was 43. The VM occurrences cluster more around 9, 11, 20 and 21.

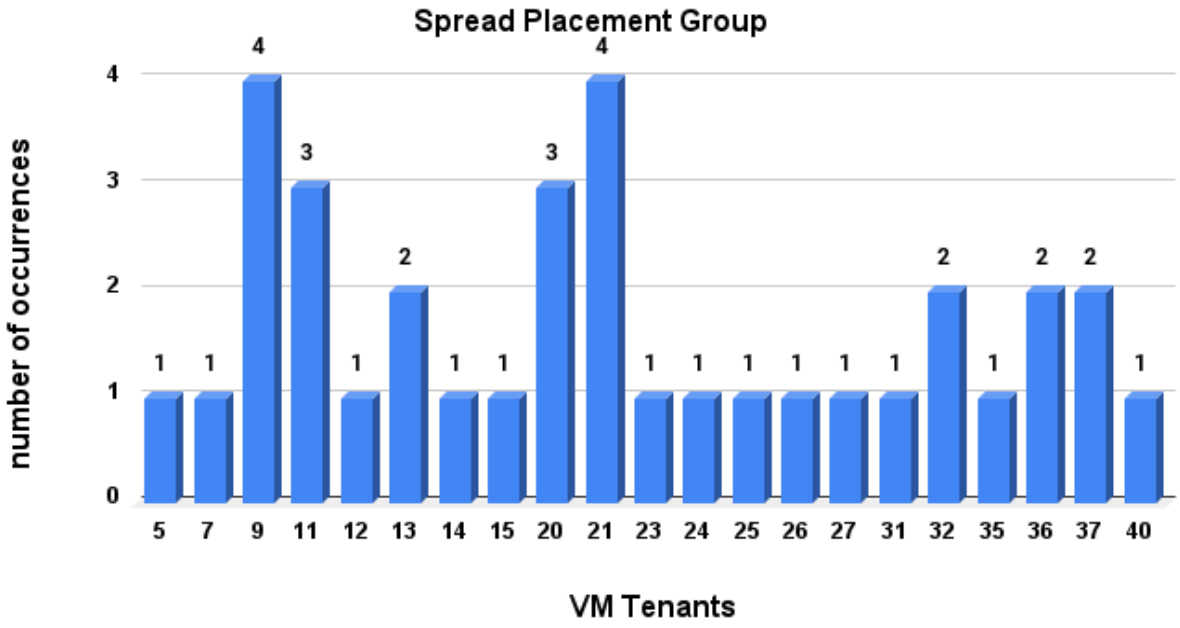

Figure 4.14 Predicted number of co-located VMs in open cloud in a pool of 35 x m5d.large instances provisioned on us-east-1 using the spread placement group

Figure 4.15 provides a histogram of VM co-residency predictions for our partition placement group testing. The minimum VM co-tenancy prediction was 1 and the maximum prediction was 41. The VM occurrences cluster more around 24 and 36.

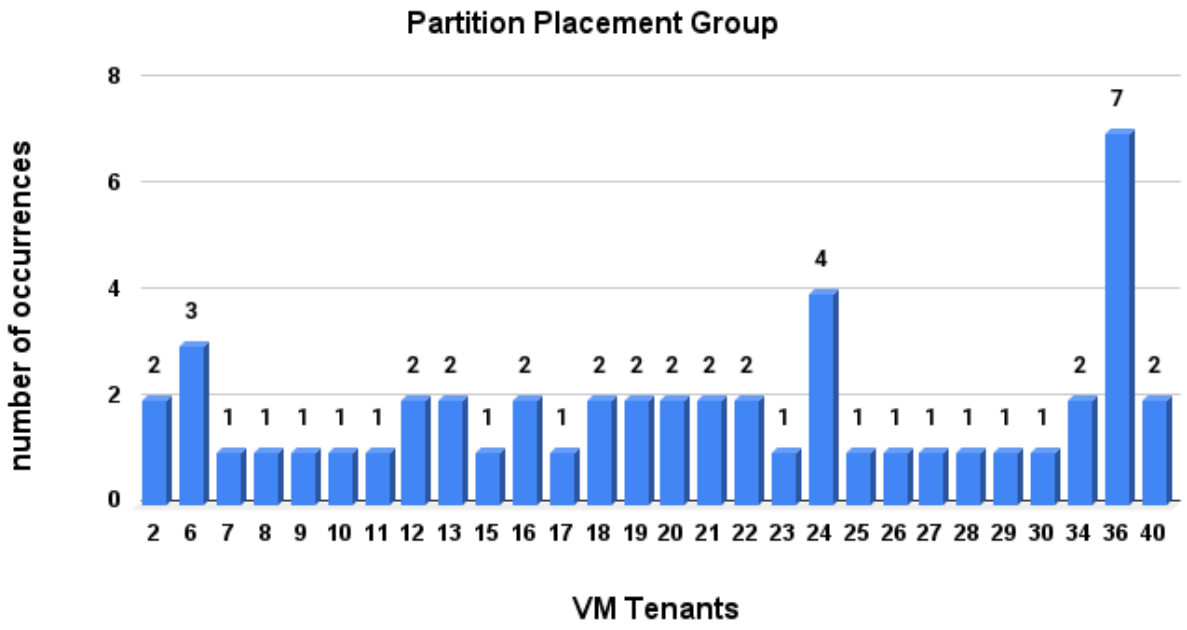

Figure 4.15 Predicted number of co-located VMs in open cloud in a pool of 50 x m5d.large instances provisioned on us-east-1 using the partition placement group

