DeepDive: Transparently Identifying and Managing Performance Interference in Virtualized Environments

EPFL Technical Report EPFL-REPORT-183449

Dejan Novaković, Nedeljko Vasić, Stanko Novaković, Dejan Kostić[†], and Ricardo Bianchini[‡]
EPFL, Switzerland [†]*Institute IMDEA Networks, Spain* [‡]*Rutgers University, USA*

Abstract

Cloud computing in general, and Infrastructure-as-a-Service (IaaS) in particular, are becoming ever more popular. Unfortunately, performance interference (and the resulting unpredictability in the delivered performance) across virtual machines (VMs) co-located on the same physical machine (PM) threatens to make cloud computing inadequate for performance-sensitive customers and more expensive than necessary for all customers.

We describe the design and implementation of DeepDive, a system for transparently identifying and managing interference. DeepDive successfully addresses several important challenges, including lack of performance information from applications, and large overhead of detailed interference analysis. We first show that it is possible to use easily-obtainable, low-level metrics to clearly discern when interference is occurring and what resource is causing it. Next, using realistic workloads, we demonstrate that DeepDive quickly learns about interference across co-located VMs. Finally, we show DeepDive's ability to deal efficiently with interference when it is detected, by using a low-overhead approach to identifying a VM placement that alleviates interference.

1 Introduction

Many enterprises and individuals have been offloading their workloads to Infrastructure as a Service (IaaS) providers, such as Amazon and Rackspace. Major market actors have been anticipating even wider adoption in years to come [3]. A key enabling factor in the expansion of cloud environments is virtualization technology. IaaS cloud providers typically use virtualization to (1) package and identify each customer's application into one or more virtual machines (VMs), (2) isolate misbehaving applications, (3) lower operating costs by multiplexing their physical machines (PMs) across many VMs, and (4) simplify VM placement and migration across PMs.

Despite the benefits of virtualization, including its ability to slice a PM well in terms of CPU and memory space allocation, performance isolation is far from perfect in these environments. Specifically, a challenging problem for providers is identifying (and managing) *performance interference* between the VMs that are co-located at each PM. For example, two VMs may thrash in the shared hardware cache when running together, but fit nicely in it when each is running in isolation. As another example, two VMs, each with sequential disk I/O when running in isolation, may produce a random access pattern on a shared disk when running together. To make things worse, technology trends point to manycore PMs with hundreds or even thousands of cores. On these PMs, the chance of experiencing interference will increase.

Interference can severely diminish the trust of customers in the cloud's ability to deliver predictable performance. Thus, interference might become a stumbling block in attracting performance-sensitive customers.

Effectively dealing with interference is challenging for many reasons. First, the IaaS provider is oblivious to its customers' applications and workloads, and it cannot easily determine that interference is occurring. Moreover, the IaaS provider cannot rely on applications to report their performance levels (and therefore know when interference is occurring), because this might overburden application developers who moreover cannot be trusted. This challenge speaks against non-transparent approaches [14, 19, 26, 27, 28, 33, 35]. Second, interference is complex in nature and may be due to any server component (e.g., shared hardware cache, memory, I/O). An effective solution has to account for all components. Further, interference might only manifest when the co-located VMs are concurrently competing for hardware resources. The existing approaches for predicting performance degradation [14, 19, 26, 27, 35] are not applicable, as they require the provider to have access to the co-located VMs for long periods prior to deployment. Interference detection must be a quicker, online activity. Finally, the sheer volume of new VMs deployed daily at a large public provider may cause scalability issues.

Given these challenges and limitations of the prior work, we propose DeepDive, a system for transparently and efficiently identifying and managing performance interference in IaaS providers. Our contributions are:

1. A method for transparently obtaining the ground truth about interference, including a black-box detection of application behavior and the ability to pinpoint the culprit resource for interference using only low-level metrics.

2. A *warning system* that reduces the overhead of detailed interference analysis by learning about normal, non-interfering behaviors.

3. A technique for leveraging global information to increase scalability that uses the behavior of VMs running

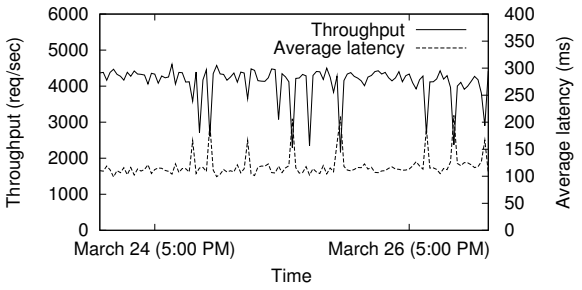


Figure 1: Measured performance of a service running on EC2 under a fixed workload and resource configuration. Performance is periodically affected by co-located VMs.

the same workload on other PMs rather than detailed and expensive interference analysis.

4. A mechanism for transparently and cheaply migrating the culprit VM, by using a simple synthetic benchmark to mimic the low-level behavior of a VM and its impact on other VMs before actual migration.

5. Evaluation using realistic cloud applications and workloads that shows: i) DeepDive transparently infers performance degradation with high accuracy (less than 5% error on average), identifies interference, and pinpoints the culprit resource; ii) DeepDive is highly accurate (no false negatives) and has low overhead (needs just several dedicated profiling machines even under extreme new-VM arrival scenarios); and iii) DeepDive makes quick and accurate VM placement decisions (less than a minute).

To the best of our knowledge, DeepDive is the first end-to-end system that can transparently and efficiently handle interference on any major server resource, including I/O. Its deployment would have two key benefits. First, it would enable cloud providers to meet their service-level objectives (SLOs) using fewer resources, which would increase user satisfaction and reduce energy costs. Second, the smarter and more efficient VM placement would enable cloud customers to purchase fewer resources from the provider.

2 Background and Motivation

Virtualization software chronically lacks effective performance isolation, especially in the context of hardware caches and I/O components. For instance, recent efforts [17] reveal that interference may cause same-type VMs (e.g., those offering the same amount of virtual resources) to exhibit significantly different performance levels over time. This impact can be seen in our experiment using Cassandra [9] (a key-value store) running on Amazon EC2. We deploy one Cassandra VM and monitor its performance under a fixed workload and resource allocation during a three-day period. As shown in Figure 1, although both the workload and virtual resources remain the same, Cassandra faces many periods of significantly degraded performance. We attribute the performance drops to interference because we tightly control the experiment, except of course for the virtualization platform and the PM, where interference can occur.

When faced with such performance degradation, users might compensate by overprovisioning their VMs [27, 28, 33], which increases their costs. However, increasing the virtual resources is not a panacea, especially for "scale out" applications that dynamically increase the number of running VMs while keeping the instances affected by interference in the active set. As a result, many (potential) cloud customers still find interference as a key barrier to migrating their loads to the cloud [7].

3 Challenges

Dealing with the interference-induced performance degradation in virtualized environments is hard for the following reasons.

Transparency is paramount. While relying on the customer application to report performance-level metrics would make interference detection easier, such an approach would face at least a few deployment obstacles. First, cloud customers might become uncooperative (i.e., by under reporting performance levels) for purely selfish or sometimes even malicious reasons. Second, considering the diversity of the applications running in the cloud, the provider cannot rely on prior application knowledge; such an approach would not scale as the number of cloud tenants increases. Thus, we argue for an approach that transparently manages interference.

Interference is complex in nature. Previous works and our own experience suggest that all the server components may contribute to the overall performance degradation. For instance, one VM may suffer because another VM aggressively pollutes the shared hardware cache, or because they affect each other's I/O activities. Therefore, omitting any hardware component from the interference diagnosis would most likely lead to incorrect results.

