Untangling Mixed Information to Calibrate Resource Utilization in Virtual Machines

Lei Lu
College of William and Mary
Williamsburg, Virginia, U.S.A
llu@cs.wm.edu

Hui Zhang
NEC Labs America
Princeton, New Jersey, U.S.A
huizhang@nec-labs.com

Guofei Jiang
NEC Labs America
Princeton, New Jersey, U.S.A
gfj@nec-labs.com

Haifeng Chen
NEC Labs America
Princeton, New Jersey, U.S.A
haifeng@nec-labs.com

Kenji Yoshihira
NEC Labs America
Princeton, New Jersey, U.S.A
kenji@nec-labs.com

Evgenia Smirni
College of William and Mary
Williamsburg, Virginia, U.S.A
esmirni@cs.wm.edu

ABSTRACT

Server virtualization brings benefits in autonomic resource management, but also leads to new challenges. The challenge the paper addresses is on profiling physical resource utilization information of VMs when consolidated on a single server. Profiling is very difficult due to dynamic mapping relationships of resource activities between the virtual layer and the physical layer. The problem is further exacerbated by cross-resource utilization causality relationships due to virtualization overhead and resource utilization multiplexing across different VMs. We formulate profiling as a source separation problem as studied in digital signal processing, and design a directed factor graph (DFG) to model the multivariate dependence relationships among different resources (CPU, memory, disk, network) across virtual and physical layers. A benchmark based methodology is designed to build a DFG based model for the VM information calibration problem. A run-time calibration mechanism is proposed based on the DFG based model and further enhanced with a robust remodeling method based on guided regression. The proposed methodology outputs estimates of physical resource utilization on individual VMs and physical server aggregate resource utilization. We present a case study using the Xen-virtualization platform and evaluate the methodology for different consolidation scenarios with diverse applications including RUBiS, IOzone, SysBench, and Netperf. The results show that the DFG calibration significantly improves the accuracy of the resource utilization information collected within guest VMs as it reduces relative errors in CPU utilization from 44.8% down to 3.9% for CPU-intensive applications and relative errors in disk write rate from 391.5% down to 10.6% for IO-intensive applications, strongly arguing for the effectiveness of the proposed DFG calibration methodology.

Categories and Subject Descriptors

G.3 [**Probability and Statistics**]: Multivariate statistics; I.6.5 [**Simulation and Modeling**]: Model Development; K.6.4 [**System Management**]: Quality assurance

General Terms

Management, Measurement

1. INTRODUCTION

Autonomic resource management becomes increasingly demanded in large-scale computing systems for reducing administrators' burdens. Autonomic resource management and system capacity planning often rely on analytic performance models [10, 8, 24]. Correct parameterization of such models is critical for their effectiveness [24]. Server virtualization clearly enhances flexibility in resource control, but introduces new challenges in parameterization of performance models. The relationship between application workload and physical resource utilization can be greatly obscured by the virtualization layer.

This paper addresses a fundamental problem in virtual machine (VM) resource management: how to effectively profile physical resource utilization of individual VMs. Our focus is not just on collecting usage statistics but on extracting the utilization of physical resources by a VM across time, where the resources include CPU (utilization in CPU cycles), memory (utilization in memory size), network (utilization in traffic volume), and disks (utilization in disk I/Os). Correct VM resource utilization information is tremendously important in any autonomic resource management that is model based. For example, in dynamic provisioning, correct per-VM resource utilization information is the basis for the right VM sizing decision; in application management, performance modeling requires correct per-VM resource utilization information to build the relationship between application performance and resource demands.

Profiling is a hard problem because mapping virtual-to-physical (V2P) resource activity mapping is not always one to one and may depend on application workload characteristics. The problem is further exacerbated by cross-resource utilization causality among different resources due to virtualization and multiplexing among VMs in a consolidated environment.

Here, we formulate VM resource utilization profiling as a source separation problem, which is originally studied in digital signal processing. The aggregate utilization information of one physical resource is viewed as a mixed signal superimposed by the utilization signals of every individual VM. The objective is to figure out what are the original per-VM "signals". In this paper we extend the factor graph model [11] with directionality and factoring generalization, and design a *directed factor graph* (DFG) that models the multivariate dependence relationships among different resources and across virtual and physical layers.

To build the base DFG model, we first focus on building separately DFG sub-graphs using micro-benchmarks and benchmark applications that are CPU-intensive (SPEC CPU 2006 [3]), memory-intensive (SPEC CPU2006), network- intensive (Netperf [2]) and disk I/O-intensive (IOzone [1], SysBench [4]). We also design a run-time calibration mechanism which outputs physical resource utilization estimation on individual VMs based on monitoring information and the DFG based model. The run-time calibration mechanism also includes a robust remodeling process that can make a new guided regression model to adapt to the temporal dynamics in the modeled resource relationships.

We use the Xen-virtualization environment and apply the calibration mechanism on a set of consolidated VMs hosting diverse applications including RUBiS (a 3-tier app), Netperf, IOzone, and SysBench. The VM resource calibration output is compared with the "baseline" case defined as the physical resource utilization when that VM is hosted *alone* on the same server and with the same application workload. The results show that the calibration mechanism significantly improves the accuracy of the resource utilization information that are collected within guest VMs, reducing the relative error in CPU utilization from 44.8% down to 3.9% for CPU-intensive applications, the relative error in disk write rate from 383% down to 16.7% for IO-intensive applications.

The rest of the paper is organized as follows. Section 2 presents the motivation and problem formulation. Section 3 describes the source separation problem in signal processing and factor graphs. In Section 4, we describe the definition of a directed factor graph model; in Section 5 we present the benchmark based methodology to build the DFG base model in Xen virtualization. Section 6 describes the run-time VM calibration approach along with a guided regression method for robust remodeling, and Section 7 presents the evaluation results of the VM calibration methodology. Section 8 presents the related work and Section 9 concludes this paper and outlines future work.

2. PROBLEM FORMULATION

In this section, we first present background to Xen virtualization and then report a few cases of representative VM measurement information mismatching that motivate us for the development of a reliable information calibration approach. Then, we present the problem formulation.

2.1 Xen Virtualization

Xen [5] is an open source x86 virtual machine monitor which can create multiple virtual machines on a physical server. Each virtual machine runs an instance of an operating system. A scheduler is running on the Xen hypervisor to schedule virtual machines on the processors. Domain-0 in Xen is a privileged control domain used to manage other domains and resource allocation policies.

