2011 IEEE 4th International Conference on Cloud Computing

VirtPerf: A Performance Profiling Tool for Virtualized Environments

Prajakta Patil, Purushottam Kulkarni and Umesh Bellur
Department of Computer Science and Engineering
Indian Institute of Technology Bombay, India
{*prajaktap, puru, umesh*}@*cse.iitb.ac.in*

***Abstract*—Several applications in the "physical" world are being consolidated in "virtual" environments using different virtualization technologies. An important criteria for this exercise is to understand potential resource requirements and performance levels achieved in virtual environments. Empirical evidence of these can be gotten by benchmarking the application's performance in a controlled manner in virtual environments. These measurements can be used for a variety of purposes from virtual machine capacity planning to building sophisticated performance models to predict performance for loads that cannot be practically tested. In this paper, we present *VirtPerf* , an integrated workload generator and measurement tool to capture resource utilization levels and performance metrics of applications executing under controlled circumstances in virtualized environments. The tool aims to provide comprehensive measurement-based analysis for applications in different virtualization settings. Additionally, a configurable workload generator can be used to stress and profile applications under different load conditions. We present the detailed design of *VirtPerf* and a comprehensive empirical study to demonstrate its correctness and capabilities.**

***Keywords*-virtualization; application profiling; workload generation;**

I. Introduction

Traditionally, each web application has been deployed onto a dedicated set of production resources to ensure performance and enforce strict resource isolation. Risk management dictates over provisioning of resources to these applications, sometimes to the extent of doubling the resources over an anticipated peak in order to handle flash traffic. A proliferation in the numbers of these applications has led to *server sprawl* [1]—a situation in which there exist a large number of underutilized servers in a data center. An attractive solution, enabled by the advent of virtualization technologies, is to *virtualize*. Application servers instantiated in virtual machines can be consolidated and provisioned effectively over fewer physical machines (while retaining the isolation guarantees provided in a physical environment). Resources freed up by consolidation can either be allocated on-demand to any of the applications to meet unanticipated load or to deploy new applications. This multiplexing works well in situations where application peaks are out of phase with one another. However, to exploit benefits of virtualization-based provisioning, one still needs to capacity plan the virtualized environment where the range of resources that each virtual machine (VM) needs and corresponding performance levels are well understood.
The capacity planning exercise prior to a deployment helps answer questions such as:

- How much workload can a system support for a given amount of resources allocated to the VMs?
- What is the maximum achievable throughput and average response time of the system for a given provisioning of the VMs and what is the load in terms of concurrent users at which it is achieved? Does it satisfy SLA requirements ?
- Given a load level, what would be the resources needed to sustain it?
- What is the relation between VM resources and response time or throughput?

Profiling exercises stress applications with different workloads, monitor resource usage and performance levels and aggregate results to answer questions of the variety mentioned above. The profiles generated are vital for decisions regarding service level agreements (SLAs) offered to customers and the resources required for the same. Today's capacity planning tools suffer from two main disadvantages:

(i) They are not "virtualization aware". Traditional capacity planning tools are not built for virtualized environments. They are not aware of the flexibility in resource allocation and virtualization specific processes such as VM migration that these environments bring into play.
(ii) Lack of synchronization between system setup (VM provisioning), load generation and profiling. System setup is usually done manually and separate tools exists for load generation and profiling and one must correlate results across them.

The result is that one has to use virtual environment management tools and monitoring APIs, manually perform tasks of resource allocation and for each setting, coordinate load generation and profiling tools to cobble together a picture of how the application performs in a virtual environment. The task is cumbersome (to perform manually) and complicated because of the multiple possible system configurations to profile, particularly when the application has multiple tiers.

In this paper, we present *VirtPerf* — an automated capacity planning tool for virtual environments. *VirtPerf* integrates monitoring APIs of virtual environments, a load balancer with multiple control knobs (#clients, request rates etc.) and a resource configuration manager that can operate the system in different resource availability scenarios (both with and without VM migration). As a final step, *VirtPerf* presents back to the user a comprehensive performance and resource utilization picture of the application under test. Profiling results of *VirtPerf* can be used as is for capacity sizing virtual machines to run multi-tier applications OR can be used as an input to a performance model to play out multiple "'what-if"' scenarios. *VirtPerf* supports the Xen [2] and KVM [3] virtual environments and has been architected to be extensible to

This project is funded in part by a grant from Yahoo! Software Development India (Grant #09YAHOO001 at Indian Institute of Technology, Bombay)

other virtualization solutions.

II. A Survey of Existing Profiling Tools

There are many performance monitoring tools available for virtualized environments today. Xentop[4], XenMon[5] and XenoProf[6] are profiling tools for Xen. They are stateless in that they have to be called each time to get a snapshot of the system under test. They provide resource consumption data of all the VMs as well as of the hypervisor(Dom0 in Xen).

These tools are pure profilers. They monitor VMs when told to do so but lack the intelligence to do anything else. They have to be integrated with an existing load generator. Most of them just measure resource usage information and say nothing about performance trends. They cannot set up resource allocation limits on the VMs and hence can't automatically create multiple allocation scenarios. They are NOT aware of virtualization processes such as migration. Most of the tools are not cross-platform products.