An efficient, on-line approach is required. Many recent efforts [14, 26, 27, 35] demonstrate that application classification based on its sensitivity and aggressiveness – i.e., how much it suffers when co-located with other applications and how much damage it causes to them – may accurately predict the application's performance degradation due to co-located VMs. In practice, however, cloud operators typically do not have access to the applications prior to their deployment, so they cannot easily conduct the sensitivity/aggressiveness analysis.

To make things worse, it is hard to characterize an application's potential for interference in the absence of the other co-located VMs. For instance, when running alone, a workload might not be memory-intensive (i.e., its working set fits in the shared cache), but when co-located with

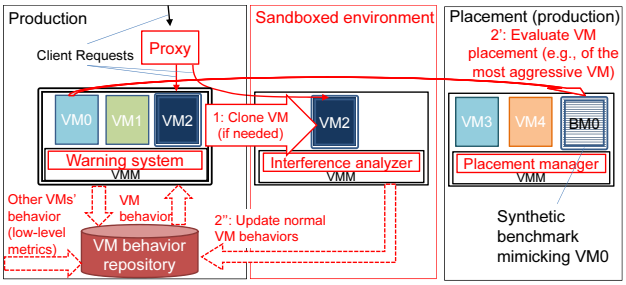


Figure 2: DeepDive overview, showing how it detects and mitigates the effect of interference on VM2.

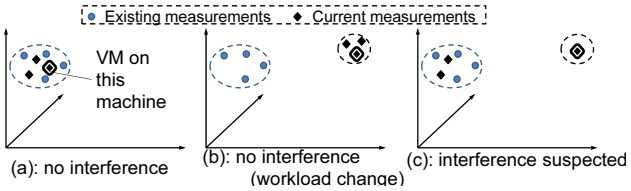


Figure 3: The warning system uses previously collected data and current global measurements, to decide whether DeepDive should further investigate interference.

certain other workloads it might become so. These reasons suggest an on-line approach that can characterize interference in realistic scenarios.

Given these challenges, we investigate whether it is possible to transparently and efficiently diagnose whether and how a VM is being affected by performance interference.

4 Approach

DeepDive operates in parallel with applications, seeking to provide application performance that is comparable to, or ideally the same as, that observed in an isolated environment. Figure 2 highlights DeepDive's main components and the way they interact. DeepDive transparently deals with interference by inspecting low-level metrics, including hardware performance counters and readily available hypervisor (VMM) statistics about each VM. To reduce the overhead of interference detection and mitigation, DeepDive introduces two interference analyses that differ in their accuracy and overhead.

DeepDive first relies on a **warning system** running in the VMM to conduct early interference analysis. This analysis is fast, and induces negligible overhead as we can collect the required statistics without affecting the applications currently running on the PM. DeepDive places the measurements it collects in the multi-dimensional space, where the interference and non-interference cases cluster into easily separable regions.

Figure 3 depicts the decision-making process in the warning system by illustrating the important cases in the multi-dimensional space (shown here only using three dimensions for clarity). One option is for the current measurements to fall within a cluster of acceptable behaviors (Figure 3(a)). If that is not the case but other VMs running this workload are behaving similarly (e.g., due to a change in the client-induced workload), again there is no need to perform further interference analysis (Figure 3(b)). Further investigation is required only if the current measurement is substantially different (i.e., by more than an automatically-determined threshold) from both the existing behaviors as well as other VMs (Figure 3(c)). In the absence of other VM behavior (single-VM case), the reasoning is simplified to cases depicted in Figures 3(a) and 3(c).

While the warning system reduces DeepDive's overhead, it is not perfectly accurate and cannot pinpoint the source of interference. DeepDive thus relies on an **interference analyzer** to perform a highly reliable but expensive analysis, when necessary. Only when the warning system suspects that one or more VMs are subjected to interference, DeepDive invokes the analyzer to conduct the exhaustive interference analysis.

The analyzer clones the VM on-demand and executes it in a sandboxed environment. By using a proxy to duplicate client requests, the cloned VM is subjected to the same workload as the VM co-located with other tenants. The analyzer then uses the low-level measurements to estimate performance of the original and cloned VMs. The performance estimates should be similar – different by less than an *operator-defined threshold* percentage – in the absence of interference. This VM cloning, workload duplication, and comparison approach has been studied extensively in [33, 34]. The approach provides the ground truth, and enables DeepDive to pinpoint the dominant sources (server components) of interference. The analyzer uses the classic cycles per instruction (CPI) model to transparently identify these sources. Researchers have used this model to detect performance issues other than interference, e.g. [10]. We augment the model with system-level metrics that extend the CPI stack to include I/O.

In the absence of interference, the analyzer updates the repository of VM behaviors with this new information. If interference does exist, the analyzer forwards its findings to the **VM-placement manager** to determine a preferable (e.g., minimal) change in VM placement that will eliminate or at least reduce interference. The default behavior is to migrate the most aggressive VM, in terms of its use of the resource that is causing interference.

The VM-placement manager tries to find a PM that will be the best match (e.g., non-interference causing) for the VM at hand. It does so by running a synthetic benchmark that mimics the behavior of the VM for a short time on another machine (with other VMs present),

Name	Description	Name	Description
cpu_unhalted	Clock cycles when not halted	*resource_stalls*	Cycles during which resource stalls occur
inst_retired	Number of instructions retired	*bus_tran_any*	Number of completed bus transactions
l1d_repl	Cache lines allocated in the L1 data cache	*bus_trans_ifetch*	Number of instruction fetch transactions
l2_ifetch	L2 cacheable instruction fetches	*bus_tran_brd*	Burst read bus transactions
l2_lines_in	Number of allocated lines in L2	*bus_req_out*	Outstanding cacheable data read bus requests duration
mem_load	Retired loads	*br_miss_pred*	Number of mispredicted branches retired
iostat	T_{disk} presents all the idle CPU cycles while the system had an outstanding disk I/O request.		
netstat	T_{net} presents all the idle CPU cycles while the system had a packet in the Snd/Rcv queue.		

Table 1: Low-level metrics used to differentiate normal VM behaviors from interference. The $iostat$ and $netstat$ tools can be used to approximate I/O-related stalls associated with different VMs, using VM introspection tools like XenAccess. Note that the table has two columns of metrics names and descriptions.

and evaluates whether interference reappears. If it does not, DeepDive can migrate the VM to that machine. If interference reemerges, the VM-placement manager tries a different PM. Fundamentally, DeepDive's ability to reproduce VM behaviors derives from its sole reliance on low-level metrics.

4.1 The warning system

The warning system prevents unnecessary interference analyzer invocations by differentiating workload changes from interference. It does so based on the metrics listed in Table 1. These metrics represent the major PM resources (CPU cores, memory, disk, and network interface), and have been enough for our experiments to date. Vasić *et al.* [33] considered an even larger set of metrics, but found the larger set to be overkill.

The system uses both local and global information to infer if interference may be happening. It first locally tries to match the current values of the metrics against the previously learned set of normal behaviors. If it cannot find a match, it globally checks whether other VMs running the same code are experiencing similar behavior.