Xen does not account for resource consumption in the hypervisor on behalf of an individual VM, e.g., for I/O processing. On Xen's I/O model, a special privileged virtual machine called driver domain (by default "Domain-0") hosts unmodified device drivers and directly controls physical devices. Other virtual machines, called guest domains in Xen, have to communicate through the driver domain to access the devices (e.g., network cards or disks). This I/O model results in a complex resource utilization model. For example, an IO-intensive application has two components in its CPU utilization: CPU consumed by the guest domain where the application runs and CPU consumed by the driver domain which performs I/O processing on behalf of the guest domain. When multiple VMs are co-hosted on a single physical server, a problem posed by the Xen I/O model is to classify the driver domain's CPU consumption across the various guest domains. Similar problems arise for classifying the resource consumption of network and disk activities.

2.2 Information Mismatching Paradox

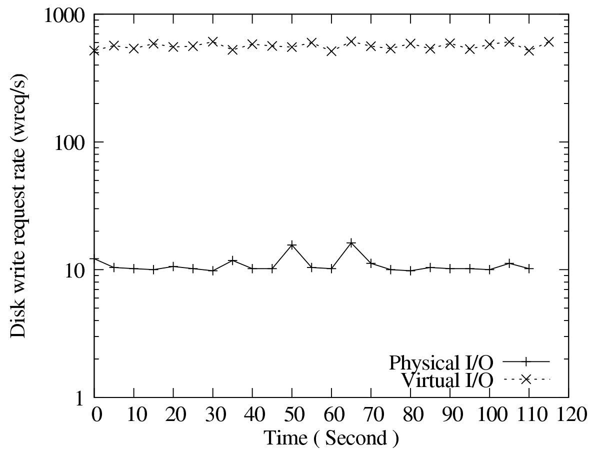

Figure 1: Measurement information mismatching: a disk I/O utilization example

Directly profiling VM's resource utilization inside the VM does not always give the correct information. For example, on a Xen-virtualized physical server hosting a single VM running IOzone [1] (a filesystem benchmark application), Figure 1 shows the disk I/O activities (write requests per second) measured inside the VM (called virtual I/Os) and the I/O activities measured on the physical disk (called physical I/Os). Both IO measurements are collected from the /proc file system in the guest domain and Domain-0 separately. There is more than one order of magnitude difference between the two readings.[1]

Figure 2 shows another example on CPU utilization. On a Xen-virtualized physical server hosting a single VM running an Apache web server, we use the VM monitoring tool XenMon [9] to measure the VM's CPU utilization and the physical server's CPU utilization. While only a single VM is running, the server's CPU utilization is more than twice of the CPU utilization of the VM. This mismatching is mainly caused by the CPU overhead of Domain-0 in network and disk IO processing.

[1]Careful examination reveals that this is caused by write coalescence at the cache subsystem in the disk I/O layer.

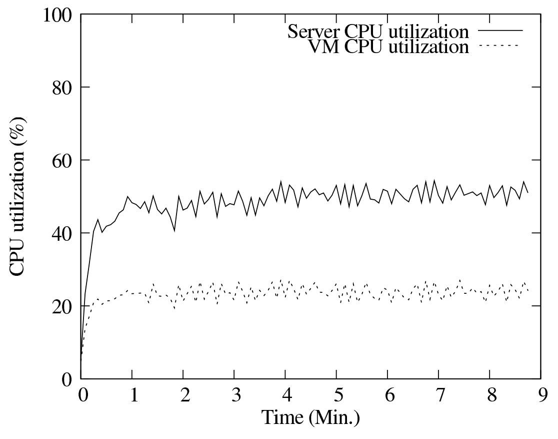

Figure 2: Measurement information mismatching: a CPU utilization example

2.3 Problem Formulation

We define the problem as profiling physical resource utilization for an individual VM. That is, we want to profile how many physical resources have been utilized by each VM across time, where resources include CPU, memory, network, and disk.

2.3.1 *Virtual Resource Monitored Information*

Per-VM resource utilization information can be collected within the VM (e.g., via the sar utility tool) or from the VM Manager (e.g., Domain-0 of Xen). We implement the monitoring system to track various VM resource usage without modifying any virtual server:

- **CPU:** we monitor the consumed CPU by every individual guest VM. In Xen's Domain-0, CPU usage of guest VMs and Domain-0 itself are provided by the XenMon utility [9].

- **Memory:** we collect the memory usage as the ratio of used memory and the total memory allocated to the guest VM. While memory utilization is only known to the OS within each VM, tracking accesses to swap partitions from Domain-0 can infer such information [23].

- **Disk:** we collect the disk IOs issued from the guest VM to the privileged Domain-0 in four metrics - *wtps* (write requests per second), *bwrtn/s* (data written to vbd block device in blocks per second), *rtps* (read requests per second), *bread/s* (data read from vbd block device in blocks per second). In Domain-0 of Xen, such information for the guest VMs is available at */sys/devices/xen-backend/vbd-<domid>-<devid>* for virtual block devices.

- **Network:** we collect the network traffic issued from guest VMs to Domain-0 in four metrics - *rxpck/s* (packets received per second), *txpck/s* (packets transmitted per second), *rxbyt/s* (bytes received per second), *txbyt/s* (bytes transmitted per second). In Domain-0 of Xen, such information on guest VMs is available in the proc filesystem at */proc/net/dev* for virtual network devices.

2.3.2 *Physical Resource Monitored Information*

The following resource utilization information is collected at the privileged driver domain (e.g., Domain-0 of Xen):

- **CPU:** consumed CPU by the privileged domain.

- **Memory:** memory utilization as the ratio of used memory to total allocated memory of the privileged domain.

- **Disk:** four types of metrics are collected for aggregate physical IO that are the same as those for virtual disks.

- **Network:** four metrics are collected for aggregate traffic on physical network cards that are the same as those for virtual network devices.

3. BACKGROUND INFORMATION

In this section, we describe the source separation problem defined in signal processing and a solution framework called factor graphs.

3.1 Source Separation

In digital signal processing, source separation problems [21] are those in which several signals have been mixed together and the objective is to find out what are the original signals. In particular, blind source separation is the source separation problem without any information about the source signals or the mixing process. Several approaches have been proposed for the solution of this problem type such as Singular Value Decomposition (SVD) and Principal Components Analysis (PCA). These approaches typically rely on the assumption that the source signals are mutually statistically independent. Unfortunately, such assumption does not always hold in our application. For example, CPU overhead and network traffic originated from multiple VMs may have strong correlation when they belong to the same application services. VM disk write requests can be accumulated (delayed) and executed in batches on physical disks due to the page cache mechanism at the OS layer. Therefore, we focus on model based source separation approaches where domain knowledge on the mixing process can be encoded in the separation process.