[7] presents a framework for automated server benchmarking. They have concentrated on automation policies which are independent of underlying framework e.g server implementation, automated workload generator, resource allocations and virtualization technology. [8] presents the workload characterization of a busy WWW server (NCSA webserver) deployed on non virtualized high end HP server. They explain characteristics of the systems response, including the distribution of response sizes and server response times. But, they have not studied system resource utilization patterns as a function of workload and thus cannot examine saturation point for the webserver. [9] presents workload generation toolkit for virtualized applications, which considers three dimensions for workload generation - variation in amount of load, variation in mix of operations performed by clients and variation in popularity of data accessed. But, they have not studied workload generation under different virtualization aware scenarios such as multiple resource allocations to VMs, virtual machine migration. [10] talks about an open testbed architecture, where user can install and start web applications, configure workloads and instantiate controller that then coordinates experiment runs.

There is some primitive support for figuring out CPU charge back [5], [11] to individual VMs in Xen (as a result of the CPU overhead at Dom0 caused by network I/O). However there is little done by way of disk I/O analysis - indeed most tools fail to even capture all Disk I/O statistics.

VirtPerf is more than a pure profiler, it is a more complete capacity planning tool that can give the user the complete range of performance and resource usage statistics for different resource allocations in virtual environments. There are no other tools similar to *VirtPerf* that we have come across so far.

Autoperf [12] is an automated tool for resource profiling and capacity analysis of web-based systems deployed in physical environment. *VirtPerf* is an extension of *Autoperf* that understands the flexibility of resource allocation in virtual environments and features such as VM migration.

III. Tool Requirements

In the previous section we discussed the feature and drawbacks of existing profiling tools and motivated the need for a new tool to profile applications in virtual environments. A list of essential requirements of such a virtualization-aware profiling tool are:

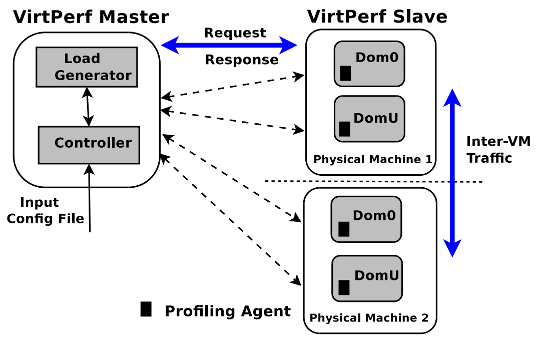


Figure 1. *VirtPerf* Architecture

- Support for multiple virtualization technologies such as Xen, KVM, VMWare etc.
- Tight synchronization of load generation and profiling to avoid errors and to control load parameters according to nature of application.
- Should be able to detect warm-up and stable periods of the application for correct profiling.
- Should support multi-tier applications and be able to profile aggregate and individual behavior of tiers.
- Should support real load scenarios, e.g., configure request-level think time generation, set limits on resource availability for VMs and map VMs to CPU cores.

While we assume a web-based application, i.e., a web-interface for request and responses, the design of *VirtPerf* makes it easily extensible to applications with other types of interfaces.

IV. *VirtPerf* Architecture and Features

VirtPerf employs a master-slave architecture and consists of three main modules, the Controller, the Load Generator and the Profile agents (as shown in Figure 1). The Controller and the Load generator together form the master while profilers are the slaves that run in the virtual environments where the application under test is deployed. The master part is implemented in JAVA and profiler daemons are implemented in C++.

- *Controller*: The Controller co-ordinates and synchronizes load generation, resource configuration on the virtual machines and monitoring of performance and resource parameters. The controller reads a configuration file and passes relevant information to the respective modules, i.e., transaction information to the load generator and node configuration information to the profiler. It directs the load generator to start generating load at a selected level (based on number of clients and think time distribution). The controller detects warm-up of the server based on the stability of response time of requests and signals the profiler agents to start monitoring the virtual machines. After completion of load generation (number of successful requests), the load generator collects statistics from both the load generator and the profiler agents, and computes client side performance metrics such as response time and throughput. These numbers drive the selection of the next load level. The controller also receives resource usage metrics from the profiler agents after the load generation of each round.

- *Load Generator*: The Load Generator generates requests at the specified load level—number of clients, number of requests per client and think time distribution parameters. We assume a closed loop system where each client issues a request, waits for a response from the server and samples a distribution to emulate think-time behavior before issuing the next request. Each client is emulated using a separate thread, which issues requests according to the above loop, till it has requested a maximum number of requests or throughput of the application on the VMs reaches saturation. Once load generation starts, the module first detects warm-up after which it informs the profiler agents to start profiling. The generator conveys start, warm-up detected and end states with regards to load to the controller for coordination with the profiler agents.
- *Profiling Agents*: The profiling agents are responsible for collecting server side statistics. With the Xen virtualization solution, a profile daemon runs in the privileged domain (Dom0) to collect aggregate resource statistics and additionally per-VM profile daemons collect statistics from within the VM. We term this assumption of being able to instantiate profiling daemons inside VMs to be profiled as the *gray box* approach (similar to [13]). A more practical approach is that of collecting statistics related to VMs from the host operating system or the virtualization control plan, a *black box* approach. The trade-off being certain measurement parameters may not be available or exposed at the host OS. We comment on this trade-off with regards Xen in Section VI. The *VirtPerf* controller (master) issues commands to the daemons in VMs (slaves) to start and stop capturing resource usage information, and to send back the profiled data.