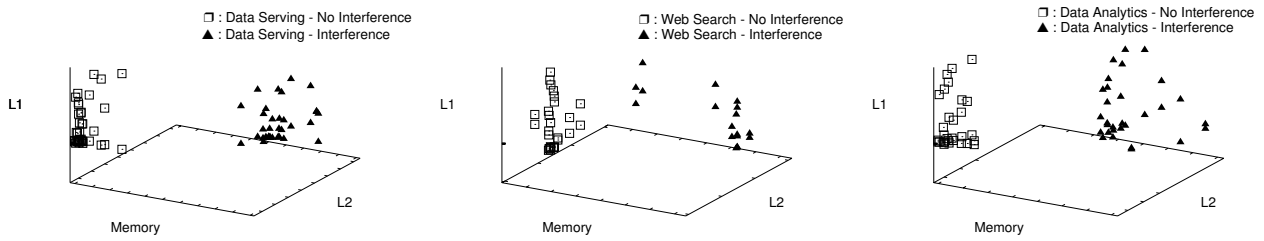
More precisely, when first faced with a VM, the warning system has no information about it and activates the interference analyzer. The analyzer then provides the warning system with: i) a set of normal VM behaviors S that are obtained in isolation, and form the ground truth, and ii) a vector of *metric classification thresholds* M_T used to filter out the workload noise from actual interference. Note that these classification thresholds are different from the operator-defined performance threshold for acceptable performance degradations (Section 4.2), and are set automatically by the clustering algorithm (described below). From this point on, the warning system continuously collects the metrics and tries to retrieve a match from the set of normal VM behaviors, respecting the acceptable metric deviations M_T. The detailed warning system algorithm appears in the appendix.

Like any other statistical method, the warning system can only identify performance anomalies (interference) if they are exceptional. Fortunately, our measurements performed on a real-world platform (Figure 1) suggest that anomalies are indeed exceptional in practice. Even if performance anomalies were common for an application, i.e. they cannot be used to detect that the application is undergoing interference, DeepDive would eventually learn so via invocations of the interference analyzer.

To prevent VM load changes unrelated to interference from causing analyzer invocations, we normalize the metrics with respect to the amount of work performed (the number of instructions retired). We find that the metrics' normalized values are persistent across a wide range of load intensities. This finding is critically valuable, since cloud loads frequently fluctuate over time.

Local information. To demonstrate experimentally that the warning system can differentiate normal from interference behaviors, we use typical cloud workloads under different quantitative and qualitative load changes, and interference conditions. Specifically, in Figure 4, we extensively experiment with the Data Serving, Web Search, and Data Analytics workloads from CloudSuite [21]. (More details about these workloads appear in Section 5.) Although we collect the dozen or so metrics listed in Table 1, the figure includes only three of them for clarity. The figure presents normalized metric values relating to the first-level cache (L1), the second-level cache (L2), and main memory. Each point in the graphs depicts a different experimental setting, including various load intensities, and different key and word popularities for Data Serving and Web Search, respectively. In the absence of interference, the data points cluster on one side of the space. Once we inject differently modulated interference effects, the normalized metric values experience significant deviation, which allows the warning system to detect new interference conditions. (We detail the interfering VM in Section 5.1.)

Global information. To further reduce the number of invocations of the interference analyzer, the warning system leverages the fact that cloud applications regularly execute the same code on many (perhaps dozens or even



(a) Data Serving (b) Web Search (c) Data Analytics

Figure 4: Metric values when running under different workload and interference scenarios.

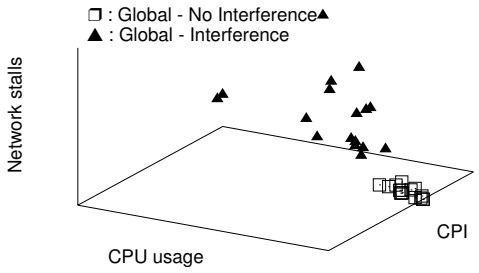


Figure 5: Metric values for Data Analytics. Observing multiple VMs prevents unneeded invocations of the analyzer.

thousands of) VMs. This scaling out enables the warning system to diagnose if the observed deviations come from interference effects or application behavior changes. If the VMs executing the same application code, spread across multiple PMs, observe similar metric value deviations at about same time, it is highly likely that the application is subjected to workload changes and further interference analysis is not necessary. Furthermore, DeepDive considers several metrics, which further reduces the chance that multiple VMs reporting similar behavior is a consequence of interference.

To illustrate the use of global information, we perform a set of experiments with our Data Analytics workload running across nine PMs in our cluster. We inject varying amounts of network interference into the cluster by progressively co-locating more interfering VMs that run a network-intensive benchmark (*iperf*). This scenario stresses the warning system because interference manifests only when the mappers and reducers (from the Hadoop MapReduce-like framework) have to fetch data remotely. Figure 5 plots some of the normalized metrics (relating to network and core utilization) obtained from each of the PM's local warning systems. The metrics corresponding to the PMs where we run the interfering VMs clearly deviate from the remaining VMs' behaviors. The figure hence demonstrates that DeepDive: i) deals with I/O-related interference, and ii) can further minimize the profiling overhead by merely observing the behavior of VMs running the same workload on different PMs.

DeepDive's ability to use global information relies on the assumption that it knows which VMs are running the same application. This is a reasonable assumption, since VMs can be rented in a pre-configured state. Moreover, cloud providers often provide load balancing functionality that tenants explicitly request from the cloud provider for groups of VMs that execute the same code.

Finally, note that one can automatically determine whether a metric should be included as a dimension in DeepDive's space; Vasić *et al.* have solved a similar feature selection problem [33].

False positives and false negatives. False positives occur when the warning system unnecessarily invokes the analyzer under non-interference conditions. For instance, changes in a VM's working set or qualitative workload changes (e.g., the request mix substantially shifts) may lead to substantial statistical variation. Although false positives may sporadically lead to unnecessary analyzer invocations, they are mostly benign and only marginally affect DeepDive's overhead. We have verified this empirically by running extensive experiments under realistic workload conditions.

On the other hand, if the warning system confuses interference with normal workload changes – a false negative – the impact is more severe. Fortunately, our sensitivity analysis demonstrates that the vector of metric thresholds M_T determined by a standard clustering technique (described below) prevents false negatives, while still maintaining high warning system efficiency. Moreover, cloud providers might periodically (e.g., at a frequency driven by VM priority) invoke the analyzer to even further reduce the false negative rate.

Clearly, the challenge here is to define metric thresholds $\mathbf{M_T}$ that properly separate representative VM behaviors from background noise, while also properly identifying interference. If the thresholds are too strict, even minor deviation from prior VM behaviors would cause the warning system to fire. On the other hand, excessively loose thresholds might let interference proceed undetected. We leverage the *expectation-maximization* clustering algorithm [22] to produce interference-free clusters in N-dimensional space, where N is the number of low-level metrics that DeepDive uses. In producing the clusters, the algorithm also defines the metric thresh-

olds. DeepDive enhances the clustering results by providing a set of constraints [11, 13] along with the collected VM behaviors – when diagnosing a VM's behavior with interference, the analyzer also prevents the algorithm from assigning this behavior to an interference-free cluster. This has a positive effect on the detection rate, as we have verified empirically.

Shortly after a VM's deployment, the metric space is empty or sparsely populated. To create the interference-free clusters, the warning system operates in a conservative mode – every drop in VM performance above the performance threshold causes invocation of the analyzer. This is how DeepDive ensures that no interference goes undetected, and accelerates learning of the interference-detecting metric thresholds.

4.2 The interference analyzer

If the warning system suspects that one or more VMs may be facing interference, it invokes the analyzer to confirm. To do so, the analyzer uses VM cloning, workload duplication, and VM performance comparison. If interference is indeed present, the analyzer also determines which resource is the most likely to be causing the interference (e.g., shared cache, I/O).

Identifying the ground truth. DeepDive uses the same approach to determine VM performance in the absence of interference as DejaVu [33]. Though we do not claim any novelty in this approach, we summarize it here for completeness. DeepDive clones the VM under test in a sandboxed environment that uses non-work-conserving schedulers to tightly control the resource allocation. The amount of time to complete VM cloning depends on the amount of state in the VM, but is typically small compared to the frequency of invocation of the analyzer. DeepDive relies on a proxy that intercepts the clients' traffic to: 1) duplicate and send copies of the requests to the sandboxed environment, and 2) forward the traffic to/from the production VM to avoid negatively impacting the applications running inside that VM. DeepDive can then compare the metrics in isolation and in production. Previous works [33, 34] have studied this overall approach and its challenges (including non-determinism) extensively, so we do not repeat this study here.

