vmBBThrPred: A Black-Box Throughput Predictor for Virtual Machines in Cloud Environments

Javid Taheri[1(✉)], Albert Y. Zomaya[2], and Andreas Kassler[1]

[1] Department of Computer Science, Karlstad University, Karlstad, Sweden
{javid.taheri,andreas.kassler}@kau.se
[2] School of Information Technologies, University of Sydney, Sydney, Australia
albert.zomaya@sydney.edu.au

Abstract. In today's ever computerized society, Cloud Data Centers are packed with numerous online services to promptly respond to users and provide services on demand. In such complex environments, guaranteeing throughput of Virtual Machines (VMs) is crucial to minimize performance degradation for all applications. vmBBThrPred, our novel approach in this work, is an application-oblivious approach to predict performance of virtualized applications based on only basic Hypervisor level metrics. vmBBThrPred is different from other approaches in the literature that usually either inject monitoring codes to VMs or use peripheral devices to directly report their actual throughput. vmBBThrPred, instead, uses sensitivity values of VMs to cloud resources (CPU, Mem, and Disk) to predict their throughput under various working scenarios (free or under contention); sensitivity values are calculated by vmBBProfiler that also uses only Hypervisor level metrics. We used a variety of resource intensive benchmarks to gauge efficiency of our approach in our VMware-vSphere based private cloud. Results proved accuracy of 95 % (on average) for predicting throughput of 12 benchmarks over 1200 h of operation.

Keywords: Performance prediction and modeling · Throughput degradation · Cloud infrastructure

1 Introduction

The demand for cloud computing has been constantly increasing during recent years. Nowadays, Virtualized Data Centers (vDCs) accommodate thousands of Physical Machines (PMs) to host millions of Virtual Machines (VMs) and fulfill today's large-scale web applications and cloud services. Many organizations even deploy their own private clouds to better manage their computing infrastructure [7]. In fact, it is shown that more than 75 % of current enterprise workloads

M. Aiello et al. (Eds.): ESOCC 2016, LNCS 9846, pp. 18–33, 2016.
DOI: 10.1007/978-3-319-44482-6_2

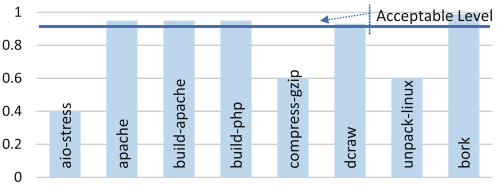

Fig. 1. Relative performance of eight applications when co-located with a Mem+Disk (unzipping large files) intensive application

are currently running on virtualized environments [11]. Despite massive capital investments (tens to hundreds of millions of dollars) however, their resource utilization rarely exceeds 20 % of their full capacity [11,14]. This is because, alongside its many benefits, sharing PMs also leads to performance degradation of sensitive co-located VMs and could undesirably reduce their quality of service (QoS) [13]. Figure 1 shows relative throughput (with regard to their isolated run) of eight high resource demanding VMs when co-located with a background VM running a high Memory+Disk intensive application (unzipping large files). All VMs had 2vCPU, 2 GB of RAM, and 20 GB of Disk. For each test, VMs were pinned on the same set of CPUs/Cores and placed on the same disk to compete for CPU cycles, conflict on L1/L2/L3 memory caches, and interfere with each others' disk access. As it can be inferred from Fig. 1, despite being classified as resource demanding, five of these applications (e.g., apache) could be safely co-located with the background resource intensive application (Mem+Disk) – assuming that performance degradation of up to 10 % is allowed. Nevertheless, a conservative view would separate/isolate all VMs to allocate them on separate PMs. This simple example shows/motivates that understanding, measuring, and predicting performance degradation is essential to identify VMs that can be safely co-located with minimum interference to each other. It also motivates the importance of designing effective affinity rules to guarantee optimal placement of VMs, and consequently maximize the overall performance of vDCs.

This work is a major step to predict throughput, and consequently performance degradation of general purpose applications/VMs through profiling a variety of benchmarks under different working scenarios and resource limitations. Such profiles are then used to predict throughput of a VM only based on the amount of resources (CPU, Mem, and Disk) it is consuming as seen by the Hypervisor. We used 12 well-known benchmarks with different resource usage signatures (CPU/Mem/Disk intensive and various combinations of them) to run on three different PMs. Results were collected and used to model throughput, and consequently performance degradation. We finally aligned our results with actual throughput of these benchmarks to show the accuracy of our approach: VM Black-Box Throughput Predictor (vmBBThrPred).

Our contribution in this work can be highlighted as: unlike all available similar approaches, (1) vmBBThrPred uses only Hypervisor level metrics to predict throughput and performance degradation of a VM. No code/agent is required

to be developed, installed, and/or executed inside VMs; (2) vmBBThrPred provides a systematic approach to formulate throughput of VMs; (3) vmBBThrPred uses a wider range of benchmarks (from pure CPU/Mem/Disk intensive benchmarks to various combination of CPU+Mem+Disk intensive ones) to produce such formulas; and (4) vmBBThrPred produces a polynomial formula for each application/VM so that its throughput can be directly and dynamically (online) calculated according to its current CPU, Mem, and Disk utilization.

The remainder of this paper is structured as follows. Section 2 reviews the related work. Section 3 explains the architecture of vmBBThrPred and elaborates on its components. Section 4 demonstrates vmBBThrPred's step-by-step procedures. Section 5 lays out our experimental setup. Results are discussed and analyzed in Sect. 6, followed by Conclusion in Sect. 7.