A. Inputs to VirtPerf

VirtPerf reads in an input configuration file during startup to obtain load generator parameters, get locations of application tiers, learn about resource configuration ranges to be configured at the application tiers and get profiling characteristics (whether to perform migration during a profiling run etc.). The following input information is part of the input configuration of *VirtPerf* and is specified in an XML format:
(i) *Deployment and configuration information:* The location of the server is specified using the server's IP address, the port number on which it is receiving requests and server process identifier (process name). If the profiled server is running on an hosted virtualization solution, like Xen, separate location information (IP address) for the host OS (Dom0, in case of Xen) is also mentioned.
(ii) *Resource allocation information:* As part of resource configuration parameters, the user can specify the number of CPU cores to be allocated to the host and guest VMs, range of CPU capacity (and increment factors) for each VM. The application is profiled for different combinations of resource allocation within the range of CPU capacity settings.
Configuration information can include instructions of whether a VM should be migrated and if so at what instant in the profiling run. The migration start time is specified in terms of % of requests executed by the application server.
(iii) *Transaction information:* This information is related to the set of valid URLs to be used to load the application service. A user can specify a set of URLs to use for load generation or specify a URL sequence and range of values for each argument, to generate URLs dynamically.
(iv) *Load description:* The load to be stressed on the server is specified using, number of concurrent users, request execution count for each and a think-time probability distribution (e.g., Poisson, Uniform) along with required distribution parameters.
(v) *Migration Configuration:VirtPerf* is also able to profile the effect of migrating a VM while the service is under load (either the service VM OR an unrelated VM is migrated). Users can configure exactly when the migration is to happen, which VM is to be migrated and the direction of migration (towards or away from the application VM).

B. Reports and Analysis produced by VirtPerf

Based on the above input setting, *VirtPerf* performs a set of profiling experiments based on the range and increment factor of resource settings. Each run of *VirtPerf* is at one combination of resource settings and corresponds to incrementing the load level by increasing number of requesting clients by a fixed value. Additionally, in each run one of the VMs being monitored can be migrated or a 'interfering' VM can be migrated into our outside the multi-tier application setup.
For each profiling run, *VirtPerf* reports following resource usage information of all active domains in the virtual environment:

- *R*esource Usage Information: Percentage CPU utilization, Memory consumed by each VM, disk blocks read and written, network bytes received and transmitted, and average service time per transaction.
- *A*pplication behavior: Average response time per transaction and throughput in transactions per second.

In the case of Xen, resource utilization data is collected at the privileged domain using inbuilt utility *xentop*, and at other virtual domains (non-privileged) using system commands *ps, netstat, iostat*. In case of the KVM virtualization solution, resource usages are collected at both the domains (host and guest) using commands *ps, netstat, iostat*.
At the controller (in coordination with the load generator), *VirtPerf* also reports the maximum throughput (req/sec) of the system, the load level at which throughput saturates and the response time for this load level.

C. Key Features

AutoPerf, the predecessor of *VirtPerf* has the following features that we use: automatic detection of saturation load level, automated detection of server warm up, We now describe the important features of *VirtPerf* beyond what already exists in AutoPerf:

- *Profiling Modes*: *VirtPerf* can operate in 3 execution modes.
  (i) *Single load level:* This is the simplest execution mode. In this mode, load is generated at a specified load level specified in the input file. The server process is profiled at a fixed, single load level and resource usage and performance metrics are reported by the tool.
  (ii) *Multiple load levels:* In this mode, load is generated at incrementally higher levels (determined automatically) till it detects throughput saturation. This mode is helpful to see the change in service demand or resource usages with the changes in load level. It can help detect unanticipated behavior of a server process. We achieve different load levels by increasing