Performance analysis. Given the statistics from the production and sandboxed environments, DeepDive uses the analyzer's performance model to transparently estimate the performance degradation that a VM is experiencing due to interference. Given this model, DeepDive can opt for VM migration if the degradation is substantial, or refrain from any action otherwise.

Since we do not expect the VMs to assess and communicate their performance levels, the key question here is knowing when the VM's performance is degraded by simply looking at low-level metrics. The analyzer contrasts the *instructions retired* rate in production with that in isolation (in the sandbox) to approximate how much the shared resources contribute to the overall degradation: $Degradation = Inst^{production}/Inst^{isolation}$.

Once the analyzer estimates the degradation, it may proceed in one of two ways. If the degradation is below the operator-defined performance threshold, the analyzer notifies the warning system about the false alarm. This extends the warning system's set of acceptable VM behaviors with the new metrics' values. If the degradation exceeds the threshold, the analyzer forwards the results of its analysis to the VM placement manager, which may migrate the VM to a more appropriate PM.

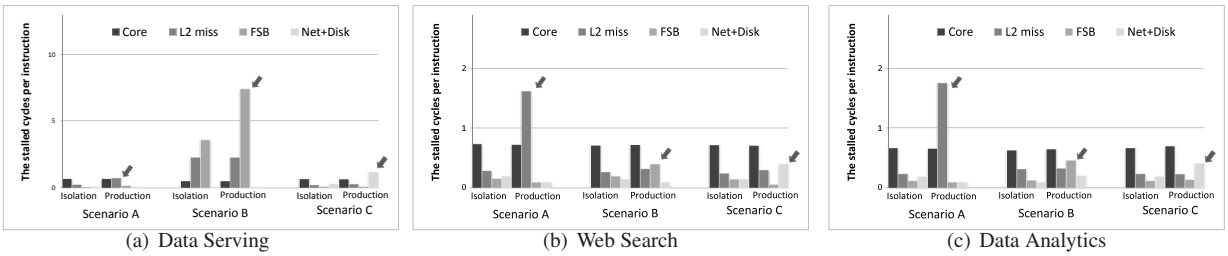
Importantly, [8, 20] have shown that the number of instructions retired is not always a reliable performance metric in multithreaded applications, since spin-based synchronization may cause timing and thread interleaving variations. This is not a serious problem for DeepDive for two reasons. First, the computed degradation need not be accurate with respect to absolute performance; rather, it simply needs to properly identify anomalies. Second, if these inaccuracies become a problem in practice, we can leverage prior efforts that exclude spinning instructions, or augment the measurements to account only for the useful computation [20]. Multithreading has not been a problem for us so far.

Identifying dominant sources of interference. If the amount of performance degradation requires invocation of the VM placement manager, the analyzer further pinpoints the resources that are likely the source of interference by leveraging CPI analysis augmented with system-level metrics (to capture I/O) about observed VMs. The augmented CPI "stack" effectively and transparently captures the amount of work the VM is doing, while identifying where the VM is spending time. Intuitively, interference causes the VM to suffer more stall cycles, and perform less useful work.

Our root cause analysis hence estimates a breakdown of the various run-time stall components of the server:

$$T_{overall} = \underbrace{T_{core} + T_{off\_core}}_{\text{CPI analysis using hardware counters}} + \underbrace{+T_{disk} + T_{net}}_{\text{using system-level statistics}}$$

where T_{core} represents the time running instructions on the core (and hitting in private caches), T_{off_core} represents the stalled cycles due to memory accesses (including shared caches), T_{disk} represents the time waiting for disk, and T_{net} represents network-related stalls. We infer these values from the metrics in Table 1. The metrics are clearly architecture-dependent, but sufficiently generic for DeepDive not to be tied to any particular architecture (our Appendix contains the results of running DeepDive on the Intel Core i7 architecture).



(a) Data Serving (b) Web Search (c) Data Analytics

Figure 6: Breakdown of stalled cycles in production and isolation. Our analysis reveals the sources of interference.

We estimate the resources' individual contributions to the performance degradation via the discrepancies in the metrics obtained in isolation and production:

$$Factor_{resource} = \frac{T^{production}_{resource} - T^{isolation}_{resource}}{T^{production}_{overall}}$$

To validate this performance model, we run a set of experiments with the Data Serving, Web Search, and Data Analytics workloads. Figure 6 contrasts the various resource stalls in the production environment (which is undergoing interference) and in isolation (in the sandbox). Each experiment carefully tunes the interference, so as to move it from the last level cache (Scenario A) to the front side bus (Scenario B) to the I/O subsystem (Scenario C). We then invoke the analyzer to estimate the amount of performance degradation, and identify the resources that primarily contribute to the degradation. We mark the resources identified by the analyzer with arrows in the figure. We observe that the analyzer correctly identifies the resources that primarily contribute to the performance degradation as their growing (degrading) factors clearly dominate over the remaining resources.

4.3 The VM-placement manager

If the analyzer detects interference on a PM, DeepDive runs the VM-placement manager to determine a new VM placement that will mitigate the interference.

The manager can implement multiple policies for selecting which VM to migrate: it may select the VM that is suffering the most from interference, or it may select the VM using the culprit resource most aggressively. Although we view the placement policy as orthogonal to this work, we design a simple interference-mitigating strategy to evaluate our placement manager. Upon identifying a resource that is the source of interference, the placement manager selects the VM that is most aggressive in using the resource, and then migrates it if an appropriate destination PM exists. To ensure better performance isolation, DeepDive repeats this process until the performance interference is sufficiently reduced, or ideally eliminated altogether.

The remaining challenge is ensuring that a VM migration will not cause even worse interference on the destination PM. A naive placement manager might speculatively migrate the selected VMs in the hope that this will not cause further interference on the destination PMs. However, this could result in numerous and expensive VM migrations (especially for applications with large memory and/or persistent state), as well as prolonged periods of severe performance degradation. DeepDive therefore anticipates the resulting interference conditions on the destination PM prior to actual VM migration.

Toward this end, DeepDive uses a novel synthetic benchmark that can mimic the behavior of an arbitrary VM. The key goal of our benchmark is that an actual VM and its synthetic counterpart should exhibit similar interference characteristics, when co-located with other VMs running on a PM. The benchmark models the working set size, data locality, instruction mix, level of parallelism, and disk and network throughput of a VM. In more detail, our benchmark is a collection of loops that exercise the different PM resources to match the metric values collected from an actual VM. The resources can be exercised locally to a PM, except for the network interface. For this resource, the benchmark spawns a thread that acts as a communication partner for a benchmark running on another PM. The loops execute numbers of iterations given as inputs to the benchmark. Thus, creating the benchmark involved learning the set of input values that best approximates any set of metric values. We used a standard regression algorithm for this training task. Though the training phase may take a long time (a few days in our experiments), this training is done *only once for each server type*. Moreover, one can use existing, more sophisticated workload synthesizers; we find this extra sophistication unnecessary for our needs.

The placement manager uses the benchmark to evaluate potential migrations. Specifically, given a set of metric values to reproduce, it runs the benchmark (with the proper learned inputs) in a VM on all candidate PMs concurrently. The runs take less than a minute in our experiments. With metric data collected from these runs, the manager picks the best destination PM for the migration.

Finally, note that DeepDive could benefit from having access to additional information in selecting a candidate VM for migration and a candidate destination PM. For example, being aware of the architecture of the PMs (e.g., as in Barrelfish [12]), or the way VMs are mapped to physical cores by the scheduler could help. However, we have not yet seen the need for this extra information.