2 Related Work

The ever increasing popularity of virtualization [12] in vDCs is probably one of the most significant paradigm shifts in the IT industry. Through virtualization, PM resources are partitioned for VMs to run services. Running a highly efficient vDC is however not a trivial task. Firstly, vDCs are envisaged to run several VMs on each PM assuming proper partitioning of its resources. Although resources like CPU and Network seem to be fairly partition-able, Mem and Disk are proven to be much more cumbersome. Secondly, vDCs need to provide accurate/online operational information to both administrators and users so that functionality of deployed services can be monitored, controlled, and ensured at all times. This requires identifying under-performing VMs – those who suffer the most because of co-location – immediately, effectively, and dynamically. This also demands the ability of vDC management systems to accurately – at least within acceptable margins – predict the performance of different VMs in various working scenarios; i.e., isolated or co-located as well as under or free of resource contentions. This concern, in particular, seems to be more important than the other two because it could directly lead to further optimizations of the whole system as well as significant increase of the productivity of vDCs.

To date, many approaches are proposed to measure throughput, and consequently performance degradation of VMs in vDCs; they can be categorized into the following two main themes.

High-Level (SLA) Based Measurements: Approaches in this group use high-level metrics to measure actual throughout of an application/VM (e.g., the number of transactions a SQL server responds to per second) in its current situation. They all rely on developing tailor-made foreign agents/codes for each application, installing them in VMs, and giving them enough system privileges to collect and send out performance data.

Xu et al. [19] proposed two Fuzzy based systems (global and local) to monitor resource utilization of workloads in VMs. The local system is an SLA sensor that is injected into a VM to directly compare its performance with the desired SLA, and request or relinquish resources (e.g., CPU share) if required.

The global controller receives all local requests and decides what VM should get more resources in cases of contention. Tested for CPU-intensive workloads, their self-learning fuzzy systems could efficiently tune itself to demand for "just right" amount of resources. Their approach however assumed that high-level SLAs (e.g., http requests per second) can be accurately defined and measured per application/VM. Rao et al. [16] proposed VCONF, an auto-configuration RL-based agent, to automatically adjust CPU and Memory shares of VMs to avoid performance degradation. They, too, used direct application measurements to generate efficient polices for their Xen based environment. Watson et al. [18] used probabilistic performance modeling to control system utilization and response time of 3-tier applications such as RUBiS. They showed that CPU allocation of VMs are enough to control high level SLAs such as response time of applications. Caglar et al. [9] proposed hALT, an online algorithm that uses Artificial Neural Networks (ANNs) to link CPU and Memory utilization of CPU intensive applications/tasks in Google trace data to performance degradation of VMs. They used another ANN to recommend migration of VMs to assure QoS for Google services. For real deployments, they still need an agent to report "performance" of an application/VM to feed and train their ANNs. Bartolini et al. [8] proposed AutoPro to take a user-defined metric and adjust VMs' resources to close the gap between their desired performances and their current ones. AutoPro uses a PI controller to asymptotically close this gap and can work with any metric – such as frame/s, MB/s, etc. – as long as developers can provide it.

Approaches in this group are generally more accurate than others because they use direct measurements/feedback from applications inside VMs. Their usage however could be very limited, because (1) they all rely on an inside tailor-made agent to report the exact throughput of an application/VM, and (2) their focus is mostly to improve performance of VMs rather than modeling throughput of applications/VMs according to their resource utilization.

Low-Level (Resource) Measurements: Approaches in this group use low-level metrics (e.g., CPU utilization) to predict throughput (e.g., the response time of a web-server) of an application/VM in its current situation. They too rely on developing tailor-made foreign agents/codes for each application/VM, installing them in the VM, and giving them enough system privileges to collect and send out performance data.

Q-cloud [15] uses a feedback-agent inside each VM to report its CPU utilization. They used five CPU intensive application from SPECjbb [1] and showed that there are direct relations between the amount of CPU a VM uses with its actual throughput. Using a MIMO linear model, authors then model interference of CPU intensive applications and feedback "root" in Hyper-V to adjust CPU allocations of VMs to improve their performances. Du et al. [10] proposed two profiling agents to collect guest-wide and system-wide performance metrics for developers so that they can accurately collect information about their products in KVM-based virtualized environments. They did not use any specific benchmark, but simple programs to engage different parts of a system.

Approaches in this group generally predict throughput of an application/VM in relation to its resource utilization, although mostly to avoid performance degradations rather than modeling and predicting throughput. Also, although these approaches can be easily modified to use Hypervisor level metrics – instead of reports from their inside agents – to predict applications' throughout, their focus on only CPU or Disk intensive applications makes them non-generalizable.

After close examination of many techniques presented to date, we have noticed the following shortcomings. Firstly, many techniques require an agent/code to be injected to a VM to report either its throughput or its performance data. The need to have access to VMs and permission to run tailor-made foreign codes is neither acceptable not practical in most general cases. Secondly, many techniques aim to predict throughput of an application only to avoid contention by using/controlling one resource type (CPU, Mem, or Disk). Finally, most approaches target known applications that do not have multidimensional resource demands: they are pure CPU, Mem, or Disk intensive.

To address these shortcomings, we designed vmBBThrPred to directly model and formulate throughput of an unknown application/VM according to its resource usages. vmBBThrPred is an application-agnostic non-intrusive approach that does not require access to the VM to run foreign agents/codes: it only uses Hypervisor level metrics to systematically relate multidimensional resource usage of a VM to its actual throughput, and consequently performance degradation for various working scenarios (free or under resource contention).