3.2 Factor Graphs

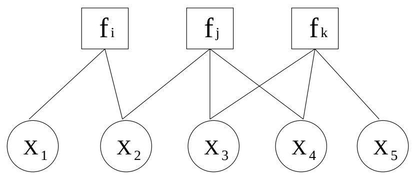

Figure 3: An example factor graph

Factor graphs [15] are graphical representations of complex mathematical models. They allow a unified approach to many source separation problems in signal processing and beyond. A factor graph is a bipartite graph representing the factorization of a global function of several variables. For example, assume that some global function, $f(x_1, x_2, x_3, x_4, x_5)$ can be factored as multiple local functions, e.g.,

$$f(x_1, x_2, x_3, x_4, x_5) = f_i(x_1, x_2)f_j(x_2, x_3, x_4)f_k(x_3, x_4, x_5)$$

This factorization is represented by the factor graph in Figure 3. In our application, we have to bring directionality into a factor graph so as to model a general decomposition of a global function into multiple local functions. We give details on this extension in the following section.

4. DIRECTED FACTOR GRAPHS

In this section, we present the factor graph model that we use in the VM resource calibration problem.

4.1 Graph Model

Formally, a directed factor graph (DFG) is a bipartite digraph $G = (V, F, E)$. V and F are two disjoint node sets. V represent the set of variables, F represents the set of functions. One edge $x \rightarrow f$ in E connects a vertex x in V to one vertex f in F when x is an input parameter of the function represented by f. One edge $f \rightarrow y$ in E connects a vertex f in F to one y in V when y is an output parameter of the function represented by f.

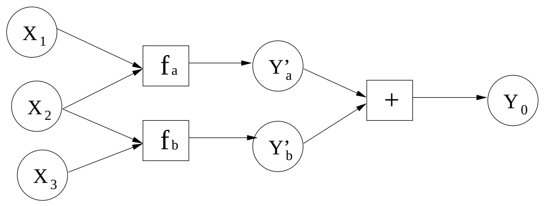

Figure 4: A directed factor graph example

Figure 4 shows the directed factor graph for a global function $Y_0 = g(x_1, x_2, x_3)$ with decomposition given as

$$g(x_1, x_2, x_3) = f_a(x_1, x_2) + f_b(x_2, x_3)$$

in Figure 4. The new variable nodes Y'_a, Y'_b are two temporary variables recording the output of the functions f_a and f_b.

4.2 DFG in VM Information Calibration

We use the directed factor graph in Figure 5 as the base for VM information calibration. From left to right, the virtual resource activities are first transformed into the physical resource activities generated by each VM, and then are aggregated to render the physical resource activities of the hosting server. The left-most variable nodes represent observable virtual resource activities, the right-most variable nodes represent observable physical resource activities, see Section 2.3. The intermediate variable nodes, such as CPU^p_{VM-1} representing the physical CPU consumption by VM-1, are the data we want to infer and we derive them through statistical inference techniques on the function nodes such as f^{VM-1}_{CPU}.

We choose the DFG model in Figure 5 as it naturally describes the resource demand transformation and aggregation processes in a virtualization environment. The edges in the graph depict statistical causality relationships between resource utilization at different components/layers. The generalization of this graph model is possible thanks to the flexibility in the identification of the function nodes, which may be different for different hypervisor architectures. In the following section, we show how to build a base model for Xen by identifying the function nodes through benchmark profiling and regression analysis.

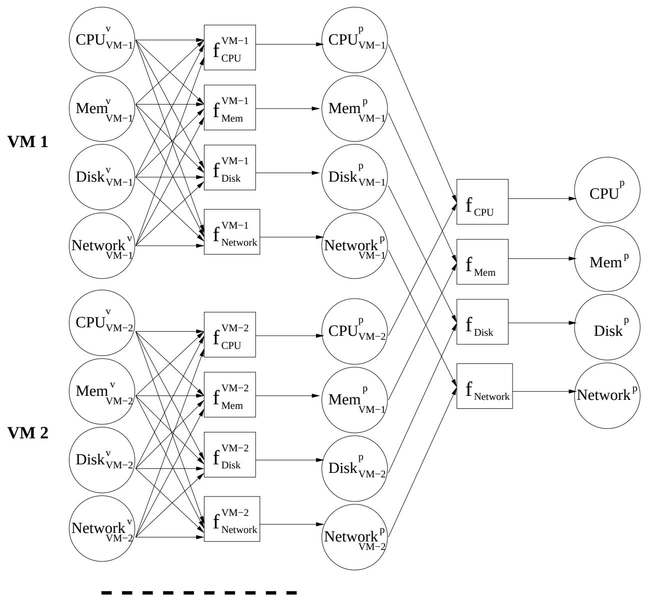

Figure 5: The directed factor graph in VM monitoring information calibration.

5. DFG BASED MODEL

In this section, we run different benchmark applications in one guest domain to generate workload on each virtualized resource separately and build a DFG based model for our calibration mechanism.

5.1 Methodology

The modeling process consists of the following steps:

1. Host a single VM in a server.

2. Run a benchmark for a specific virtual resource (e.g., a CPU-intensive benchmark).

3. Apply statistics analysis to find out the set of physical resources on which the benchmark incurs non-negligible utilization and learn the models for the function nodes such as f^{VM-1}_{CPU}.

The benchmark based modeling process aims at capturing the stable causality relationships between virtual and physical resource demands. We carefully select a fixed set of benchmark applications to cover all the four resources (CPU, memory, disk, and network) at the virtual layer. If the causality relationship changes across time, as for example in the disk IO access patterns of an application, then we resort to the guided regression method that is described in Section 6.2.

5.2 Regression Analysis

In Step 3, stepwise regression [7] is applied to the collected data to find out correlated measurement variables and to remove co-linearity that may exist between variables. Stepwise regression uses the same analytical optimization procedure as multiple regression but differs in that only a subset of predictor variables is selected sequentially from a group of predictors by means of statistical testing of hypotheses.

5.2.1 Source Node: Virtual CPU load

We first run a micro-benchmark in the guest VM that alternates between sleeping and calculating Fibonacci numbers, the ratio of which determines the VM CPU utilization. Figure 6 shows the CPU overhead in the privileged domain while the micro-benchmark VM's CPU utilization changes from 5% to 50%. The Domain-0 CPU overhead is stable and remains close to 0. Figure 6 also shows the Domain-0 overhead when the guest VM runs *gromacs*, a CPU-bound SPEC CPU2006 benchmark. The Domain-0 overhead is close to 0 while the guest VM uses up its allocated CPU capacity (one core of the dual-core processor in the server).