4.4 Discussion

Can DeepDive tackle interference due to an oversubscribed network? Currently, DeepDive can tackle interference at the network interface, but requires a well-provisioned connection to the sandbox to determine the impact of network oversubscription. This is not a major constraint, since the number of PMs required for the sandbox is small, as we demonstrate in the next section.

Can DeepDive deal with non-determinism? DeepDive can tolerate deviations coming from different sources, such as OS-level non-determinism (e.g., periodic flushing of dirty pages). DeepDive views such non-deterministic events as noise, as they are typically too short and infrequent. Nevertheless, if they are persistent across multiple monitoring epochs, DeepDive is able to recognize this and label the behavior as normal.

Can DeepDive deal with oscillating interference conditions? While we have not focused on possible interference oscillations in this work, interference might vary over time. This would require us to repeat the interference analysis to ensure better guarantees on interference detection. In fact, we could install a simple controller that would react only upon detections that are persistent across multiple epochs.

Can DeepDive deal with heterogeneity? Our experience so far has been with homogeneous PMs. This is reasonable since cloud providers typically use disjoint sets of homogeneous PMs for simpler management. Nevertheless, DeepDive can deal with heterogeneity by grouping the low-level metrics by PM type, performing the CPI analysis according to PM type, and training a synthetic benchmark for each PM type.

Can DeepDive benefit from collaboration with the hypervisor? Both the hypervisor and DeepDive individually address some aspects of resource allocation, and their synergistic interaction indeed has great potential. For instance, the hypervisor scheduling could decide to postpone its moving of a VM from one core to another until DeepDive finishes its current measurement epoch. This would substantially reduce the measurement noise and enable the warning system to trigger alarms even faster.

Can DeepDive be ported to different hardware architectures? To demonstrate DeepDive's ability to operate on different architectures, one of the authors ported DeepDive to a NUMA (non-uniform memory access) server with two quad-core Core i7-based Xeon (E5640) processors. Both processors (each core runs at 2.67 GHz, with a 1-MB L2 cache) have their own integrated memory controller, and a 12-MB L3 cache. The server also features Intel's QuickPath Interconnect which replaces the front-size bus that characterizes the older Xeon servers. The port took just a few days to complete, with most of the time consumed on designing a new performance model starting fresh from the CPU/server datasheets. We believe that this additional engineering cost is not an issue, since it is amortized across a large number of VMs running on the same server type.

We repeated most of our experiments (e.g., Figure 4) on the new platform. Figure 7 shows that it is easy to identify interference in the new environment as well.

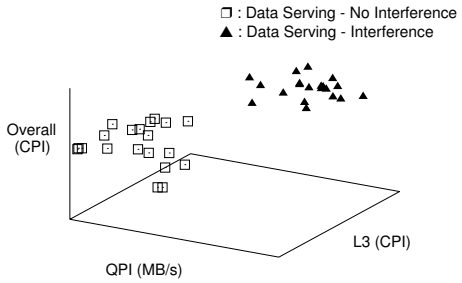


Figure 7: Metric values with and without interference for the Data Serving workload running on our i7-based server.

5 Evaluation

5.1 Experimental infrastructure

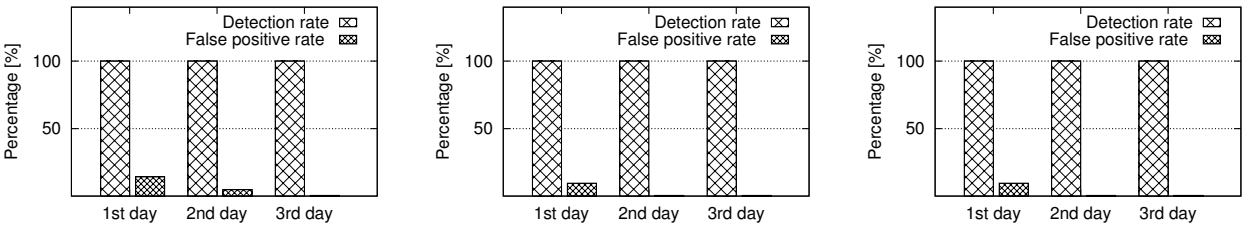
Servers and clients. We run our production and sandboxed environments on up to 10 servers with Intel Xeon X5472 processors. The servers have eight 3-GHz cores, with 12 MB of L2 cache shared across each pair of cores. The servers also feature 8 GB of DRAM, two 250-GB 7200rpm disks, and one 1-Gb network port.

The servers run the Xen VMM. We configure the VMs to run on virtual CPUs that are pinned to separate cores (we assign two cores per VM). We allocate enough memory for each VM to avoid swapping to disk.

The clients run on a separate machine with four 12-core AMD Opteron 6234 processors running at 2.4 GHz, 132 GB of DRAM, and two 1-Gb network ports.

Cloud workloads. We use diverse, representative cloud workloads from CloudSuite [21]. Our *Data Serving* workload consists of one instance of Cassandra [9], a distributed key-value store. To experiment with different loads, we instrument clients from the Yahoo! Cloud Service Benchmark [16] to vary both the key popularities and the read/write ratio.

Our *Web Search* workload involves a single index serving node (available from the Nutch open-source project [2]) that holds a 2GB index. To experiment with different loads, we instrument the Faban client emulator [4] to vary word popularities and the number of client sessions (driven by the traces described below).



(a) Data Serving (b) Web Search (c) Data Analytics

Figure 8: Detection and false positive rates while replaying the HotMail traces. DeepDive always detected the injected interference. The false positive rate quickly decreases as DeepDive learns more about normal behaviors.

Our *Data Analytics* workload uses Hadoop [5] to run a modified Bayes classification example from the Mahout package [1] across 35 GB of Wikipedia data. The cluster consists of nine VMs configured with 2 GB of memory and two dedicated cores, and the master which is provisioned with 8GB of RAM and four cores.

Real-world traces. To evaluate DeepDive under dynamic workloads, we use real load intensity traces to drive the execution of our cloud workloads. Specifically, we use traces from Microsoft's HotMail from September, 2009. The traces represent the aggregated load across thousands of servers, averaged over 1-hour periods. We ensure that the maximum number of active client sessions is within the servers' maximum capabilities.

In addition to load traces, we injected interference conditions mimicking a real cloud platform. Specifically, we rented four Amazon EC2 instances and let our Data Serving workload run for a three-day period. During this period, we continuously measured the performance reported by our client emulator. Whenever the client reported performance degradation of at least 20%, we labeled these performance crises as interference. We later use the time slots corresponding to the cloud's performance crises to drive our stress workloads (described below) on a co-located VM while replaying the traces. We further quantify the cloud's performance crises and use this information to drive the inputs of our stress workloads so as to cause similar performance degradation with respect to the particular VM we are stressing.

Using the clients' measured performance levels (e.g., response time), we evaluate DeepDive's ability to identify interference conditions that lead to measurable performance degradation. The clients label certain performance degradation as due to interference only if the amount of degradation is larger than 20%. In Section 5.3, we demonstrate that DeepDive is capable of dealing with arbitrary interference conditions.

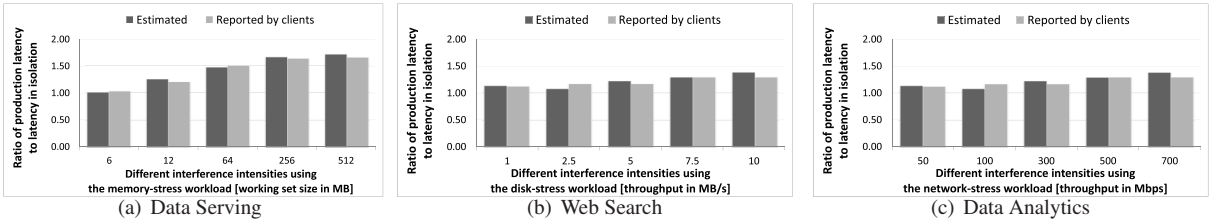
Interfering workloads. We evaluate DeepDive with three interfering workloads. Our *memory-stress* workload is inspired by the stress test introduced by Mars *et al.* [27]. It aggressively exercises shared resources, like last-level caches and the memory controller. The workload takes the desired working set size as an input. We also use *iperf* as our *network-stress* workload. This workload takes the desired network throughput as an input, and creates bi-directional UDP data streams to exercise network resources accordingly. Finally, we designed a simple *disk-stress* workload that copies files from one source to another, while respecting the maximum transfer rate defined as an input.