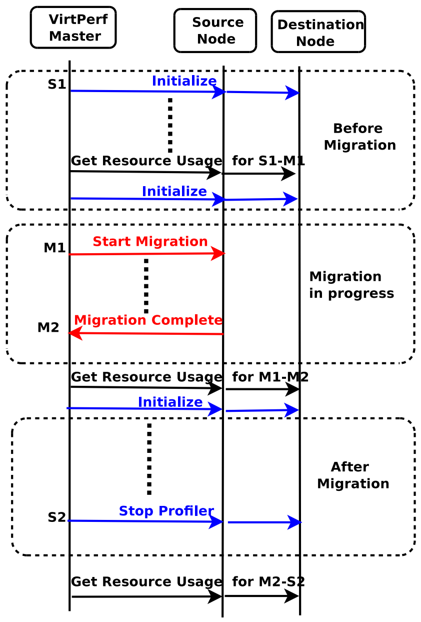


Figure 2. Profiling application with VM migration enabled

the number of concurrent clients accessing the server process. (iii) *Fixed Multiple load levels:* In this mode, the user can specify a fixed range for load, say 1-100 concurrent clients. *VirtPerf* profiles the server in steps of increasing load within the range. For each load level, it generates a constant number of requests. This execution count can also be specified in the input file, else *VirtPerf* determines the execution count of each thread automatically.
(iv) *Profiling with probabilistic access patterns :* In this mode, realistic scenarios are emulated by generating a workload that contains mix of transactions according to a customer behavior model graph (CBMG). A CBMG [14] summarizes the navigational patterns of a group of customers. A CBMG description formally consists of a set of states, a set of transitions between states, and an $n * n$ matrix, P = $[p_{i,j}]$, of transition probabilities between the n states. States represent steps in an end to end transaction and they vary according to web applications. *VirtPerf* generates load on the server in such a way that the mix of transactions generated resembles the workload distribution specified using a CBMG.

- *Think time specification*: Think time is the average time a client remains idle after a response has been received and before sending the next request. *VirtPerf* can be configured to use different distributions for think time samples between requests. Currently, *VirtPerf* supports Poisson, Uniform and Exponential distributions and has been architected so that additional distributions can be added with no change to the *VirtPerf* code. This is necessary to simulate likely user behavior.
- *Profiling with resource usage tuning :* The server on which the application VMs execute will most likely have multiple CPU cores. *VirtPerf* can be configured to setup CPU resource limitations per server virtual machine (including privileged domain, Dom0, with Xen). *VirtPerf* allows the specification of CPU percentage to be allocated per VM and also configured mapping of VMs to specific CPU cores. Further, a range for each can specified, e.g., vary CPU allocated to a VM from 10–100% and assign it 2–4 cores. *VirtPerf* will request the server slave profiler daemons to configure the VMs for each combination, iteratively. For each combination, the server is stressed for profiling.
- *Profiling with VM migration:* Migration of virtual machines is an important benefit to leverage with VM-based provisioning. Since live migration of VMs needs significant resources for the migration state transfer and can cause disruption in application performance, *VirtPerf* supports profiling the behavior of an application during migration. *VirtPerf* can be configured to instruct server VMs to migrate at any point during execution of the profiling run. Further, either the server being profiled can be migrated, or another tier of the same application can be migrated to the the same machine as one of it's other tiers or an "interfering' VM can migrate out or into the same machine as the profiled server. *VirtPerf* facilitates specification of all these parameters—the VM to migrate, when to migrate and destination of migration. Further, it keeps track of migration start time and the interval to generate resource usage and performance statistics for the intervals before migration, during migration and after migration.
It's control protocol (shown in Figure 2) for these scenarios takes cognizance of the migration process and profiles accordingly.

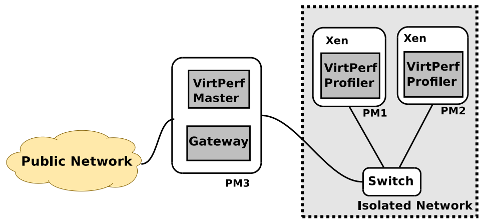
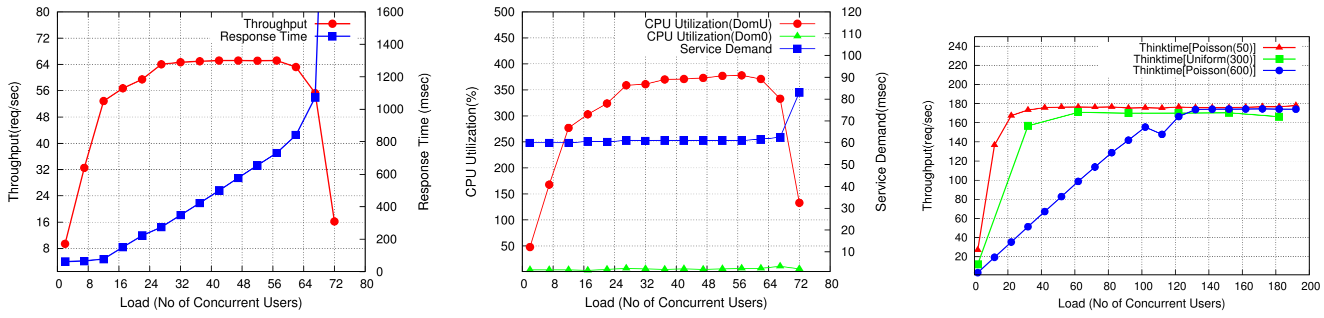


Figure 3. Setup used for experimental evaluation.