3 Architecture of vmBBThrPred

The key idea of vmBBThrPred is to use the sensitivity values of an application to model/formulate its throughout. vmBBProfiler, our systematic sensitivity analysis approach in [17], was designed to pressure an unknown VM to work under different working scenarios and reveal its sensitivity to each resource type. vmBBProfiler calculates three sensitivity values ($\in [0, 1]$) upon profiling a VM: Sen^c, Sen^m, and Sen^d to respectively reflect sensitivity of a VM to its CPU, Mem, and Disk. For example, $\text{Sen}^c = 1$ implies that the profiled VM significantly changes its behavior, and consequently its throughput when it suffers to access CPU. $\text{Sen}^c = 0$ implies that throughput of the profiled VM is insensitive to its CPU share; e.g., when the VM is running a Disk intensive application. Other values of $\text{Sec}^{c/m/d}$ reflect other levels of sensitivity: the larger the $\text{Sec}^{c/m/d}$ the more sensitivity to a resource type. vmBBProfiler is also application-oblivious and uses no internal information about the nature of the applications running inside the VM when profiling it; Fig. 2 shows the architecture of both vmBBProfiler and vmBBThrPred and how they are related to each other. All components of vmBBProfiler and vmBBThrPred are totally separate and performing non-redundant procedures; both are run outside the VM and are currently implemented using PowerShell [2] and PowerCLI [4] scripts for Windows-7 and above.

vmBBProfiler: The key idea in vmBBProfiler is to identify how a VM behaves under resource contention. Its architecture relies on two components (Fig. 2):

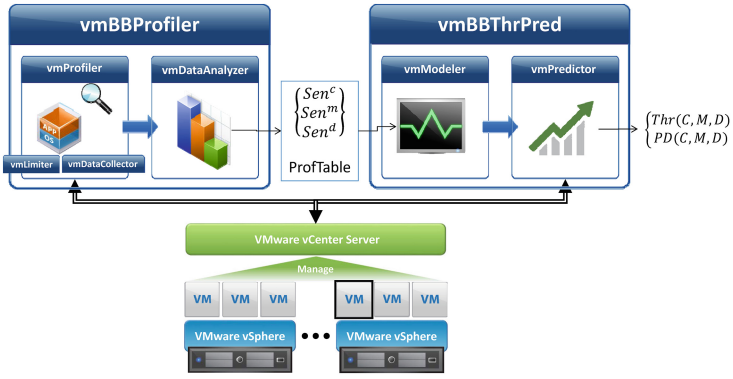

Fig. 2. Architecture of vmBBProfiler and vmBBThrPred

vmProfiler and vmDataAnalyser. The vmProfiler, in turn, consists of two parts: vmLimiter and vmDataCollector to respectively command a Hypervisor, through VMware-vCenter [5] in our case, to impose resource limits to a VM, and collect/record its behavior under the imposed limits.

vmProfiler aims to emulate contention through limitation. That is, instead of challenging a VM to compete with other co-located VMs to access/use resources (CPU, Mem, and/or Disk), the vmLimiter limits resource usage of the VM so that it reveals its behavior under hypothetical contentions. We showed in [17] that although resource starvation under "contention" and "limitation" are different, they always lead to very similar performance degradation (less than 5 % different on average). `cpu/mem/diskLimit` $\in [0, 1]$ sets the percentage of CPU/Mem/Disk that the VM can use. For example, if a VM has two 2.4 GHz vCPUs, `cpuLimit = 0.25` would limit CPU usage of this VM to $0.25 \times 2 \times 2.4 = 1.2$ GHz. After imposing a set of limits to resources, vmDataCollector is then launched to collect/record performance of the VM through polling several Hypervisor level metrics; it only polls VM metrics (e.g., CPU utilization) that are already collected by the Hypervisor: it neither demands nor needs any specific metric from the VM itself.

Table 1 shows a sample profiling table upon completion of vmBBProfiler; this table will be refereed to as "ProfTable" for the rest of this article. In this table, `cpuLimit` $\in \{c_1, c_2, \ldots, c_{nc}\}$, `memLimit` $\in \{m_1, m_2, \ldots, m_{nm}\}$, and `diskLimit` $\in \{d_1, d_2, \ldots, d_{nd}\}$ produce a total number of $nc \times nm \times nd$ profiling scenarios. `metricX` is the average of the X-th Hypervisor metrics (e.g., disk.read.average (KBps)) during the imposed limitation scenario. It is worth noting that each metric is a series of values during the profiling phase (e.g., 15 values for 5 min of profiling in [17]), however because they showed to have negligible standard deviation, their average values proved to be accurate enough to be used in vmBBThrPred.

Upon profiling behavior of a VM under several limitation profiles, vmDataAnalyser is invoked to analyze the profiled data and calculate sensitivity of the VM to its CPU, Memory, and Disk allowances; they are respectively named Sen^c,

Table 1. vmBBProfiler output table (ProfTable) after profiling a VM

Scenario #	`cpuLimit`	`memLimit`	`diskLimit`	(`metric1`, `metric2`, ..., `metricK`)
1	c_1	m_1	d_1	$\cdots$
2	c_1	m_1	d_2	$\cdots$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$nc \times nm \times nd$	c_{nc}	m_{nm}	d_{nd}	$\cdots$

Sen^m, and Sen^d. For example, it would suggest that the application in the VM, which is assumed responsible for its resource demands, would always stay more sensitive to its CPU allocation than to its Memory bandwidth. As a result, it speculates that performance of this VM, for example, would be degraded more if its CPU-share – as opposed to its Memory share – is halved.

vmBBThrPred: After profiling a VM using vmBBProfiler, vmBBThrPred is launched to use its sensitivity values and predict its throughput under any working scenario, even those that have not been observed in Table 1. vmBBThrPred consists of two parts: vmModeler and vmPredictor. vmModeler uses $\text{Sen}^{c/m/d}$ values and the ProfTable (both calculated and provided by vmBBProfiler) to produce a polynomial model to relate resource utilization of a VM to its throughput; vmPredictor connects directly to VMware-vCenter [5], dynamically (online) polls CPU, Mem, and Disk utilization of a VM, and uses the produced formula to predict throughout (Thr) and performance degradation (PD = 1 − Thr) of the VM at its current working condition.