5.2 How accurate is the warning system?

To demonstrate the effectiveness of the warning system, we clear the set of VMs' behaviors S before each experiment. We therefore rely on the warning system to detect interference by merely using the information it obtained from the analyzer in the previous steps, as described in Section 4. Figures 8(a) to 8(c) plot the detection rate and the false positive rate of DeepDive while running our cloud workloads. The detection rate measures DeepDive's consistency in identifying interference, whereas the false positive rate reflects scenarios where the warning system unnecessarily invoked the analyzer. In this set of experiments, we use memory-stress to generate interference, and we further vary the working set size to reproduce interference amounts that we obtained from our experiments on Amazon EC2. Because this workload primarily affects memory-related metrics that vary at a fine grain, this is the most challenging scenario for DeepDive to separate normal from interference conditions.

The figures show that DeepDive reliably identifies the interference, each time VM performance is substantially affected by the co-located VMs. Besides the detection rate, the number of analyzer invocations is important, as it determines DeepDive's overhead. On the first day after deployment, DeepDive shows a fairly high false positive rate, as it is still learning the normal behaviors. Starting from the second day, this rate drops to near-zero, as the warning system recognizes behaviors it has seen earlier. We did not observe false negatives in our experiments.

Importantly, recall that false positives do not result in unnecessary VM migrations, since the interference analyzer will realize that these metric deviations correspond to workload changes, rather than interference.



(a) Data Serving (b) Web Search (c) Data Analytics

Figure 9: DeepDive accurately and transparently estimates performance loss from the metrics' values.

5.3 How accurate is the analyzer?

We now run experiments to demonstrate that DeepDive accurately estimates performance degradation under various interference conditions. We use client emulators for our workloads that continuously report average performance, enabling us to compare the client-reported degradations with those estimated by the analyzer.

We run the experiments at the maximum-possible request rate. We allow the servers to warm up for several minutes and start reporting stable performance. At this point, we launch the stress workloads on a co-located VM to inject various interference levels. Influenced by the nature of our cloud benchmarks, and the server components they primarily exercise, we co-locate: i) the memory-stress workload with Data Serving, ii) the network-stress workload with Data Analytics, and iii) the disk-stress workload with Web Search. We vary the interference intensity by varying: i) the working set size of memory-stress from 6 MB to 512 MB, ii) the throughput of network-stress from 50 Mbps to 700 Mbps, and iii) the file transfer rate of disk-stress from 1 MB/s to 10 MB/s. Our goal is to select the stress workloads' inputs so as to replicate the cloud's performance degradations seen in our experiments on Amazon EC2.

Figure 9 plots both the estimated and client-reported latency degradations for Data Serving and Web Search, and task completion time degradations for Data Analytics, reported by the interference-suffering VM. Each group of bars represents a different amount of interference, yielding performance degradation roughly from 5% to 50%. We observe that the analyzer's CPI analysis can faithfully approximate the degradation across the interference levels. In particular, we observe that the analyzer estimates the degradation within 10% accuracy in the worst case, and less than 5% on average.

5.4 How robust is DeepDive's placement?

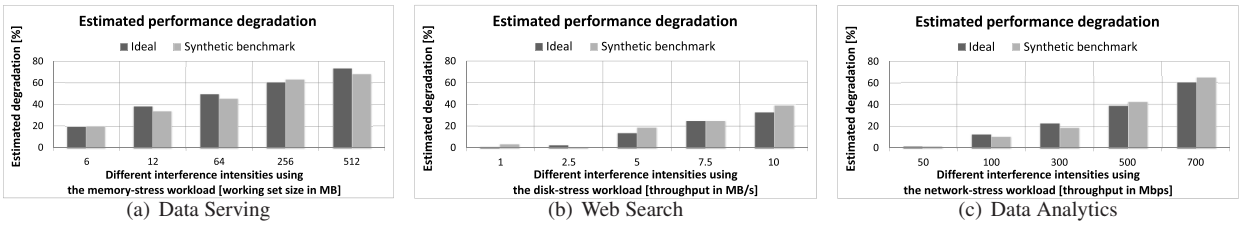
Here we evaluate the ability of the DeepDive's synthetic benchmark to mimic the low-level behavior of a VM in two cases of handling interference by VM migration: 1) the VM affected by interference, and 2) the most aggressive VM that is the culprit for interference.

First, we monitor the performance degradation that both the monitored VM and its synthetic representation experience when co-located with our stress test workloads. If they match, the synthetic benchmark can successfully be used to quickly test if a migrated VM would no longer suffer interference. To evaluate the clone's accuracy under different interference conditions, we leverage our three stress workloads to tune interference intensities as described in Section 5.3. Figures 10(a) to 10(c) contrast the performance degradation reported by the real VM and its synthetic representation, while the real VM runs different cloud applications. We see that the synthetic benchmark can closely approximate the performance degradation of a real VM – the median and average estimation error of our synthetic benchmark across all our experiments were 8% and 10%, respectively. These results can be further improved, especially if representative interference conditions are considered during the training phase of the synthetic benchmark.

Next, we show how the placement manager migrates an aggressive VM to an appropriate destination PM so as to minimize the resulting interference. In response to detecting an interference-inducing VM (memory-stress workload), DeepDive runs the synthetic representation of this aggressive VM on three PM candidates, each of which is running one of our cloud workloads. Based on these runs, the placement manager selects the destination PM on which the analyzer reports the least interference. Figure 11 plots the resulting performance degradation at that PM relative to the best (but impractical) scenario where the placement manager learns the interference effects on the destination PM by actually performing VM migration. During the experiment, we also record the resulting performance degradation for all the possible placements, allowing us to: i) compute the average performance degradation, and ii) label the placement with the highest performance degradation as the worst. We observe from the figure that DeepDive finds the best destination PM relying on its synthetic benchmark to estimate the interference. This result is important, because it shows that we can entirely eliminate expensive and yet worthless (for placement) VM migration that could cause performance degradation elsewhere.

5.5 What is the overhead of DeepDive?

DeepDive imposes a small per-VM memory overhead. For example, even when a VM is experiencing interference every hour, DeepDive requires less than 5KB to



(a) Data Serving (b) Web Search (c) Data Analytics

Figure 10: Synthetic benchmark's accuracy.

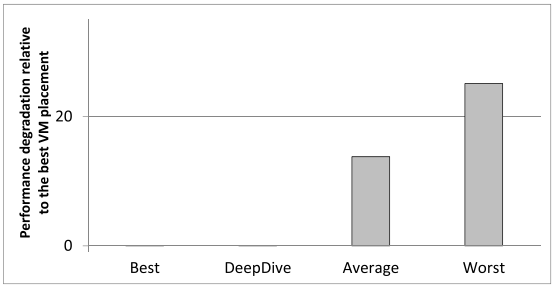


Figure 11: The placement manager properly predicts interference on the possible destination PMs.