V. Performance Measurements Using *VirtPerf*

A. Experimentation Setup

The setup for experiments included physical machines with the following characteristics—Quad core Intel i5 CPU @ 2.80 GHz, 3 GB main memory and Gigabit Ethernet. Two such machines are used to host multiple tiers of the application under test and a third machine acts as the *VirtPerf* master. Note that the master does not need as much resources as do the machines hosting the application since it's primarily a controller. The Xen-enabled host OS and guest OSes use the 2.6.32 64-bit Linux kernel. Each virtual machines if configured with 512 MB and be allocated 1 to 4 cores based on *VirtPerf* input configuration. Figure 3 shows the experimental setup. The three physical machines PM1, PM2 and PM3 form an isolated network. Since we wanted to make measurements without interference resulting from the network traffic of other applications. In that sense, this represents a clean room environment for profiling. Machine PM3 has two network interfaces, and acts as gateway between the evaluation setup and the department network.



(a) Throughput and Response time variation (b) CPU utilization and Service demand variation (c) Sensitivity to think-time distribution

Figure 4. Throughput and Response time correlations with load.

B. *Workloads*

We used two applications - a Web-Calendar [15] and RUBiS [16], to generate workload for evaluation of *VirtPerf* . WebCalendar is a PHP-based multi-tier calendering application used for pinning events, reminders etc. It uses *MySQL* as the back end database for storing information. Load is generated in terms of concurrent users accessing the application for viewing or creating calendar entries. RUBiS is a benchmark prototype to model an auction site. It implements the core functionality of an auction site - selling, browsing and bidding. For a visitor session, users need not register but are only allowed to browse. Buyer and seller sessions need registration for transactions. RUBiS also uses *MySQL* as the back end database. For emulating real time scenario, database tables are populated with large number of dummy entries - users (1000000 users), items (130000 items), categories (50 categories) and bidding entries (1200000 bids), totaling to about 24,00,000 records in simulation database.

C. VirtPerf *Validation*

We have validated the results reported by *VirtPerf* by setting up web services that take up a known amount of resources - Network, Disk and CPU and/or have a known response time. We have observed that the results reported by *VirtPerf* (both from Dom0 and from within the VM where this benchmark service executes) are consistent with the expected/known values.

VI. Profiling with *VirtPerf*

A. *Impact of Load on Performance and Resource Usage*

Figure 4 shows the results of profiling a Web-calendar service stressed using different load levels. The load generated used a Poisson think time distribution with mean of 150 milliseconds. Both tiers of the web-calendar application are hosted on the same physical machine.

Additionally, the *VirtPerf* feature of specifying multiple load levels within a range is used to auto-increment load levels for profiling performance metrics and resource usages.

Figure 4a shows the variation in the throughput and user-perceived response time as the number of users increases from 1 to 72 users. As seen from the graph, the throughput increases and stabilizes at 64 requests/sec and plummets after load increases beyond 62 concurrent users. Mimicking the throughput trend, the response time for each user also increases with load, from less than 100 ms at very low load to 1 second at a load of 67 concurrent users, and beyond which response time increases drastically due to non-linearity. The different CPU utilizations and service demand per request is depicted in Figure 4b. As can be seen, the web-calendar service is CPU bound and not network bound—Dom0 CPU utilization, to service network traffic of the user domains, is very low (5 %). On the other hand, DomU CPU utilization (utilization of the guest domain) increases with load and saturates at about 375 % (the guest virtual machines is configured to use 4 cores). Beyond a load of 62 users, as seen with throughput and response time behavior, the system goes into overload and the CPU utilization of the server plummets. The load generated, uses a set of similar URL and hence the service demand reported is stable at 60 ms until a load of 67 concurrent users, beyond which it starts increasing as well.

Thus, as shown above, different load level parameters like, think-time distributions and their parameters, load types (browsing-mix, ordering-mix etc.) can be applied and profiled with *VirtPerf* for detailed performance and resource usage analysis.

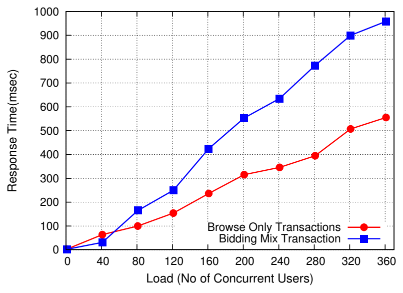
B. *Sensitivity to Think-time Distribution Parameter*

Next, we vary the think-time distribution and its mean value to study the impact on throughput of the RUBiS application. Figure 4c plots throughput variation with think time distribution following Poisson and Uniform distributions. For think-time sampling based on a Poisson distribution with mean 50 ms and a Uniform distribution with mean 300 ms, the throughput saturates at 180 requests/s with close to 36 concurrent users. While with a poisson distributed thinktime with mean 600 ms, as thinktime $>>$ service demand, arrival rate is very low. So, throughput curve goes up gradually and reaches saturation level with 130 concurrent users.

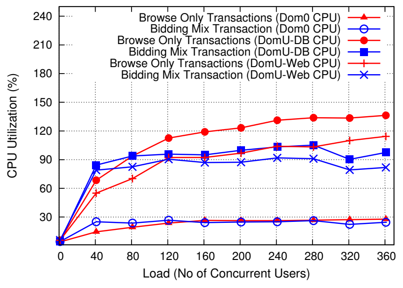
Varying the think-time distribution type and its parameters *Virt-Perf* can capture different limits of load before saturation.