4 Procedures of vmBBThrPred

The first step before delving into the procedures of vmBBThrPred is to select several Hypervisor metrics that can directly or indirectly relate to the actual throughout of an application/VM. Here, because vmBBThrPred is designed to be application-oblivious, we define the term "throughput" as a normalized value ($\in [0, 1]$) where Thr = 1 always reflect the maximum performance of a VM. Similarly, "performance degradation" (PD) is defined as (1−Thr) to reflect the amount of degradation a VM encounters in its current working situation. For the apache server (2vCPUs, 2 GB of RAM, and 20 GB) in our experimental setup (Sect. 5) for example, we observed the maximum response rate of 10900 'requests per second', when the VM hosting the apache server was run in an isolated environment. After migrating the VM to a contention environment, its respond rate was reduced to 4045. In this case, the respond rate of 10900 and 4045 would map to Thr = 1.00 (PD = 0.00) and Thr = 4045/10900 = 0.37 (PD = 0.63), respectively.

4.1 Identify Relevant Hypervisor Metrics

We performed a series of engineered experiments to find Hypervisor metrics that have significant correlations with the actual throughput of different applications. Note that the actual throughput of applications/VMs is not accessible/measurable for general purpose VMs – because of the need to install/inject monitoring codes. However, we could have access to these values because the Phoronix Test Suits [3] that we used in this article actually provides such detailed values at the end of its runs. It is worth noting that we used such detailed values only to identify (reverse-engineer) relative Hypervisor metrics; general use cases of vmBBThrPred does not require actual throughput measurements.

To this end, we used four benchmarks (out of the total 12 for this article) with different resource utilization behavior from the Phoronix Test Suite [3] to identify correlated metrics. They were 'apache' to represent CPU intensive (H/–/–), 'blogbench' to represent Memory intensive (–/H/L), 'aio-stress' to represent Disk intensive (–/–/H), and 'unpack-linux' to represent CPU+Mem+Disk intensive (L/L/L) applications/VMs. We tested each benchmark on three different PMs (Table 2) for 64 different contention scenarios (Table 1). Actual throughput values of these runs (provided by the Phoronix at the end of each run) are statistically correlated with 134 metrics provided by our VMware-vSphere [6] private cloud to identify the most significant/influential ones. Table 3 lists five metrics with the highest correlation to the actual throughput for each benchmark.

Table 2. Characteristics of used physical machines

PM name	CPU family	# Cores (speed)	Memory	Cache (L1/L2/L3)
AMD	AMD Opteron 6282 SE	64 (2.599 GHz)	256 GB	(768 KB/16 MB/16 MB)
DELL	Intel i7-3770	8 (3.40 GHz)	16 GB	(256 KB/1 MB/8 MB)
SGI	Intel Xeon(R) E5420	8 (2.493 GHz)	32 GB	(256 KB/12 MB/–)

As it can be seen, for one-resource-intensive benchmarks (Table 3a–c), throughput of apache, blogbench, and aio-stress is highly correlated with CPU, Mem, and Disk, respectively. For the unpack-linux with multi-resource-intensive nature however, metrics for all three resource types are listed. To compile a list of metrics to cover all cases, we averaged correlation values for all four benchmarks and build Table 4. Based on this table, we chose the cpu.usage.average (%), mem.usage.average (%), and disk.usage.average (KBps) as the three most correlated metrics to actual throughput of general purpose/unknown applications/VMs. In Sect. 5, we will show that throughout, and consequently performance degradation of all sorts of applications with various utilization patterns can be accurately (≈90–95 %) predicted using these selected metrics.

4.2 Blind Prediction

After selecting three of the most correlated Hypervisor metrics to actual throughput of applications/VMs, we performed another set of statistical analysis to dis-

Table 3. Five most correlated Hypervisor metrics for the selected benchmarks

(a) apache

Metric Name	Correlation
cpu.run.summation(millisecond)	0.99
cpu.usage.average(%)	0.99
cpu.ready.summation(millisecond)	0.99
cpu.demand.average(MHz)	0.99
cpu.overlap.summation(millisecond)	0.98

(b) blogbench

Metric Name	Correlation
mem.active.average(KB)	0.69
mem.usage.average(%)	0.69
mem.granted.average(KB)	0.68
mem.activewrite.average(KB)	0.67
mem.entitlement.average(KB)	0.65

(c) aio-stress

Metric Name	Correlation
virtualdisk.write.average(KBps)	0.99
datastore.write.average(KBps)	0.98
disk.usage.average(KBps)	0.98
virtualdisk.mediumseeks.latest(number)	0.98
disk.numberwrite.summation(number)	0.97

(d) unpack-linux

Metric Name	Correlation
disk.usage.average(KBps)	0.94
virtualdisk.mediumseeks.latest(number)	0.90
cpu.used.summation(millisecond)	0.89
cpu.usage.average(%)	0.88
mem.usage.average(%)	0.79

Table 4. Six most correlated Hypervisor metrics for all benchmarks

Metric name	Correlation
disk.numberwrite.summation (number)	0.87
disk.usage.average (KBps)	0.85
cpu.usage.average (%)	0.77
cpu.used.summation (millisecond)	0.77
mem.usage.average (%)	0.61
mem.latency.average (%)	0.55

cover the actual relation (formula) between the selected metrics and throughput values. To this end, we observed that there is a significant alignment between sensitivity values computed by vmBBProfiler and calculated correlation values. Figure 3 aligns "Correlation to Throughput" with "Sensitivity" values calculated by vmBBProfiler for all benchmarks in Table 5 on all PMs in Table 2. Comparing such alignments with the "ideal" line, which represent a perfect alignment, in these sub-figures motivates us to believe/hypothesize that the actual throughput of applications/VMs can be accurately predicted using their sensitivity values instead of their correlation values. To mathematically formulate this, we designed the following formula to predict "throughput" of a VM using only its current normalized CPU, Mem, and Disk utilization values.

$$\mathrm{Thr(C,M,D)} = \frac{C \times \mathrm{Sen}^c + M \times \mathrm{Sen}^m + D \times \mathrm{Sen}^d}{\mathrm{Sen}^c + \mathrm{Sen}^m + \mathrm{Sen}^d} \quad (1)$$