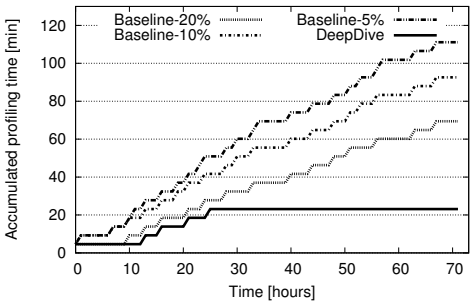


Figure 12: DeepDive's profiling overhead is low, and diminishes as it learns more about the VM behaviors.

record the VM's behavior for the whole day. Storing this information into a repository is not an issue, as there are many works on high-performance NoSQL datastores.

We next explore DeepDive's profiling overhead, i.e. the amount of time and the number of machines required by the interference analyzer. We have conducted our evaluation using both live experiments with the Data Serving workload (it invokes the analyzer most frequently) and simulations. Running live experiments in our testbed helps us understand how often DeepDive triggers the analyzer in dynamic, realistic environments, and gives us an idea of the overall profiling overhead. Using this information, we drive simulations to analyze the scaling properties of DeepDive when applied to large-scale datacenters with high VM-arrival rates.

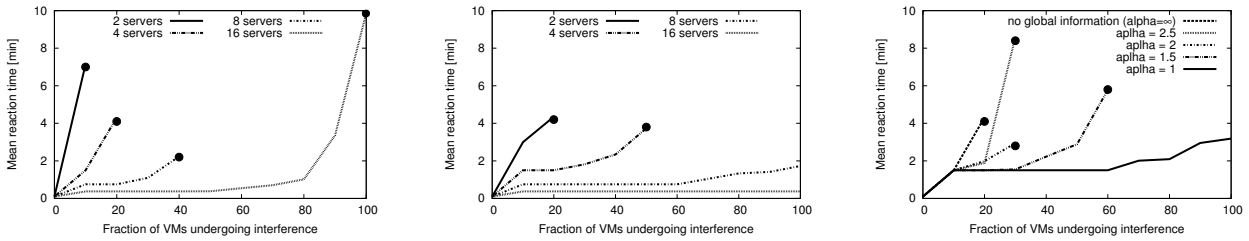
Using real experiments, Figure 12 plots the accumulated profiling time for a VM undergoing interference for both DeepDive and a baseline approach. The baseline triggers the analyzer every time performance varies more than a threshold (5%, 10%, and 20%). Triggering the analyzer too frequently renders the baseline unscalable and infeasible in practice. On the other hand, DeepDive relies on its warning system and its observed VM behaviors to prevent unnecessary VM profiling. The figure shows that DeepDive's overhead accumulates to only twenty minutes of profiling over 3 days. In fact, after the first day, no more profiling is needed.

To extrapolate from these results, we next drive our simulator to trigger the analyzer exactly at the points in time that were previously recorded by our live experiment. We also used Matlab to model DeepDive's profiler as a simple queue: i) the VM arrival rate follows a Poisson process (we also experiment with a lognormal distribution of VM arrivals below), ii) the service time is replicated from the live experiments, and ii) the datacenter handles 1000 new (incoming) VMs every day.

Figure 13(a) presents DeepDive's reaction time as a function of the percentage of VMs undergoing interference. The figure plots the reaction time as long as the system is stable (mean service time < mean inter-arrival time), and the waiting time is acceptable (less than 10 minutes). As expected, the mean reaction time decreases as DeepDive uses more profiling servers. Most importantly, the figure demonstrates a desirable scaling behavior. For instance, only four profiling servers provide reaction time within four minutes, even under an aggressive rate of 20% of VMs undergoing interference.

These results assume that each VM runs a different workload, thus preventing DeepDive from being able to leverage global information. We design another set of experiments where VM reoccurrence follows a typical Zipf distribution – a few cloud tenants execute their workloads on a large number of VMs (available global information), and the remaining tenants run their deployments on a handful of VMs ("the long tail"). Figure 13(b) shows that leveraging global information significantly improves DeepDive's reaction time and allows it to further reduce the number of profiling servers required (by a factor of two in these experiments).

To mimic various deployment scenarios, we vary the power-law tail index (from light- to heavy-tailed, using the α parameter) while using four profiling servers. Figure 13(c) plots the mean reaction time as a function of interference. While leveraging global information is most effective under the "light tail" conditions (α=1), it sub-



(a) Using only local information. (b) Using both local and global information. (c) Under a range of VM popularities [6].

Figure 13: Reaction time for 1000 new VMs per day. Curves stop where the system becomes unstable or excessively slow.

stantially improves DeepDive's reaction time for all the scenarios we considered.

To demonstrate DeepDive's scaling under more bursty workload behaviors, we repeat the same set of experiments under a lognormal VM-arrival distribution, again assuming 1000 new VMs per day. Figure 14 shows that fewer than 10 dedicated profiling machines are required, even under this extreme new-VM arrival scenario.

6 Related Work

Most of the existing efforts on predicting interference either focus on on-chip contention or target private clouds. This section summarizes the previous work and points out differences with respect to our approach.

Interference analysis. Recent efforts [14, 19, 26, 27, 35] demonstrate that an analysis of the sensitivity of workloads to co-located applications may accurately predict the degradation due to interference. In public clouds however, applications are not available prior to their deployment and cloud providers cannot easily perform this analysis. Thus, DeepDive does not rely on prior knowledge of applications or their interactions.

Mars *et al.* [27] proposed Bubble-Up, a methodology for predicting degradation originated by co-locating applications that share memory subsystem resources. The approach uses the "bubble", a carefully designed stress-test that runs in parallel with the application to measure its sensitivity to sharing of the memory subsystem. The performance degradation is estimated by indexing the corresponding sensitivity curve with the co-located application's pressure score.

Our synthetic benchmark also places artificial pressure on the system to estimate the resulting performance. However, our approach differs in that it tries to produce the exact pressure that a VM would generate, rather than creating a sensitivity curve. Furthermore, DeepDive focuses on all key shared resources (including CPU and I/O), rather than just the memory subsystem.

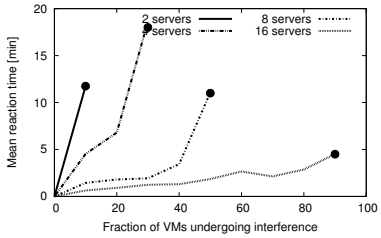
Sandboxing. Running experiments in an isolated environment has been proposed before. For example, Zheng *et al.* [34] proposed running isolated VM experiments to obtain preferable resource allocations for different workloads, whereas DejaVu [33] first performed isolated VM experiments to detect interference with the support of applications. DejaVu attempted to eliminate interference by overprovisioning the involved VMs. In contrast, DeepDive does not require any application assistance, performs experiments in isolation only when it detects reliable signs of interference-caused degradation, and manages interference by intelligently migrating applications.

Datacenter profiling and workload characterization. Sample-based profiling tools, like Magpie [24] and Pinpoint [15], produce workload models and automatically manage failures in distributed systems. Although these tools are useful for understanding workload behaviors or miss-behaviors, they are not particularly useful in virtualized environments where cloud providers do not have access to applications running inside VMs. Relative to this approach, DeepDive can transparently pinpoint the principal source of interference between VMs, and migrate VMs to reduce or even eliminate interference.

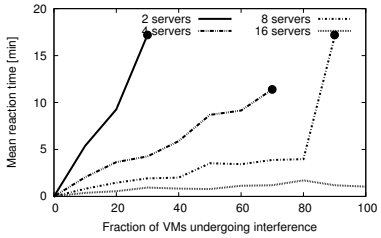
Synthetic benchmarks. Given their easy development, synthetic benchmarks are often used to mimic behaviors of a specific application on different hardware platforms. Even more convenient, tunable benchmarks can closely approximate a large portion of an arbitrary application's behavior by merely determining a suitable set of input parameters [32]. Several recent efforts [23, 29, 30, 31] have also demonstrated that one can reproduce any application's behavior using a limited number of the application's characteristics, like the memory access pattern, instruction dependencies, etc. These previous efforts inspired the design of our synthetic VM benchmark. Importantly, we are the first to use such a benchmark to manage interference.