We also observe (not shown on this graph) that the CPU-intensive guest VM did not incur overhead on other server resources (e.g., network or disk).

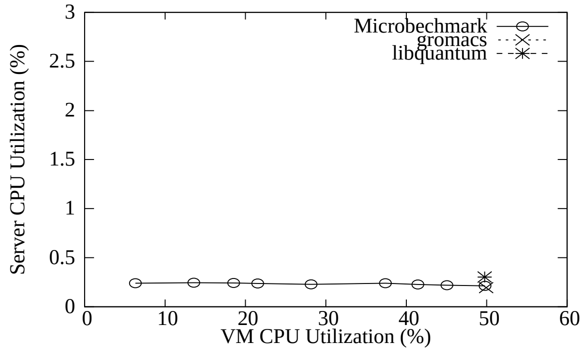

Figure 6: Compute intensive workload has no impact on privileged domain performance

5.2.2 Source Node: Virtual Memory load

We examine how memory-intensive applications on guest domains impact the resource utilization in the privileged domain. Figure 6 shows the Domain-0 CPU overhead when running *libquantum*, a memory-bound SPEC CPU2006 benchmark, and the overhead is close to 0. We also observe (but not include these results here) that the memory-intensive guest VM did not impose overhead on other server resources (e.g. network or disk).

5.2.3 Source Node: Virtual Network load

We use Netperf [2] to perform controlled network load generation. Netperf allows both UDP and TCP stream tests. Here, we measure the number of packets (bytes) sent/received per second for both guest and privileged domains.

For the UDP case, we change the packet sending rate from 1,000 to 36,000 pkts/sec, the packet payload size from 100 bytes to 1,400 bytes, and measure server resource metrics on both sender and receiver sides. Since the TCP protocol is not allowed to specify the transfer speed, we can not change the sending rate settings as in the UDP protocol.

The regression coefficients of different metrics from stepwise regression algorithm is shown in Table 1, and only the correlated variables are presented. Each row corresponds to a Domain-0 resource metric and each column corresponds to a guest domain variable. For example, the first row in the figure means the privileged domain CPU overhead(%) $= 6.64 \times 10^{-4} \times \text{recv\_pkt\_rate} + 5.02 \times 10^{-4} \times \text{send\_pkt\_rate} - 0.06$. It shows that the Domain-0 CPU overhead has a clear linear relationship with the packet sending and receiving rates, but not with the network throughput in bytes. The same observation is also reported by Wood et al. [22].

Table 1: Network regression model.

		Domain-U				
		rxbyt/s	rxpkt/s	txbyt/s	txpkt/s	intercept
D	CPU(%)	0	6.64e-04	0	5.02e-04	-0.06
o	rxbyt/s	1.01	0	0	0	-49.51
m	rxpkt/s	2.24e-06	1.00	0	0	2.56
-	txbyt/s	0	0	1.00	20.38	1094.2
0	txpkt/s	0	0	-4.14e-06	1.01	4.26

5.2.4 Source Node: Virtual Disk IO load

Xen supports many different storage options for the guest domain. These options can be divided into three categories: *file based*, *device based*, and *LVM-based*. The file-based block devices can be differentiated by how Xen accesses them: *blktap* and *loopback*. Blktap replaces the common loopback driver for file-based images because it allows for improved performance and more versatile filesystem formats, such as QCOW [16]. It also avoids problems related to flushing dirty pages which are present in the Linux loopback driver.

We build our model for guest domains with file based disk storage. For the other two storage options, the same methodology can be also applied. Here, we perform controlled IO-intensive experiments with SysBench [4]. SysBench is a multi-threaded benchmark tool for evaluating database (e.g., MySQL) server performance under intensive load. We exploit its file IO performance testing functionality to generate different IO activities. To control the write/read operation rate, we add different sleeping time between each write/read operation in the source code. The SysBench IO testing module supports six IO operations: sequential write, sequential rewrite, sequential read, random read, random write, and combined random read/write. We take samples of each IO operation and create a dataset based on all the samples. The block size is set to 16K bytes. The total size of testing files is set to 4G bytes. We choose "default" for other option settings.

Tables 2 and 3 show the regression models extracted by blktap based and loopback based devices. We note that besides the difference on CPU overhead, the relationship functions from virtual disk IOs to physical IOs are also different for blktap based and loopback based devices. For example, the coefficient of virtual *rtps* to physical *rtps* is 1.29 for blktap based devices, while the coefficient is 0.34 for loopback based devices. We observe (but not include the results here) that the regression models for disk IOs are dynamic and dependent on workload patterns (e.g., sequential vs random access, high vs low access locality). The coefficients in the regression models of Tables 2 and 3 represent the resource relationships under the standard SysBench workload. Later, we present a model relearning scheme for online information calibration that adapts to inevitable workload dynamics, see Section 6.

5.2.5 System Overhead

When multiple VMs are consolidated in one physical machine and lead to heavy IO utilization, possible system overhead needs to be considered . One example of the source of such overhead is the *ksoftirqd* daemon process. *ksoftirqd* is a per-CPU kernel thread that runs when the machine is under

Table 2: Blktap based device regression model

		Domain-U				
		rtps	bread/s	wtps	bwrtn/s	intercept
D	CPU(%)	0	2.02e-04	0	2.21e-04	0.47
o	rtps	1.29	0	0	0	10.57
m	bread/s	0	1.00	0	0	1.59
-	wtps	0	0	0.11	0.0018	1.6
0	bwrtn/s	0	0	0.07	0.998	15.3

Table 3: Loopback based device regression model

		Domain-U				
		rtps	bread/s	wtps	bwrtn/s	intercept
D	CPU (%)	0	8.27e-05	0	1.11e-04	0.23
o	rtps	0.34	0	0	0	0.14
m	bread/s	0	1.0	0	0	-0.53
-	wtps	0	0	0.10	0	1.52
0	bwrtn/s	0	0	0	0.997	22.1

heavy soft-interrupt load. If a soft interrupt is triggered for a second time while soft interrupts are being handled, the *ksoftirq* daemon is triggered to handle the soft interrupts in process context. The sudden run of *ksoftirqd* daemon process under heavy network workload could lead to unexpected CPU utilization bursts. In our solution, this type of overhead is taken as system noise and is excluded from the aggregate resource utilization contributed to guest VMs.

6. INFORMATION CALIBRATION

In this section, we present the run-time calibration mechanism. The mechanism takes as input the VM monitoring information as described in Section 2.3 and outputs per VM physical resource utilization information based on the DFG model in Section 4.