C. *Profiling with Probabilistic Mix of URLs*

In the real world, a typical client browses through different types of URLs of a complex web application e.g browse a catalog, select items, buy an item, put comment etc. As a result, it is beneficial to profile the applications emulating scenarios which resembles the realistic and representative workload patterns. *VirtPerf* can profile the web applications with a probabilistic mix of URLs defined by a CBMG. To demonstrate this feature, we use the multi-tier RUBiS



(a) Application Response Time



(b) CPU Utilization

Figure 5. Profiling with Probabilistic Access Patterns

application with both tiers hosted on the same physical machine but on different VMs. A CBMG is defined for two types of customer behaviors - one that contains only browsing transactions (read-only interactions, labelled *Browse Only Transaction*) and a bidding mix (15% write interactions, labelled *Bidding Mix Transactions*).

Figures 5a and 5b show response time and CPU utilization variation with load (#concurrent users) for two types of customer behaviors. As mentioned earlier, *Browse Only Transaction* have only database read operations, and hence average response time of requests is less than that with *Bidding Mix Transactions*. As can be seen from the graph,as load changes from 1 user to 370 concurrent users the response time increases from 4 msecs to 550 msecs with *Browse Only Transaction*. The corresponding rate of increase in response time is higher, increasing from 3 msecs to 950 msecs per request with *Bidding Mix Transactions*. As more time is spent for I/O completion during write operations (when CPU is unused for that request) as compared to during read only transactions, CPU utilization of both the DomUs (web tier and database tier) and Dom0 for *Bidding Mix Transactions* is slightly less than *Browse Only Transaction* (as shown in Figure 5(b)). For *Bidding Mix Transactions* CPU utilization stabilizes close to 95% for the Database tier and 85% for the Web tier from a load of 40 concurrent users onwards. While for *Browse Only Transaction* CPU utilization saturates at around 135% for the Database tier and 90% for the Web tier at a load of 240 concurrent users. For *Browse Only Transaction*, throughput increases and stabilizes at around 1050 req/sec at a load of 120 concurrent users, while for *Bidding Mix Transactions* throughput increases up to 1000 req/sec at a load of 40 concurrent users and then drops down if load increases beyond 40 concurrent users.

D. Profiling with Resource Usage Tuning

In an virtual environment, with VMs being multiplexed on shared platforms, it is beneficial to know the behavior of applications under different resource availability scenarios. Currently, *VirtPerf* can control allocation to the CPU resource, in terms of number of CPU cores and maximum allowed CPU utilization.

To demonstrate capability of *VirtPerf* to tune the CPU resource allocated to a VM, we use the multi-tier RUBiS application with both tiers hosted on the same physical machine. Load is generated with 50 concurrent users with a Poisson distribution with mean 150 ms for think time sampling. The CPU allocated to Dom0 (the privileged management domain of Xen) is configured to use 400% of the CPU, while the allocated CPU fraction for each of the two guest VMs (one for each tier—the web tier and the database tier) are varied. These specifications are part of the input configuration of *VirtPerf* .

Web tier CPU	**Database tier** CPU	**Response Time** (milliseconds)	**Throughput** (request/sec)
10%	20%	5902	8.43
10%	100%	1088	45.49
10%	180%	585	79.89
10%	260%	540	85.43
10%	340%	507	90.16
30%	20%	5525	9.07
30%	100%	1058	45.72
30%	180%	555	83.55
30%	260%	366	121.90
30%	340%	278	155.21

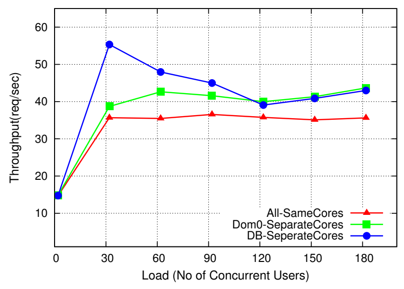
Table I
IMPACT OF CPU RESOURCE ALLOCATION ON PERFORMANCE METRICS.

Table I depicts variation of CPU resource allocated in different amounts the the Web and the DB tiers respectively. The privileged domain (Dom0) is configured to use all 4 cores. As can be seen from the table, with increase in CPU allocation the DB tier from 20% to 340 % the response time decreases ten-fold—from 5901 ms to 507 ms, with 10% CPU allocated to the Web tier. The corresponding decrease in response time is a factor of 20 with 30% CPU to the Web tier. This significant increase hints at a dependence of performance on both a minimum CPU availability at the Web tier and as high as possible at the DB tier. The throughput shows similar trends, an improvement by a factor of 11 and 17 respectively, when CPU allocated to DB is varied from 20% to 340% for the two cases of 10% and 30% CPU allocated to the Web tier.