In this formula, C, M, and D are respectively the proportional of CPU, Mem, and Disk utilization of a VM with respect to their counterpart values in an isolated run. For example, assume a VM with sensitivity values of $\mathrm{Sen}^c = 1.00$, $\mathrm{Sen}^m = 0.05$, and $\mathrm{Sen}^d = 0.03$ uses 80 % of its CPU, occupies 22 % of its Mem, performs 25 KBps of Disk activity, and responds to 200 requests per second when it is run in a contention free environment (isolated run). Also assume its hosting

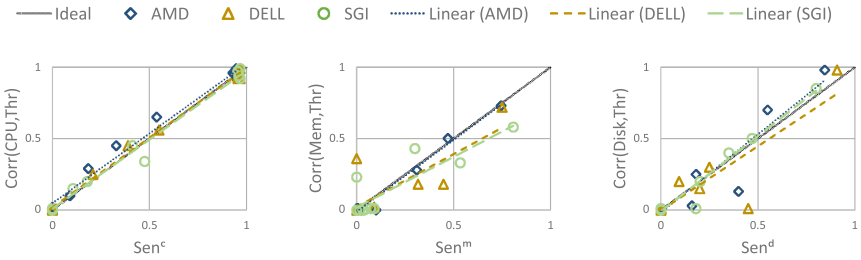

Fig. 3. Point-by-point alignment of "Correlation to Throughout" with "Sensitivity" values

VM is migrated to a PM where utilization of its resources are reduced to 45 % of CPU, 10 % of Mem, and 8 KBps of Disk because of contention. According to Eq. 1, its throughout, in this case, is predicted to be 55 % of its maximum throughout (200) in the isolated run; i.e.:

$$\text{Thr}=\left(\frac{\frac{45\,\%}{80\,\%}\times 1.00+\frac{10\,\%}{22\,\%}\times 0.05+\frac{8\text{KBps}}{25\text{KBps}}\times 0.03}{1.00+0.05+0.03}\right)\times 200 = 0.55\times 200$$

The rationale behind this formula is based on our direct observations across months of profiling. To explain it, assume CPU-usage of a CPU intensive application (such as an apache server) is 90 % and it responds to 10000 requests per seconds. Now assume that its CPU-usage is reduce to 30 % because of contention. The general sense, also confirmed by direct measurements, dictates that the VM should respond to one-third of 10000; i.e. $30/90\times 10000 = 3333$. For more complicated cases where a VM is sensitive to more than one resource, assume a Mem+Disk application (such as blogbench) is using 10 % of CPU, 70 % of Mem, and perform 17,000 KBps on Disk to conduct 100,000 blog activities in a contention free environment. Now assume this VM is migrated to another PM and its resource usages are reduced to 9 % of CPU, 63 % of Mem, and 8500 KBps because of contention. In this case, although its Mem- and disk-usage are respectively reduce by 10 % and 50 %, its final throughput will not reduce by $\max(10\,\%, 50\,\%) = 50\,\%$. This is because a VM's throughput is actually reduced based on its nature and in proportion to how sensitive it is to each of its resources. For blogbench in this example with $\text{Sen}^c = 0.00$, $\text{Sen}^m = 0.75$, and $\text{Sen}^d = 0.20$, we observed (measured) the final throughput of 83,460 that is very close to 82,000 that Eq. 1 predicts as:

$$\text{Thr}=\left(\frac{\frac{9}{10}\times 0.00+\frac{63}{70}\times 0.75+\frac{8500}{17000}\times 0.20}{0.00+0.75+0.20}\right)\times 100000 = 0.82\times 100000$$

In Sect. 5 we will show that using sensitivity values to weight average usage proportion of resources leads to accurate blind prediction of throughput for all benchmarks we used in this work.

4.3 VmModeler Procedures

Algorithm 1 shows procedural steps of modeling, and consequently deriving a formula to relate throughput of an application/VM to its CPU, Mem, and Disk utilization. Modeling can be performed in two modes: Blind or Assisted. In the Blind mode, it is assumed that vmBBThrPred has no knowledge of the application inside a VM, and it purely relies on the sensitivity values reported by vmBBProfiler ($\text{Sen}^{c/m/d}$) to predict throughput of the VM under different working scenarios. In the Assisted mode, it is assumes that there exists a "known" measurement/metric that could directly or indirectly reflect the actual performance of a VM. For example, the amount of network traffic for an apache server or the amount of IOPs (i/o operation per second) for an ftp server can both indirectly reflect performance of these servers. The Assisted mode is to address the current theme of using internal and/or external measurements to predict throughput, and consequently performance degradation of a VM in its current working condition. We included this mode only to show that not only vmBBThrPred can be easily adopted/employed by current systems, but also its bundling with vmBBProfiler yields more than 95 % accuracy in predicting throughout of any application with any resource sensitivity. Similar to vmBBProfiler [17], vmModeler also uses the normalized values of C, M, and D to propose a polynomial function with prototype

$$\text{Thr(C,M,D)} = \text{x}_1\text{C} + \text{x}_2\text{M} + \text{x}_3\text{D} + \text{x}_4\text{CM} + \text{x}_5\text{CD} + \text{x}_6\text{MD} + \text{x}_7\text{CMD} + \text{x}_8 \quad (2)$$

where C, M, and D are the current values of cpu.usage.average (%) divided by 100, mem.usage.average (%) divided by 100, and disk.usage.average (KBps) divided by 50000 (the maximum read/write speed for our testing environment), respectively.