6.1 Run-time Calibration Mechanism

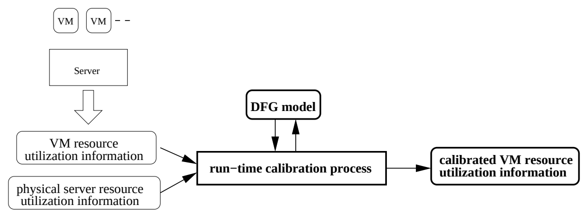
VM
VM
Server
DFG model
VM resource utilization information
physical server resource utilization information
run−time calibration process
calibrated VM resource utilization information

Figure 7: The run-time calibration mechanism

Figure 7 shows the overview of our run-time calibration mechanism. The inputs to the mechanism are the resource utilization monitoring information within the guest VMs and the privileged domain (for aggregate physical server load information). The mechanism uses the DFG model to decode the input information and outputs per-VM physical resource utilization information. In the run time, the calibration process may also update the DFG model if the existing one incurs non-trivial estimation errors.

We exemplify the algorithmic steps in the context of CPU utilization, but the same steps apply for all four resources. The run-time process on per-VM CPU utilization information calibration is given in Algorithm 1.

In Step 0, the initial model parameters are obtained from benchmark based profiling, see Section 5, or from offline application-specific profiling for calibrated VMs. While the latter method is expected to give a more accurate model than the former one, it comes with an extra profiling overhead. In the evaluation, we use the benchmark based profiling results for this step.

```
0 Initialize the model parameters of DFG function
  nodes;
1 Feed the DFG model with per-VM virtual resource
  utilization information;
2 Calculate the value of DFG latent variables on
  per-VM CPU utilization information;
3 Calculate the value of server CPU utilization variable;
4 while $CPU\_err > thresh$ do
5     Re-learn the DFG function models;
6     Re-calculate the value of the DFG latent variables
      for per-VM CPU utilization information;
  end
7 Output per-VM CPU utilization information;
```

Algorithm 1: per-VM CPU utilization information calibration algorithm

Step 4 brings a feedback loop to make our calibration process adaptive to inevitable model dynamics, typically caused by the change of workload patterns. As shown in Section 5, the relationship between a virtual resource activity and its overhead on physical resources can vary and depend on the workload contents. The mapping of virtual I/O to physical I/O activities is one such example. To be robust to transient workload changes or monitoring noise, the discrepancy is calculated on the average of the estimation errors during a sliding window including the past K time points. The threshold is chosen as $(\epsilon + Z_\alpha * \sigma)$, where (ϵ, σ^2) is the mean and variance of the regression model estimation error from the last remodeling process (or those learned from the benchmark based profiling at the beginning of the process). Z_α is the standard score in statistics [13], and here measures how unlikely an estimation error is if the current model is correct. If $Z_\alpha = 3$, $\alpha = 99.75\%$, then an estimation error larger than $(\epsilon + 3 * \sigma)$ is unlikely (with probability $< 0.25\%$) to appear if the virtualization environment were the same as during the last remodeling process. Therefore, if several large estimation errors in a row indicate the change of some factors in the virtualization environment, then a remodeling process is triggered. This process is presented in the next subsection.

6.2 Robust Remodeling: Guided Regression

While we use linear regression models in Section 5 for single VM based benchmark profiling, a new problem arises in the calibration process when multiple VMs are co-hosted in a single server: now $\mathbf{y}$ represents a physical resource utilization which is the summation of physical resource utilization of multiple VMs. Since the physical resource utilization of each individual VM is a latent variable, a straightforward regression model is as follows, assuming m co-hosted VMs:

$$\begin{aligned} y^{(i)} &= (\beta_1^{VM-1} x_1^{VM-1(i)} + \cdots + \beta_p^{VM-1} x_p^{VM-1(i)}) + \\ &\quad \ldots + (\beta_1^{VM-m} x_1^{VM-m(i)} + \cdots + \beta_p^{VM-m} x_p^{VM-m(i)}) \end{aligned}$$

where $y^{VM-j(i)} = \beta_1^{VM-j} x_1^{VM-j(i)} + \cdots + \beta_p^{VM-j} x_p^{VM-j(i)}$ is the latent variable for the j^{th} VM.

If we directly solve the above problem with the least square solution

$$\hat{\beta} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y} \ , \tag{1}$$

it could lead to re-learning the models of all the VMs in the server. We seek to enhance the original regression modeling method for remodeling robustness due to three reasons. The first comes from common run-time monitoring data error and noise (e.g., system noise, transient VM migration overhead) that might add transient perturbation onto otherwise stable resource relationships. The second is due to the factor that some relationships (such as virtual disk I/Os and its resource overhead) are naturally dynamic due to their content dependence. Re-learning those models should not affect the model of other stable relationships. The third reason is due to the fact that since the co-located VMs are all involved in the regression model, the number of unknown parameters in β is large. In order to obtain accurate estimation of those parameters, a significant amount of measurements $[\mathbf{x}, \mathbf{y}]$ is usually required. However, in the model relearning process, sometimes we do not have so many observations due to the quick dynamics of the system. The lack of (enough) data may lead to large variances of the final solution β.

In order to enhance the robustness of model estimation, we propose a guided regression process to solve the model of Eq. (1). We add some constraints to describe the range of possible β values and embed those constraints into the estimation process. The constraints can come from various sources, such as the prior model knowledge based on the benchmark profiling or the model learned during the previous time period. By including such knowledge to guide the estimation, we can obtain a more reliable solution β for the regression model.

The prior constraints on β are represented by a Gaussian distribution with the mean $\bar{\beta}$ and covariance Σ

$$P(\beta|\sigma^2) = (\sigma^2)^{-K} exp\left\{-\frac{1}{2\sigma^2}(\beta - \bar{\beta})^T\Sigma^{-1}(\beta - \bar{\beta})\right\} \tag{2}$$

The mean $\bar{\beta}$ represents the prior expectation on the values of β, and is determined from the β values learned in Section 5. The covariance Σ represents the confidence of our prior "knowledge". We choose Σ as a diagonal matrix $\Sigma = diag(c_1, c_2, \cdots, c_p)$, in which the element c_i determines the level of variances of β_i in the prior distribution. If we are confident that the value of β_i is located closely around $\bar{\beta}$, the corresponding c_i value is small. Otherwise we choose large c_i values to describe the uncertainty of β_i values. Note that the least squares method in Eq (1) solves the regression without any prior knowledge, i.e., the values of c_i's are infinite, which may be inaccurate when the number of collected measurements is insufficient.