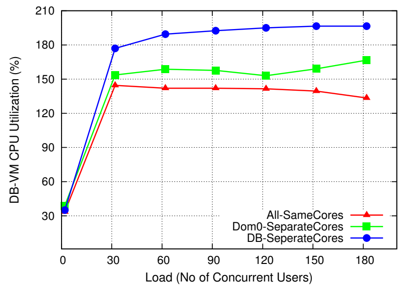
Any such number of deployments and variations in resource allocations can be studied with *VirtPerf* to understand system behavior under different resource constraints.

E. Effect of Pinning VMs to Cores

VirtPerf has the capability to pin down VMs to specific cores of a CPU. The decision to do this can have a significant effect on application performance as indicated in Figure 6a. In the case where 2 cores are shared between Dom0 and the two application VMs (containing the web and DB tier respectively), the throughput peaks at around 35 requests/sec whereas if we use the 2 cores in such a way so as to separate the DB tier onto it's own core and allocate the other core to be shared amongst Dom0 and the web tier, the

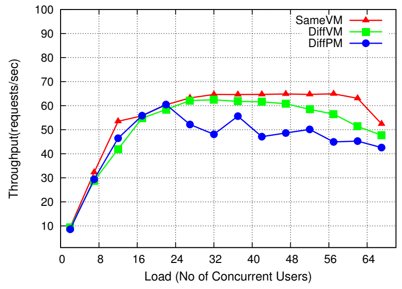


(a) Application Throughput

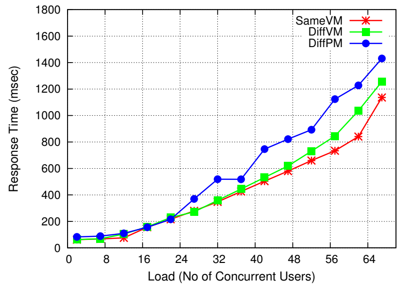


(b) Database tier CPU Utilization

Figure 6. Effect of pinning VMs to CPU cores on Throughput and CPU Utilization



(a) Application Throughput



(b) Response Time

Figure 7. Effect of deployment scenarios on performance metrics.

throughput peak jumps to around 55 requests/sec. The throughput saturation is driven by the DB tier being the bottleneck as CPU utilization numbers show in Figure 6b.

F. Impact of Deployment Scenarios

Functionality of *VirtPerf* is not dependent or restricted to any specific set of deployment scenarios. Since, tiers are assumed to the IP-addressable, they can be deployed in any manner (located on any physical machines) and appropriately specified as part of *VirtPerf* input configuration. This feature can be exploited to study behaviour of applications under different deployment scenarios. To demonstrate this feature, the web-calendar application with tiers is instantiated in different scenarios to study application level throughout and response time.

Figure 7b and 7a show response time and throughput variation with load (#concurrent users) for both processes of the application tiers in the same virtual machine (labelled $SameVM$), tiers hosted in different VMs but on the same physical machine (labelled $DiffVM$) and tiers in different VMs and located on different PMs (labelled $DiffPM$). As can be seen, over all load levels the $SameVM$ setup, as expected, has the highest throughput and lowest response time. The maximum throughput in the $SameVM$ setup is 65 req/sec, while that in the $DiffVM$ scenario is slightly lower (61 req/sec) and that in the $DiffPM$ scenario is fluctuating around 50 req/sec. Also, throughput starts decreasing (due to saturation) at much higher loads—at 60 req/s with $SameVM$, 40 req/sec with $DiffVM$ and as early as 24 req/sec with the $DiffPM$ scenario. Similarly, as compared to the $DiffPM$ scenario, the $DiffVM$ and $SameVM$ scenarios, beyond a load of 24 req/sec, have lower response times.

Thus, throughput and response time behaviour, as expected, of $DiffPM$ is the worst, due to usage of the physical network interface for data transmissions. *VirtPerf* can be used to quantify differences in these behaviors.

G. Profiling with Virtual Machine Migration

As discussed in Section IV-C, *VirtPerf* can be configured to study the impact of migration on resource usage and performance metrics. *VirtPerf* profiling agents can be instructed to migrate a VM from a source machine to a destination machine at a specified interval. As part of this experiment, we use the two-tier RUBiS application and study impact of migrating one of the tiers and another interfering VM on the application. For each experiment, the RUBiS tiers are instantiated as two separate VMs on the same physical machine.

Table II reports profiling statistics when the database tier is migrated, profiling commences at time $S1$, migration is initiated at $M1$, which terminates and time $M2$ and the profiling experiment terminates at time $S2$. Interval $S1 - M1$ is the before-migration phase, $M1 - M2$ is during migration and $M2 - S2$ the after-migration phase. As can be seen, during migration, both the source and destination Dom0's require increased CPU bandwidth—18% and 29% respectively, whereas before migration the "empty" destination Dom0 utilization was very low. The throughput during migration drops significantly from 157 req/s to 21 req/s (a factor of 7 decrease) and correspondingly the response time increases from 169 ms to 397 ms (a factor of 2.3 increase). Performance and resource usages after migration return to before-migration levels, except that the Dom0 VM on the destination machine has CPU utilization (of 5%) related to network activity of the database tier. Next, we instantiate an "empty" VM and collocated with the two RUBiS VMs (each containing a RUBiS tier) and study impact of migrating the "empty" VM. As seen in Table III, even if the "empty" is related to the RUBiS application, migrating it causes

	% CPU Utilization				Performance	
Phase	Source Dom0	Dest. Dom0	Web tier	Database tier	Response Time (ms)	Tput (req/s)
S1-M1	10	0	25	330	169	157
M1-M2	18	29	11	171	397	21
M2-S2	11	5	23	344	147	172

Table II
PROFILING RESULTS WITH MIGRATION OF DATABASE TIER VM.