To calculate $\text{x}_1 \ldots \text{x}_8$, we use ProfTable (Table 1) generated during calculation of $\text{Sen}^{c/m/d}$ by vmBBProfiler. In this table, for $nc = nm = nd = 4$ (where $c_x = m_x = d_x = x \times 0.25$, ProfTable would have 64 rows. Using these 64 runs, we define:

$$A = \begin{bmatrix} C_1 & M_1 & D_1 & (C_1M_1) & (C_1D_1) & (M_1D_1) & (C_1M_1D_1) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ C_{64} & M_{64} & D_{64} & (C_{64}M_{64}) & (C_{64}D_{64}) & (M_{64}D_{64} & (C_{64}M_{64}D_{64}) & 1 \end{bmatrix} \quad (3)$$

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_8 \end{bmatrix} \quad Y1 = \begin{bmatrix} \frac{\frac{C_1}{C_{64}} Sen^c + \frac{M_1}{M_{64}} Sen^m + \frac{D_1}{D_{64}} Sen^d}{Sen^c + Sen^m + Sen^d} \\ \vdots \\ \frac{\frac{C_{64}}{C_{64}} Sen^c + \frac{M_{64}}{M_{64}} Sen^m + \frac{D_{64}}{D_{64}} Sen^d}{Sen^c + Sen^m + Sen^d} \end{bmatrix} \quad Y2 = \begin{bmatrix} \frac{T_1}{T_{64}} \\ \vdots \\ \frac{T_{64}}{T_{64}} \end{bmatrix} \quad (4)$$

In Eq. 3, normalized values of C, M, and D for the k-th run in ProfTable are used to fill the k-th row of matrix A. In Eq. 4, each element/row of vector Y1 is the weighted average of relative CPU, Mem, and Disk utilization of the

Algorithm 1. Algorithm for vmModeler in both modes

1: **procedure** VMMODELER((Sen$^{c/m/d}$, ProfTable)) **Input** : Sen$^{c/m/d}$ and ProfTable → calculated and provided by vmBBProfiler
Output: ThrA(C,M,D) and/or ThrB(C,M,D)
2: Use ProfTable to Initialize Matrix A ▷ Eq. 3
3: Use ProfTable and Sen$^{c/m/d}$ to Initialize Matrixes Y1,Y2 ▷ Eq. 4
4: Calculate X for Y← Y1 and Build ThrB(C,M,D) ▷ Eqs. 5, 2
5: Calculate X for Y← Y2 and Build ThrA(C,M,D) ▷ Eqs. 5, 2
return ThrA and/or ThrB
6: **end procedure**

k-th run with respect to the 64-th run (the run with no limitation and maximum performance). Vector Y2, only for the assisted mode, records the relative performance value of an indirect-metric that can be used to directly or indirectly reflect the performance of a VM; it is assumed that T64 reflects the maximum throughput/performance. For example, we used disk.usage.average (KBps) (T64 = 46000 KBps) as the indirect-metric for aio-stress in our experimental setup (more information in Sect. 5). Using linear regression, the optimal value of X can be calculated as:

$$A_{64\times 8} \times X_{8\times 1} = Y_{64\times 1} \Longrightarrow X = (A^T A)^{-1} A^T Y \tag{5}$$

For Y←Y1, the X calculated using Eq. 5 yields ThrB (B for Blind) in Eq. 2; Y←Y2 yields ThrA (A for assisted) in Eq. 2.

In Algorithm 1, operations 2–3 initialize three matrices; operation 4 calculates and builds ThrB; operation 5 builds ThrA. Note that computing ThrB and ThrA are independent of each other; therefore if no "indirect-metric" could be identified to calculate ThrA, vmModeler can still build ThrB. In Sect. 5 we will show that ThrA is, as expected, more accurate (≈96 %) than ThrB (≈90 %) for all cases/benchmarks.

5 Experimental Results

To validate our proposed vmBBThrPred, we ran about 1200 h (50 days) of actual running and profiling benchmarks on our private cloud in the School of Information Technologies at the University of Sydney. We used three different PMs (Table 2) and profiled 12 different benchmarks (Table 5), varying from pure CPU/Mem/Disk intensive to various combination of CPU+Mem+Disk ones.

Benchmark Selection: We used the Phoronix Test Suite [3] (one of the most comprehensive testing and benchmarking platform) to evaluate performance and accuracy of vmBBThrPred. Table 5 lists the 12 benchmarks (out of 168 available ones in v5.2.1) we used for our experiments. We deliberately picked benchmarks with different intensities of resource usage profile of CPU, Mem, and Disk to cover realistic applications. In this table 'H', 'L', and '–' respectively mean High,

Low, and Negligible resource utilization. From the 12 benchmarks in Table 5, eight run CPU intensive, four run Memory intensive, and five run Disk intensive processes.

Experimental Results: Table 5 shows experimental results of using our approach (vmBBThrPred) to derive polynomial formulas for the selected 12 benchmarks. There are three rows for each benchmark: one row for each PM in Table 2. As it was explained in Sect. 4, vmBBThrPred can work in two modes: Blind and Assisted. ThrB is built purely based on $Sen^{c/m/d}$ and the ProfTable (Table 1); ThrA additionally uses the mentioned indirect-metric in Table 5.

6 Discussion and Analysis

We highlight the most stimulating conclusions from Table 5 in this section.

6.1 Accuracy of vmBBThrPred

Table 5 shows different prediction accuracy for different benchmarks: ranging from 76 %–99 % for the Blind (ThrB) and 94 %–100 % for the Assisted (ThrA) mode. For CPU intensive applications (marked as (*/–/–)), the accuracy of