There is also an unknown parameter σ^2 in Eq (2), which represents the variance of the data distributions. Here, we use the inverse-gamma function [13] to represent the distribution of σ^2:

$$P(\sigma^2) = \frac{b^a}{\Gamma(a)}(\sigma^2)^{-(a+1)} exp\left\{-\frac{b}{\sigma^2}\right\} \tag{3}$$

where a, b are two parameters to control the shape and scale of the distribution, $\Gamma(a)$ is the gamma function of a. We choose the inverse-gamma function because: 1) it is one of the common distributions for non-negative variables such as the variance studied here; 2) it is easy to tune its shape by setting (a, b) parameters. 3) By using the inverse-gamma function as the prior of Gaussian variance, we can obtain a closed-form solution for optimizing the posterior distribution estimation.

Given the prior distribution $P(\beta)$, the guided regression finds the solution by maximizing the following posterior distribution

$$P(\beta|\mathbf{x}, \mathbf{y}, \sigma^2) \propto P(\mathbf{y}|\beta, \mathbf{x})P(\beta|\sigma^2)P(\sigma^2) \tag{4}$$

which leads to the following solution

$$\beta^* = (\mathbf{x}^T\mathbf{x} + \Sigma^{-1})^{-1}(\Sigma^{-1}\bar{\beta} + \mathbf{x}^T\mathbf{x}\hat{\beta}). \tag{5}$$

Due to the space limit, we do not present the detail of the above derivations.

To summarize, the robust remodeling process takes the following steps: 1) decides the prior coefficients $\bar{\beta}$ and their weight metric Σ (e.g., learned through the benchmark profiling in Section 5); 2) solves Eq.(1) based on the run-time monitoring data for the standard least square solution; 3) calculates the final solution β^*, which is a weighted average of the two components from 1) and 2).

7. EVALUATION

As discussed in the previous sections, the proposed DFG based model and run-time calibration mechanism constitute the two building blocks for VM resource utilization information calibration process. These building blocks can be directly applied to existing applications. In this section, we demonstrate three case studies that clearly show the effectiveness of the calibration methodology.

7.1 Experimental Methodology

We evaluate the effectiveness and accuracy of our calibration technique with different applications. The testbed runs the Fedora release 8 operating system with Linux kernel 2.6.18-8. The evaluation is based on the Xen virtualization platform version 3.3.1. Our testbed platform uses Supermicro 1U Superservers with Intel Core 2Duo E4300 1.86 GHz processors, 2MB L2 cache. All servers have a RAM of 2GB and 250GB 5400RPM disk. The servers are connected through D-LINK DES-3226L 10/100Mbps switches. The testbed is managed by Usher [17], an open source VM management middleware with a centralized monitoring database. The run-time calibration process is co-located with the monitoring DB and it calibrates the raw monitoring data in batches with fixed time window size.

The following applications are used in our evaluation:

- RUBiS is an auction site prototype modeled after eBay. A client workload generator emulates the behavior of users browsing and bidding on items. We use the Apache/EJB implementation of RUBiS version 1.4.3 with a MySQL database server version 5.0.77.
- IOzone is used for filesystem benchmarking. It is used to generate and measure various disk I/O activities.
- SysBench is a multi-threaded benchmark tool for evaluating database (e.g., MySQL) server performance under intensive load. We use SysBench to generate MySQL workloads which lead to various I/O activities.
- Netperf is tool for network benchmarking (see Section 5.2.3). We use Netperf to generate different network traffic workloads.

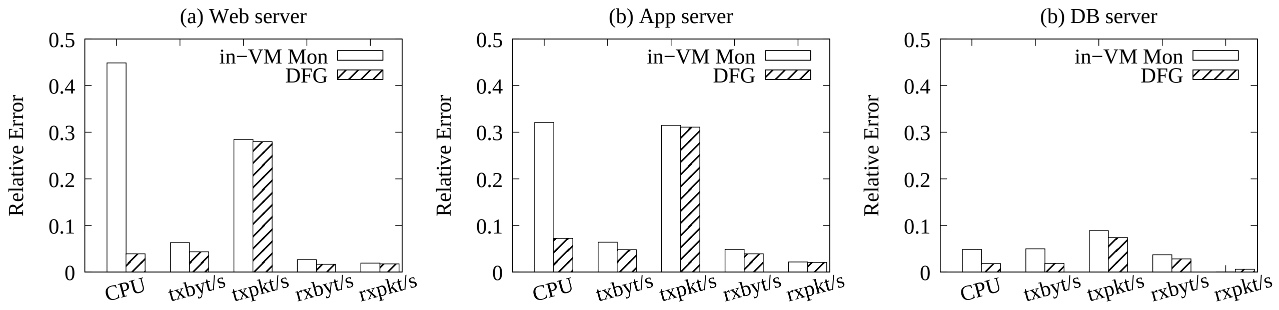

Figure 8: Relative error of in-VM monitoring method and DFG based model in RUBiS app

Using these applications as components, we present three scenarios to test the accuracy of the DFG based model. In scenario 1, we consider the RUBiS application. We collect baseline information from every component server i.e., web server, application server, and database server. We demonstrate substantial improvements in resource information estimation with the DFG based model for two different server consolidation environments. In scenario 2, we illustrate how guided regression can be used to improve on the DFG based model. The estimation after the remodeling process shows dramatic improvements. In scenario 3, we illustrate how the DFG model is used to separate a mixed disk IO stream into its component streams.

7.2 Results

7.2.1 Scenario 1: RUBiS, a 3-tier App

RUBiS is a multi-tier web service application composed by open-source software, i.e., Apache Web Server, JBoss EJB Server, and MySQL relational database. For all servers, we use blktap based virtual block devices. The resource usage baseline of each server is collected when it is virtualized and run in the same physical machine alone. Here, RUBiS is initialized with 700 simultaneous clients with the browsing workload. For the baseline comparison, we run RUBiS for 30 minutes and collect usage statistics.

We setup the following server consolidation environment (two physical machines): on Machine #1, the virtualized App server and DB server are consolidated together; the web server VM is placed on Machine #2. We collect each virtual and physical machine run-time information using the techniques described in Section 2.3. The calibrated usage information is calculated with the DFG based model described in Section 5.

Figure 8 illustrates the relative errors[1] of both in-VM monitoring and DFG based model. The figure shows results for all three component servers. We observe that RUBiS is mainly a compute and network IO intensive application with very low disk activities. In the figure, we only show metrics with significant values and ignore those close to zero. One can easily see that the DFG based model significantly reduces the relative error, e.g., the error in web server CPU utilization drops from 44.8% down to 3.9%.