The above predictions of VM tenancy for different placement strategies on the public cloud addresses **RQ-3:** *Using the EC2 placement groups (e.g., spread, cluster, and partition) what degree of VM multi-tenancy is detectable?* **Our data, as provided in** Table 4.8 **suggests that the average VM tenancy for instances in the cluster placement and no placement groups is higher compared to instances in the spread and partition placement groups.** For pools of 50 VMs launched using no placement group and the cluster placement group, our model predicted an average of 25 and 29 co-resident VMs. For the spread placement group, 35 VMs were provisioned across 5 availability zones with 7 VMs in each zone. For the Spread placement group, an average of 21 co-resident VMs were predicted by our model. For 50 VMs launched using the partition placement group, an average of 21 VMs were predicted.

Chapter 5. DISCUSSION

The three feature model metrics are better when compared to the metrics obtained from models that includes all available features for our memory benchmarking dataset. The three-feature model had higher R-squared and the root mean squared error and mean absolute error are lower.

The use of EC2 placement groups helped us to identify the problem in the original random forest models. This was observed when using these models to make VM co-residency predictions for the open cloud experiments. The predictions were simply not believable or close to the expected values for the spread, cluster, and partition placement groups. Upon training a new random forest model with just 3 features, VM co-residency predictions were closer to the expected values and more in line with those from [6]

The average prediction of VM co-residency for the spread placement group is 21, which is higher than expected. This may be due to the presence of background noise caused due to memory resource contention.

From the above set of experiments, we noticed the bimodal nature of benchmark results, the dataset was examined for outliers and cleaned by removing the outliers, but still the model performance did not improve. It was then that the CV of various features was examined to eliminate features with bimodal behavior to fine tune the set of features. This resulted in our three-feature model that generated more reasonable predictions.

Chapter 6. FUTURE WORK

This thesis investigation along with the research conducted by [6] provides insights pertaining to resource contention from co-resident VMs in the public cloud. The earlier effort focused on CPU, disk, and network resource contention. This thesis has focused on investigation of memory resource contention. One of the major findings of this research was bimodality of the memory benchmark results observed in the throughput values of the STREAM and CacheBench benchmarks. The random forest models were trained and tuned with these datasets to predict VM co-residency. As part of future research, mixture models such as Gaussian mixture models, and weighted quantile regression forests can be explored to handle bimodal data. Various methods to transform the bimodal data to normal data warrant exploration. An open question requiring further investigation is how to remove noise in the benchmark results from other co-located VMs in the public cloud that do not run benchmarks.

Chapter 7. CONCLUSION

In this thesis, we observed considerable memory performance degradation measured by running memory benchmarks in parallel. **(RQ-1)** is addressed by the memory stress benchmarks. Performance degradation was observed using three different memory stress benchmarks. STREAM Copy, Scale, Add, and Triad produced a best-to-worst case throughput difference of 64%, 63%, 66%, and 67% respectively, with idle VMs running and the same set of metrics produced a best-to-worst case throughput difference of 66%, 67%, 73%, and 71% respectively. The performance delta for these metrics increased by 2%, 4%, 7%, and 4% respectively when idle VMs were shut down for the experiment. Our results demonstrate how performance degrades with increasing VM tenancy and how performance improved when idle VMs were shutdown. CacheBench write for 1KiB, 1MiB, 1GiB, 4GiB throughput suffered a performance degradation of 103%, 102%, 19%, and 21% respectively with the idle VMs running and 92%, 91%, 19%, and 24% with idle VMs stopped respectively. This demonstrates the implications of memory resource contention and its improvement by stopping the VMs. Pmbench is another benchmark that demonstrated greater memory access latencies as the VM tenancy increased from 1 to 48.

(RQ-2) The metrics obtained from memory benchmarks were used to train random forest models. Upon close examination of our data and model results bimodality of memory benchmark behavior was discovered. We were able to identify features with strong bimodality and thereby remove them to generate a fine-tuned model that provided better predictions. 1 KiB and 1 MiB throughput values from CacheBench and Average Page Latency from pmbench were chosen as the best features to

train the fine-tuned model, which was used to make predictions of VM co-tenancy on the public cloud.