Table 5. Results for using vmBBThrPred on the selected benchmarks

Benchmark	PM	Sen^c / Sen^m / Sen^d	ThrB(C,M,D)		ThrA(C,M,D)		indirect-metric
			formula	acc.	formula	acc.	
apache (H/–/–)	AMD	0.95 / 0.00 / 0.00	(1.02)C	95%	(1.03)C–0.05	98%	cpu.latency. average(%)
	DELL	0.97 / 0.00 / 0.00	(1.02)C	97%	(0.94)C+0.06	98%	
	SGI	0.97 / 0.03 / 0.00	(0.99)C	96%	(1.00)C+0.04	98%	
john-the-ripper (H/–/–)	AMD	0.93 / 0.00 / 0.00	(1.18)C	95%	(1.14)C+0.04	95%	cpu.latency. average(%)
	DELL	0.96 / 0.00 / 0.00	(1.09)C+0.01	95%	(1.06)C+0.08	97%	
	SGI	0.96 / 0.00 / 0.00	(1.17)C	91%	(1.11)C+0.13	95%	
n-queens (H/–/–)	AMD	0.95 / 0.00 / 0.00	(1.02)C	97%	(1.07)C–0.05	99%	cpu.idle.sum mation(msec)
	DELL	0.97 / 0.00 / 0.00	(1.01)C	99%	(1.02)C	100%	
	SGI	0.97 / 0.00 / 0.00	(1.02)C	99%	(1.03)C	100%	
build-apache (H/–/–)	AMD	0.94 / 0.00 / 0.00	(1.13)C+0.01	97%	(1.19)C–0.03	97%	cpu.latency. average(%)
	DELL	0.96 / 0.00 / 0.00	(1.05)C	99%	(1.07)C–0.01	99%	
	SGI	0.96 / 0.04 / 0.00	(1.10)C+0.01	98%	(1.16)C–0.01	99%	
build-php (H/–/–)	AMD	0.95 / 0.02 / 0.00	(1.01)C	98%	(1.02)C–0.01	98%	cpu.latency. average(%)
	DELL	0.96 / 0.00 / 0.00	(1.04)C	98%	(1.03)C+0.02	98%	
	SGI	0.97 / 0.07 / 0.00	(0.96)C+0.01	95%	(1.00)C+0.03	98%	
dcraw (L/–/–)	AMD	0.54 / 0.00 / 0.00	(2.10)C	98%	(2.38)C–0.11	99%	cpu.idle.sum mation(msec)
	DELL	0.55 / 0.00 / 0.00	(2.16)C+0.02	98%	(2.26)C–0.02	98%	
	SGI	0.48 / 0.04 / 0.00	(2.01)C	94%	(2.24)C–0.03	98%	
x264 (L/–/–)	AMD	0.33 / 0.01 / 0.00	(1.28)C	96%	(1.32)C–0.06	97%	cpu.latency. average(%)
	DELL	0.39 / 0.00 / 0.00	(1.27)C	95%	(1.34)C–0.08	98%	
	SGI	0.41 / 0.02 / 0.00	(1.28)C	98%	(1.30)C–0.02	98%	
unpack-linux (L/L/L)	AMD	0.19 / 0.10 / 0.40	(1.59)C+(1.61)D–(1.14)CD–0.07	95%	(1.76)C+(4.91)D–(6.04)CD–0.53	98%	disk.numwrite. summation
	DELL	0.21 / 0.09 / 0.25	(1.36)D+0.01	95%	(1.60)D–0.07	96%	
	SGI	0.18 / 0.09 / 0.35	(0.91)C+(1.09)D–(0.18)CD–0.01	94%	–(0.58)C–(0.26)D+(5.04)CD+0.33	95%	
blogbench (–/H/L)	AMD	0.09 / 0.74 / 0.16	(0.93)M	77%	(0.60)M+0.41	90%	mem.latency. average(%)
	DELL	0.00 / 0.75 / 0.20	(0.20)M+(0.64)D+(0.16)MD+0.37	76%	–(0.89)M+(0.40)D+(0.34)MD+0.98	90%	
	SGI	0.11 / 0.81 / 0.18	(0.15)C+(0.93)M–(0.03)CM	84%	(0.46)C+(0.20)M+(0.06)CM+0.28	91%	
bork (–/L/L)	AMD	0.00 / 0.47 / 0.18	(0.75)M+(0.26)D+(0.05)MD+0.02	84%	–(0.03)M+(0.94)D+(0.11)MD	99%	mem.activewri te.average(KB)
	DELL	0.00 / 0.45 / 0.09	(0.80)M+(0.39)D–(0.10)MD	82%	(0.03)M+(1.14)D–(0.03)MD+0.05	98%	
	SGI	0.00 / 0.53 / 0.20	(0.82)M+(0.43)D–(0.15)MD	83%	(0.04)M+(1.24)D–(0.08)MD–0.01	97%	
compress-gzip (–/L/H)	AMD	0.00 / 0.00 / 0.55	(1.37)D+0.07	94%	(0.85)D+0.40	97%	disk.numwrite. summation
	DELL	0.00 / 0.00 / 0.45	(0.48)M+(0.52)D+(0.22)MD+0.11	87%	(0.16)M+(1.01)D+(0.42)MD	94%	
	SGI	0.00 / 0.00 / 0.47	(0.51)M+(0.48)D+(0.25)MD+0.09	83%	(0.10)M+(1.08)D+(0.40)MD	95%	
aio-stress (–/–/H)	AMD	0.00 / 0.31 / 0.84	(0.60)M+(0.68)D–(0.02)MD	90%	(1.10)D+(0.33)MD	99%	disk.maxtotal latency.latest
	DELL	0.00 / 0.32 / 0.91	(0.40)M+(0.95)D–(0.11)MD–0.01	96%	(0.13)M+(1.22)D–(0.06)MD–0.05	98%	
	SGI	0.00 / 0.30 / 0.80	(1.00)M+(0.75)D–(0.19)MD–0.02	86%	–(0.05)M+(2.01)D+(0.75)MD+0.02	99%	

vmBBThrPred were significantly high (>94 %). Accuracy of ThrA/B for disk intensive applications ((–/L/L) and (–/*/H)) were also noticeably high with the minimum accuracy of 82 % and 94 % for ThrB and ThrA, respectively. Memory intensive applications, (–/H/*) and (–/L/*), proved to be much more cumbersome than the other two. In this case, vmBBThrPred accuracy dropped as low as 76 % and 90 % for the Blind and Assisted modes, respectively. This is well aligned with other experiments in the literature that identify Memory Caches (L1/L2/L3) as one of the most influential components in virtualized environments. It is also well aligned with Table 4 in which Memory bandwidth showed less correlation with throughput as compared with CPU and Disk.