The substantial improvements in CPU utilization estimation is due to the fact that DFG based model takes the intensive network activity overhead into consideration. In the web server example, on the average, the server transmits about $12,500$ packets per second to the clients and application server. Meanwhile, it also receives about $13,400$ packets per second. According to the DFG based model, this amount of network traffic leads to 13% privileged domain CPU overhead that is not reported by in-VM monitoring.

[1]Relative error is defined as the ratio of absolute error to its corresponding baseline value.

7.2.2 Scenario 2: Co-hosting network- and IO-intensive Apps

For this experiment we consolidate two virtualized servers on one physical machine. Both VMs are configured with loopback based virtual block devices. The first VM runs Netperf that sends out UDP packets at the rate of 25Mbps and the second VM runs SysBench. SysBench is set to run in the “oltp” test mode, to emulate a real database. For our testbed, we choose the MySQL implementation and set the number of rows in the testing table to $5,000,000$. To make the testing more real, we select the execution mode to be “advanced transactional” in which each thread performs transactions.

In the experiment, we set a sliding window length to 15 minutes. At the beginning of the first window, the parameters of DFG are initialized according to Tables 1 and 3. The system is set to take monitoring samples every 10 seconds and report to the center database. According to the run time calibration mechanism in section 6, at the end of each window, the program calculates the estimation error between the current DFG model and physical server measurement as shown in step 4 of Algorithm 1.

Table 4 shows the in-VM monitoring information, the DFG model result, and the resource usage of the physical server which is hosting the two VMs. We only report the metrics with significant values.

Table 4: Example of DFG error that triggers remodeling mechanism

	CPU	txbyt/s	txpkt/s	wtps	bwrtn/s
in-VM1-mon	27.45	0	0	2052.77	30859.36
in-VM2-mon	0.83	3.08e+06	3000	0	0
DFG	5.14	3.15e+06	3021	214.81	**30773.46**
Server resource	5.74	3.14e+06	3009	176.96	**7927.70**

The large estimation error highlighted in bold triggers the remodeling mechanism. Note that remodeling is only applied for the IO write metrics. The model of other resources is kept unchanged. The new model is based on the data points collected in the previous time window. The new disk IO write model replaces the original one. Close examination indicates that the large estimation error (see the bold value in the table) happens because that loopback based storage uses the Domain-0 kernel page cache. When a file write occurs, the page backing the particular block is looked up. If it is already found in cache, the write is done to that page in memory.

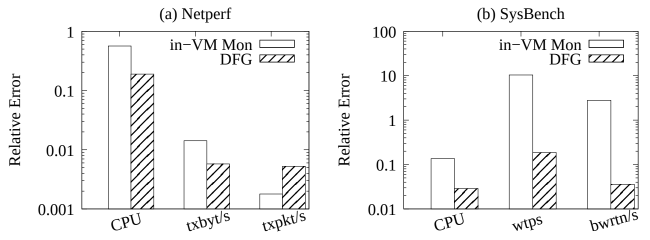

Figure 9: Relative error for in-VM monitoring method and run time calibration mechanism

Figure 9 illustrates the relative error for in-VM monitoring method and run time calibration mechanism. Note the log-scale on the Y-axis. One can easily see the significant improvement in the estimation accuracy. The relative error of the “bwrtn/s” is reduced from 276% down to 3.5%. The “txpkt/s” metric in Figure 9(a) is still better than the calibration result after remodeling but the run time calibration has already given a good estimation which only has a relative error of 0.5% only.

7.2.3 Scenario 3: Co-hosting IO-intensive Apps

In scenario 3, we show the case study of how the DFG based model is useful in decomposing disk IO. We setup two virtualized servers on one physical machine. The first VM server executes the IOzone benchmark to perform only writes and re-writes. The second VM server runs the SysBench benchmark and writes to disk at the rate of 8MBps. The block size is set to 16K bytes and the total size of testing files is 8GB. We choose “default” for other options. In both cases, we use blktap based virtual block devices.

The effectiveness of DFG based model in decomposing mixed IO and CPU is shown in Figure 10. The DFG based model successfully decomposes the mixed CPU utilization and reduces the relative error in write request rate from 391.5% down to 10.6% in Figure 10(a) and from 304.4% to 31% in Figure 10(b). Meanwhile, the figure shows that relative error in CPU is also reduced significantly.

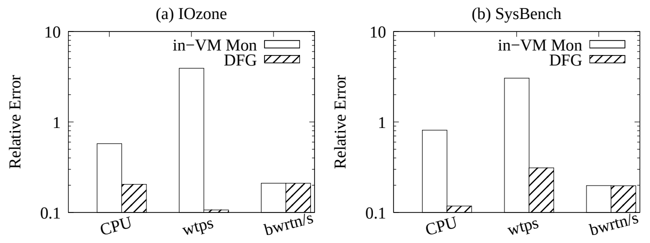

Figure 10: Relative error for in-VM monitoring and DFG model in mixed signal decomposing

8. RELATED WORK

Our benchmark based DFG model profiling is motivated by application-specific benchmarking [6, 20]. In app-specific benchmarking, a set of micro-benchmarks are run on a system to generate a system vector of resource utilization information, which serves as the fundamental primitives of the system; an application running on the system will be constructed with an application vector based on its resource utilization information, and that vector will be used with the system vector to produce a relevant performance metric for benchmarking purposes. Gupta et al. [8] present the design and evaluation of a set of primitives implemented in Xen to enforcing performance isolation among VMs. They look into per-VM CPU overhead in the driver domain caused by network traffic and use a linear model to approximate their relationships. Our work is complementary to [8] on driver domain CPU overhead modeling and extends to other resources including disk I/O and memory.

Wood et al. [22] propose a combination of application modeling and virtualization overhead profiling for estimating the Domain-0 and Domain-U CPU utilization of an application when it is moved from native to virtualized hardware. They use micro-benchmarks to profile the relationships of different I/O activities to the CPU overhead, apply robust stepwise linear regression method to build the models, and predict an application’s CPU demand after virtualization based on the benchmark models and the application’s native resource utilization. Our work is different from theirs in the following points: (1) our calibration process is a runtime process where a feedback loop controls the remodeling process, while their prediction process is a one-time offline profiling with a fixed set of regression models; (2) our calibration process covers three other resources in addition to CPU, and we show in Section 5 that there are situations where the virtual activities are not equal to their physical activities for some non-CPU resources. (3) our DFG method is a source separation framework where different functional modeling approaches can be used as plug-ins, as it is not limited to linear regression.