	% CPU Utilization				Performance	
Phase	Source Dom0	Dest. Dom0	Web tier	Database tier	Response Time (ms)	Tput (req/s)
S1-M1	11	0	27	337	164	158
M1-M2	19	25	19	261	223	31
M2-S2	11	0	27	337	161	163

Table III
PROFILING RESULTS WITH MIGRATION OF A NON-APPLICATION VM.

source Dom0 CPU utilization to increase during migration. This results in lower CPU availability for the RUBiS application (DB tier's CPU utilization reduces from 337% to 261%). Correspondingly, the response time and throughput of the application are affected—response time increases from 164 ms to 223 ms and throughput decreases from 158 req/s to 31 req/s.

As can be seen, *VirtPerf* setup can be used to emulate several migration scenarios (including multiple migrations) during a single profiling run. This feature of *VirtPerf* can provide valuable information regarding application performance metrics and resource usages, in the face various migration scenarios. An immediate extension would be to restrict resource usages at source and destination machines for migration and study impact on the application and migration time. For the results presented here, all VMs could execute on all 4 cores of the physical machine and in both cases migration time was 425 ms.

VII. CONCLUSIONS

In summary, we have described an integrated capacity planning tool for virtual environments. *VirtPerf* integrates resource provisioning, load generation and profiling while at the same time allowing the user to understand the effects of migration on application performance. A single specification drives the entire suite of tests for different provisioning setups, a range of load levels and migration of different tiers. *VirtPerf* even allows the user to migrate a VM not related to the application either into or out of the application machine while under load. With this comprehensive profiling capability, one can determine precisely the VM resource requirements to operate at a specific SLA in a virtualized environment. We have demonstrated the usefulness of such a tool with a few applications (RUBiS being an enterprise benchmark) in a lab environment thus far.

Future work includes stressing the tool with larger applications involving multiple tiers, testing the tool with KVM (no new development is needed apart from using the monitoring API provided by KVM as opposed to Xentop) and adding a host of other useful features. Other types of resources such as network bandwidth, memory can be added as a part of resource tuning feature. We are planning to add a support for ZipF and other distributions that more closely approximate web think times. The last category of feature adds includes analytics to detect bottleneck identification in multi-tier applications.

REFERENCES

[1] Michael Armbrust et. al., "Above the Clouds: A Berkeley View of Cloud Computing," UC Berkeley, Tech. Rep. 4, 2009.

[2] Paul Barham et. al., "Xen and the Art of Virtualization," in *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, 2003.

[3] Kernel Based Virtual Machine (KVM), http://www.linux-kvm.org/page/Main_Page.

[4] Xentop, http://www.xen.org.

[5] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat, "Enforcing Performance Isolation across Virtual Machines in Xen," in *Middleware*, 2006.

[6] Aravind Menon et. al., "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," in *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*, 2005.

[7] Piyush Shivam et. al., "Cutting Corners: Workbench Automation for Server Benchmarking," in *USENIX Annual Technical Conference*, 2008.

[8] John A. Dilley, "Web Server Workload Characterization," Hewlett Packard Laboratories, Tech. Rep. HPL-96-160, 1996.

[9] Aaron Beitch et.al., "Rain: A Workload Generation Toolkit for Cloud Computing Applications," UC Berkeley, Tech. Rep. EECS-2010-14, 2010.

[10] Aydan Yumerefendi et. al., "Towards an Autonomic Computing Testbed," in *Proceedings of the Second Workshop on Hot Topics in Autonomic Computing*, 2007.

[11] L. Cherkasova and R. Gardner, "Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, 2005.

[12] S. Shrirang, "AutoPerf: An Automated Load Generator and Performance Measurement Tool for Multi-tier Software Systems," Master's thesis, IIT Bombay, Mumbai, India, June 2007.

[13] T. Wood, P. Shenoy, A. Venkataramani, and M. Yousif, "Sandpiper: Black-box and Gray-box Resource Management for Virtual Machines," in *Proceedings of 4th USENIX Symposium on Networked Systems Design & Implementation*, 2009.

[14] D. A. Menascé and V. A. F. Almeida, *Scaling for E-Business: Technologies, Models, Performance, and Capacity Planning*. Prenctice Hall, 2000.

[15] WebCalendar, http://www.k5n.us/webcalendar.php.

[16] Cristiana Amza et. al., "Specification and implementation of dynamic Web site benchmarks," in *Proceedings of IEEE International Workshop on Workload Characterization, WWC-5*, 2002.