(RQ-3) The three-feature model trained in the above experiment was used to predict co-resident VMs launched using different placement strategies on the public cloud. Our model predicted average VM tenancy of 25, 29, 21, 21 for no placement group, cluster placement group, spread placement group, and partition placement group.

BIBLIOGRAPHY

[1] B. Gregg, “No Title.” https://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html

[2] AWS Dedicated hosts, “No Title.” https://aws.amazon.com/ec2/dedicated-hosts/

[3] EC2 instances info, “No Title.” https://instances.vantage.sh/

[4] AWS EC2, “No Title”, [Online]. Available: https://aws.amazon.com/ec2/

[5] W. Lloyd, S. Pallickara, O. David, M. Arabi, and K. Rojas, “Mitigating resource contention and heterogeneity in public clouds for scientific modeling services,” *Proceedings - 2017 IEEE International Conference on Cloud Engineering, IC2E 2017*, pp. 159–166, 2017, doi: 10.1109/IC2E.2017.29.

[6] X. Han, R. Schooley, D. MacKenzie, O. David, and W. J. Lloyd, “Characterizing Public Cloud Resource Contention to Support Virtual Machine Co-residency Prediction,” *Proceedings - 2020 IEEE International Conference on Cloud Engineering, IC2E 2020*, pp. 162–172, 2020, doi: 10.1109/IC2E48712.2020.00024.

[7] A. O. Ayodele, J. Rao, and T. E. Boult, “Performance Measurement and Interference Profiling in Multi-tenant Clouds,” *Proceedings - 2015 IEEE 8th International Conference on Cloud Computing, CLOUD 2015*, pp. 941–949, 2015, doi: 10.1109/CLOUD.2015.128.

[8] Y. Amannejad, D. Krishnamurthy, and B. Far, “Managing Performance Interference in Cloud-Based Web Services,” *IEEE Transactions on Network and Service Management*, vol. 12, no. 3, pp. 320–333, 2015, doi: 10.1109/TNSM.2015.2456172.

[9] Phoronix, “No Title”, [Online]. Available: https://www.phoronix-test-suite.com/

[10] Pmbench, “No Title”, [Online]. Available: https://openbenchmarking.org/test/pts/pmbench

[11] STREAM, “No Title.” https://www.cs.virginia.edu/STREAM/

[12] P. J. Mucci, “The CacheBench Report,” pp. 1–18, 1998.

[13] “AWS EC2 Placement groups”, [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/placement-groups.html#concepts-placement-groups

[14] V. Varadarajan, T. Kooburat, B. Farley, T. Ristenpart, and M. M. Swift, “Resource-freeing attacks: Improve your cloud performance (at your neighbor’s expense),” *Proceedings of the ACM Conference on Computer and Communications Security*, pp. 281–292, 2012, doi: 10.1145/2382196.2382228.

[15] M. S. Rehman and M. F. Sakr, “Initial findings for provisioning variation in cloud computing,” *Proceedings - 2nd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2010*, pp. 473–479, 2010, doi: 10.1109/CloudCom.2010.47.

[16] J. Schad, J. Dittrich, and J. A. Quiané-Ruiz, “Runtime measurements in the cloud: Observing, analyzing, and reducing variance,” *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 460–471, 2010, doi: 10.14778/1920841.1920902.

[17] Z. Ou *et al.*, “Is the Same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds,” *IEEE Transactions on Cloud Computing*, vol. 1, no. 2, pp. 201–214, 2013, doi: 10.1109/TCC.2013.12.

[18] B. Farley, V. Varadarajan, K. D. Bowers, A. Juels, T. Ristenpart, and M. M. Swift, “More for your money: Exploiting performance heterogeneity in public clouds,” *Proceedings of the 3rd ACM Symposium on Cloud Computing, SoCC 2012*, 2012, doi: 10.1145/2391229.2391249.

[19] Y. Qiu, Q. Shen, Y. Luo, C. Li, and Z. Wu, “A secure virtual machine deployment strategy to reduce co-residency in cloud,” *Proceedings - 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, 11th IEEE International Conference on Big Data Science and Engineering and 14th IEEE International Conference on Embedded Software and Systems*, pp. 347–354, 2017, doi: 10.1109/Trustcom/BigDataSE/ICESS.2017.257.

[20] S. Yu, X. Gui, J. Lin, X. Zhang, and J. Wang, “Detecting VMs co-residency in the cloud: Using cache-based side channel attacks,” *Elektronika ir Elektrotechnika*, vol. 19, no. 5, pp. 73–78, 2013, doi: 10.5755/j01.eee.19.5.2422.

[21] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter, “Homealone: Co-residency detection in the cloud via side-channel analysis,” in *2011 IEEE symposium on security and privacy*, 2011, pp. 313–328.

[22] C. Phoronix, “No Title.” https://openbenchmarking.org/test/ pts/CacheBench

[23] M. Instances, “No Title.” https://aws.amazon.com/ec2/instance-types/m5/