Isci et al. [10] study the run-time CPU demand estimation in VM consolidation for effective dynamic resource management. They derive a simple and accurate alternative estimate of CPU demand even when a server is overloaded with VMs hosting CPU-intensive applications. Extending their idea to other type of applications (e.g., IO-intensive) and other type of resources is interesting and important.

Kraft et al. [12] propose a trace-driven approach to predict the performance degradation of disk request response times due to storage device contention in consolidated virtualized environment. However, our approach is a run-time process without trace and is designed to solve the problem for different resources.

Pacifici et al. [18] consider a dynamic CPU demand estimation problem for web applications. They use statistical and classification methods to determine the CPU demand for different web request types. While that technique relies on application domain knowledge, our calibration process, as a commonly applicable solution, can be integrated with it to provide more accurate resource utilization information.

Virtualization technologies evolve in a fast speed, and many new approaches have been proposed to address virtualization overhead concern. For example, Liu et al. [14] propose hypervisor-bypassing in Xen to reduce the performance penalty of network I/O; Santos et al. [19] designs an optimized network IO scheduling algorithm to improve network throughput in Xen. These results bring more dynamics into the relationships between physical and virtual resource activities, and call for the necessity of an adaptive calibration solution like the one presented in this paper.

9. CONCLUSIONS & FUTURE WORK

In this paper, we present the design and evaluation of a VM monitoring information calibration mechanism. We formulate our problem as a source separation problem and base our solution on a directed factor graph. We show how to build a base DFG model through benchmarking and design a run-time remodeling solution which is adaptive and guided by the base DFG model. Our evaluation shows that the proposed methodology is robust as it successfully calibrates the VM monitoring information and compares well to baseline measures.

Virtualization overhead greatly depends on the hardware platform. For example, [22] shows that their regression models learned on an Intel platform will have large prediction error for applications running on AMD platform. While our run-time calibration process is adaptive to platform heterogeneity through model relearning, a low-overhead cross-platform modeling technology can still benefit our base model profiling process.

ACKNOWLEDGMENTS

We thank the anonymous reviewers and our shepherd, Jeff Kephart, for their valuable feedback on this paper. This work has been done during Lei Lu's summer internship at NEC Labs. Evgenia Smirni is partially supported by NSF grants CCF-0811417 and CCF-0937925.

10. REFERENCES

[1] *IOzone Filesystem Benchmark*. http://www.iozone.org.

[2] *Netperf*. http://www.netperf.org/netperf/.

[3] *Standard Performance Evaluation Corporation*. http://www.spec.org/cpu2006/.

[4] *SysBench: a system performance benchmark*. http://sysbench.sourceforge.net/.

[5] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, 2003.

[6] A. B. Brown and M. I. Seltzer. Operating System Benchmarking in the Wake of Lmbench: a Case Study of the Performance of NetBSD on the Intel x86 Architecture. *SIGMETRICS Perform. Eval. Rev.*, 25:214–224, June 1997.

[7] N. R. Draper and H. Smith. *Applied Regression Analysis*. Wiley-Interscience, 1998.

[8] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing Performance Isolation Across Virtual Machines in Xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 342–362, 2006.

[9] D. Gupta, R. Gardner, and L. Cherkasova. XenMon: QoS Monitoring and Performance Profiling tool. Technical report, 2005.

[10] C. Isci, J. E. Hanson, I. Whalley, M. Steinder, and J. O. Kephart. Runtime Demand Estimation for Effective Dynamic Resource Management. In *NOMS'10*, pages 381–388, 2010.

[11] V. M. Koch. *A Factor Graph Approach to Model-Based Signal Separation*. Hartung-Gorre Verlag Konstanz, First edition. 328 pages. ISBN-10: 3-86628-140-4., 2007.

[12] S. Kraft, G. Casale, D. Krishnamurthy, D. Greer, and P. Kilpatrick. IO Performance Prediction in Consolidated Virtualized Environments. In *Proceeding of the second joint WOSP/SIPEW international conference on Performance engineering*, ICPE '11, pages 295–306, 2011.

[13] R. J. Larsen and M. L. Marx. *An Introduction to Mathematical Statistics and its Applications*. Prentice Hall; 5 edition, Englewood Cliffs, NJ, 2011.

[14] J. Liu, W. Huang, B. Abali, and D. K. Panda. High Performance VMM-Bypass I/O in Virtual Machines. In *Proceedings of the annual conference on USENIX '06 Annual Technical Conference*, pages 29–42, 2006.

[15] H.-A. Loeliger. An Introduction to Factor Graphs. In *IEEE Signal Processing Magazine*, pages 28–41, January 2004.

[16] M. McLoughlin. *The QCOW image format*. http://www.gnome.org/ markmc/qcow-image-format.html.

[17] M. McNett, D. Gupta, A. Vahdat, and G. M. Voelker. Usher: an extensible framework for managing custers of virtual machines. In *Proceedings of the 21st conference on Large Installation System Administration Conference*, pages 1–15, 2007.

[18] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Dynamic Estimation of CPU Demand of Web Traffic. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools*, ValueTools'06, 2006.

[19] J. R. Santos, Y. Turner, G. Janakiraman, and I. Pratt. Bridging the Gap between Software and Hardware Techniques for I/O Virtualization. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 29–42, 2008.

[20] M. Seltzer, D. Krinsky, K. Smith, and X. Zhang. The case for application-specific benchmarking. In *Proceedings of the The Seventh Workshop on Hot Topics in Operating Systems*, HOTOS '99, pages 102–107, 1999.

[21] L. Surhone, M. Timpledon, and S. Marseken. *Source Separation: Digital Signal Processing, Signal (Electronics), Principal Component Analysis, Independent Component Analysis, Auditory Scene Analysis, Magnetoencephalography*. Betascript Publishers, 2010.

[22] T. Wood, L. Cherkasova, K. Ozonat, and P. Shenoy. Profiling and Modeling Resource Usage of Virtualized Applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 366–387, 2008.

[23] T. Wood, P. J. Shenoy, A. Venkataramani, and M. S. Yousif. Black-box and Gray-box Strategies for Virtual Machine Migration. In *Proc. of NSDI*, pages 229–242, 2007.

[24] Q. Zhang, L. Cherkasova, and E. Smirni. A Regression-Based Analytic Model for Dynamic Resource Provisioning of Multi-Tier Applications. In *Proceedings of the Fourth International Conference on Autonomic Computing*, page 27, 2007.