6.2 Transferability of Results

Table 5 shows a variety of formulas for different benchmarks on different PMs. Nevertheless, in most cases the formula was almost identical across PMs. For CPU intensive applications, (*/–/–), ThrA/B are almost identical across PMs. Disk intensive applications, (–/L/L) and (–/*/H), have also led to similar formulas. Throughput of memory intensive applications however, (–/H/*) and (–/L/*), could not be modeled using similar formulas; they also varied across PMs. This could be related to the internal nature of CPU structure and the size of caches in these PMs. As it can be observed in Table 2, these PMs have different cache sizes. The AMD machine for example has the largest cache size; we believe this is why it has the most straight forward formula for all cases. For example, ThrA/B formulas for blogbench with the highest sensitivity to memory is calculated as a function of 'M', while on the other two PMs they are related to 'C' and 'D' too. This also confirms that the cache size/structure is very important for virtualized environments.

6.3 Indirect Metrics

Table 5 also shows the indirect-metric we used for each benchmark to build its ThrA formulas to achieve slightly better (5 %–10 % more) accuracy than ThrB. This proves that having "known" metrics to directly or indirectly measure performance of applications could in fact lead to more accurate results. Nevertheless, we argue that selecting the right "indirect metric" could not be very easy sometimes because not only we need to know the nature of the application/VM, but we also need to make sure that the chosen metric has a linear relation with the actual throughput of the application/VM. In fact, selecting a wrong metric could lead to meaningless formulas, such as selecting a disk related metric (e.g., disk.usage.average (KBps)) for a CPU-intensive applications (e.g., apache).

7 Conclusion

In this work, we presented vmBBThrPred to predict throughput, and consequently the performance degradation of general purpose applications/VMs based

on their CPU, Mem, and Disk utilization as seen by the Hypervisor, and the sensitivity values calculated for them by vmBBProfiler. vmBBThrPred can work in two modes: Blind and Assisted. In the Blind mode, it uses only the Hypervisor level metrics to derive a polynomial formula in which normalized CPU, Mem, and Disk utilization values of working VMs can be dynamically (online) plugged in to predict the immediate throughput of each VM. For the Assisted mode, an indirect-metric could be nominated by the user so that vmBBThrPred can derive more accurate formulas. vmBBThrPred was implemented in our VMware-vSphere based private cloud and proved its efficiency across 1200 h of empirical studies. Using 12 well known benchmarks to cover all sorts of possible applications, it managed to successfully build accurate formulas (90 % for Blind and 95 % for Assisted on average) for a various range of applications with different resource intensity usage profiles. vmBBThrPred is the first Black-Box throughput predictor, to the best of our knowledge, that uses only basic Hypervisor level metrics for its very systematic calculations.

References

1. Specjbb (2016). https://www.spec.org/jbb2015/
2. Microsoft powershell (2016). https://msdn.microsoft.com/en-us/mt173057.aspx
3. Phoronix test suite (2016). www.phoronix-test-suite.com/
4. Vmware-powercli (2016). www.vmware.com/support/developer/powercli/
5. Vmware-vcenter (2016). www.vmware.com/products/vcenter-server
6. Vmware-vsphere (2016). www.vmware.com/products/vsphere/
7. Banga, G., Druschel, P., Mogul, J.C.: Resource containers: a new facility for resource management in server systems (1999)
8. Bartolini, D.B., Sironi, F., Sciuto, D., Santambrogio, M.D.: Automated fine-grained CPU provisioning for virtual machines. ACM Trans. Architect. Code Optim. (TACO) **11**(3), 27 (2014)
9. Caglar, F., Shekhar, S., Gokhale, A.: Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud (2011)
10. Du, J., Sehrawat, N., Zwaenepoel, W.: Performance profiling of virtual machines. SIGPLAN Not. **46**(7), 3–14 (2011)
11. Hui, C., Shinan, W., Weisong, S.: Where does the power go in a computer system: experimental analysis and implications. In: 2011 International Green Computing Conference and Workshops (IGCC), pp. 1–6 (2011)
12. Kundu, S., Rangaswami, R., Dutta, K., Ming, Z.: Application performance modeling in a virtualized environment. In: 2010 IEEE 16th International Symposium on High Performance Computer Architecture (HPCA), pp. 1–10 (2010)
13. Lingjia, T., Mars, J., Vachharajani, N., Hundt, R., Soffa, M.L.: The impact of memory subsystem resource sharing on datacenter applications. In: 2011 38th Annual International Symposium on Computer Architecture (ISCA), pp. 283–294 (2011)
14. Mars, J., Tang, L., Hundt, R., Skadron, K., Soffa, M.L.: Bubble-up: increasing utilization in modern warehouse scale computers via sensible co-locations (2011)
15. Nathuji, R., Kansal, A., Ghaffarkhah, A.: Q-clouds: managing performance interference effects for QoS-aware clouds (2010)

16. Rao, J., Bu, X., Xu, C.Z., Wang, L., Yin, G.: VCONF: a reinforcement learning approach to virtual machines auto-configuration (2009)
17. Taheri, J., Zomaya, A.Y., Kassler, A.: vmbbprofiler: A black-box profiling approach to quantify sensitivity of virtual machines to shared cloud resources. ACM Trans. Model. Perform. Eval. Comput. Syst. (March 2016, submitted)
18. Watson, B.J., Marwah, M., Gmach, D., Chen, Y., Arlitt, M., Wang, Z.: Probabilistic performance modeling of virtualized resource allocation (2010)
19. Xu, J., Zhao, M., Fortes, J., Carpenter, R., Yousif, M.: Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. Clust. Comput. **11**(3), 213–227 (2008)