Performance modeling. Recent efforts have tried to predict performance by relying on regression models. For example, Lee *et al.* [25] combine processor, contention, and penalty models to estimate performance in multiprocessors. Similarly, Deng *et al.* [18] rely on hardware performance counters to model the performance (and power consumption) of the memory subsystem.

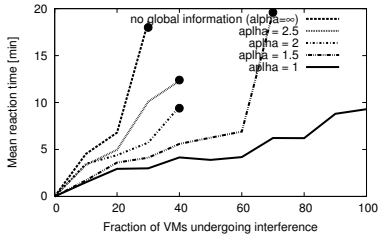
These works are orthogonal to DeepDive, since it does not try to predict performance per se, but rather to pinpoint the resource that is causing the interference. Fur-



(a) Using only local information. Under aggressive interference rate of 20%, DeepDive needs only 4 profiling servers to react to every warning signal in about 4 minutes.



(b) Leveraging both local and global information. DeepDive's reaction time is substantially improved (cut in half) when global information is used.



(c) Leveraging both local and global information allows DeepDive to substantially improve reaction time under a wide range of different VM popularities [6]. Here, DeepDive operates with four profiling servers.

Figure 14: Reaction time for accommodating an arrival rate of 1000 VMs per day under burstier VM-arrival distributions such as lognormal. Curves stop where the system becomes unstable or excessively slow.

thermore, our framework is not tied to a specific architecture, and focuses on all key shared system resources.

7 Conclusion

Cloud services are becoming increasingly popular. A key challenge that cloud service providers face is how to identify and eliminate performance interference between VMs running on the same PM. This paper proposed and evaluated DeepDive, a system for transparently and efficiently identifying and managing interference. DeepDive quickly identifies that a VM may be suffering interference by monitoring and clustering low-level metrics, e.g. hardware performance counters. If interference is suspected, DeepDive compares the metrics produced by the VM running in production and in isolation. If interference is confirmed, DeepDive starts a low-overhead search for a PM to which the VM can be migrated.

Our experimental results with real systems and traces show that the DeepDive metrics cluster nicely into two groups representing the presence or the absence of interference. The results also demonstrate that the frequency of production-vs-isolation comparisons decreases quickly (e.g., to negligible levels by the second day of the trace), as DeepDive's clustering becomes more accurate. Moreover, we find that the metrics can indeed be used to identify the exact cause of the interference. Finally, the results demonstrate that the benchmark is a good representative for the corresponding VM, enabling a low-overhead selection of VM destinations.

References

[1] Apache mahout. `http://mahout.apache.org`.

[2] Apache nutch. `http://nutch.apache.org`.

[3] Cisco Global Cloud Index: Forecast and Methodology, 2010-2015. `http://goo.gl/1LtqP`.

[4] Faban framework. `http://java.net/projects/faban`.

[5] HDFS. `http://hadoop.apache.org/core`.

[6] Pareto distribution. `http://en.wikipedia.org/wiki/Pareto_distribution`.

[7] What are the barriers to cloud computing. `http://www.interxion.com/cloud-insight/`.

[8] A. R. Alameldeen et al. Ipc considered harmful for multiprocessor workloads. *IEEE Micro*, 2006.

[9] Apache Foundation. The Apache Cassandra Project. `http://cassandra.apache.org/`.

[10] R. Azimi, et al. Online Performance Analysis by Statistical Sampling of Microprocessor Performance Counters. In *ICS*, 2005.

[11] S. Basu, et al. Active semi-supervision for pairwise constrained clustering. In *SDM*, 2004.

[12] A. Baumann, et al. The multikernel: a new os architecture for scalable multicore systems. In *SOSP*, 2009.

[13] M. Bilenko, et al. Integrating constraints and metric learning in semi-supervised clustering. In *ICML*, 2004.

[14] D. Chandra, et al. Predicting inter-thread cache contention on a chip multi-processor architecture. In *HPCA*, 2005.

[15] M. Y. Chen, et al. Pinpoint: Problem determination in large, dynamic internet services. In *DSN*, 2002.

[16] B. F. Cooper, et al. Benchmarking cloud serving systems with YCSB. In *SoCC*, 2010.

[17] J. Dejun, et al. EC2 performance analysis for resource provisioning of service-oriented applications. In *NFPSLAM-SOC*, 2009.

[18] Q. Deng, et al. Memscale: active low-power modes for main memory. In *ASPLOS*, 2011.

[19] M. Dobrescu, et al. Toward predictable performance in software packet-processing platforms. In *NSDI*, 2012.

[20] L. Eeckhout. *Computer Architecture Performance Evaluation Methods*. 2010.

[21] M. Ferdman, et al. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. In *ASPLOS*, 2012.

[22] M. Hall, et al. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009.

[23] A. Joshi, et al. The return of synthetic benchmarks. In *SPEC Benchmark Workshop*, 2008.

[24] T. Kielmann, et al. Magpie: Mpi's collective communication operations for clustered wide area systems. *SIGPLAN Not.*, 1999.

[25] B. C. Lee, et al. Cpr: Composable performance regression for scalable multiprocessor models. In *MICRO*, 2008.

[26] J. Machina et al. Predicting cache needs and cache sensitivity for applications in cloud computing on cmp servers with configurable caches. In *IPDPS*, 2009.

[27] J. Mars, et al. Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations. *IEEE Micro*, 2012.

[28] R. Nathuji, et al. Q-clouds: managing performance interference effects for qos-aware clouds. In *EuroSys*, 2010.

[29] A. Phansalkar, et al. Measuring program similarity: Experiments with spec cpu benchmark suites. In *ISPASS*, 2005.

[30] T. Sherwood, et al. Automatically characterizing large scale program behavior. In *ASPLOS*, 2002.

[31] K. Skadron, et al. Challenges in computer architecture evaluation. *Computer*, 2003.

[32] E. Strohmaier et al. Architecture independent performance characterization and benchmarking for scientific applications. In *MASCOTS*, 2004.

[33] N. Vasić, et al. DejaVu: Accelerating Resource Allocation in Virtualized Environments . In *ASPLOS*, 2012.

[34] W. Zheng, et al. Justrunit: Experiment-based management of virtualized data centers. In *USENIX*, 2009.

[35] S. Zhuravlev, et al. Addressing shared resource contention in multicore processors via scheduling. In *ASPLOS*, 2010.

Appendix

A.1 Algorithms

A.1.1 The warning system

The warning system first tries to match the current set of metrics against the previously learned set of normal application behaviors. If it cannot find a match, it checks whether other VMs running the same code are experiencing similar behavior.

```
while a VM is running do
    read its current behavior (a set of metrics M);
    if M is within distance T from previous VM
    behaviors then
        the current behavior is normal;
        refrain from any action;
    else
        retrieve current behavior of other VMs
        executing the same code;
        if most of VMs are in the same region then
            the current behavior is normal;
            extend the set of inspected VM
            behaviors with M;
        else
            trigger the interference analyzer;
        end
    end
end
```

Algorithm 1: The warning system's algorithm.

A.1.2 The interference analyzer

The interference analyzer compares the statistics collected from the production VM and its clone running in a sandboxed environment.

```
for each VM state Mp provided by the warning
system do
    determine the VM state in isolation Mi;
    if Degradation(Mp,Mi) < threshold then
        the current behavior is normal;
        extend the set of inspected VM states with
        the new state Mi;
    else
        estimate performance degradation;
        compute the principal causes of interference;
        invoke the VM placement manager;
    end
end
```

Algorithm 2: The interference analyzer's algorithm.