Control-Oriented Multirate LPV Modelling of Virtualized Service Center Environments

Mara Tanelli, Nicola Schiavoni, Danilo Ardagna and Marco Lovera

Abstract—To reduce the operating and energy costs of IT systems, nowadays software components are executed in virtualized servers, where a varying fraction of the physical CPU capacity is shared among running applications. Further, in service oriented environments, providers need to comply with the Service Level Objectives (SLO) stipulated in contracts with their customers. The trade-off between energy consumption and quality of service guarantees can be formalized in terms of a constrained control problem. To effectively solve it, accurate models of the virtualized server dynamics are needed. In this paper, an LPV multirate modeling approach is proposed, and its suitability for the considered application is assessed on experimental data.

I. INTRODUCTION

In recent years, the impact of energy consumption associated with Information Technology (IT) infrastructures has been increasing. The reduction of energy usage is one of the primary goals of green computing, a new discipline and practice of using computing resources with a focus on the impact of IT on the environment. For example, the US Department of Energy’s 2007 estimate of 59 billion kWh spent in US service centers translates to several million tons of coal consumption and CO2 emission per year. Overall, IT accounts for 2% of global CO2 emissions. This means that IT pollutes to the same extent as global air traffic. Costs reduction has also an important role, as IT analysts predict that by 2012 up to 40% of an enterprise technology budget will be consumed by energy costs, [1].

To design and manage sustainable service center environments, two key technologies have been adopted up to now: CPU Dynamic Voltage Scaling (DVS), and server virtualization. To reduce energy costs, modern CPUs implement the DVS mechanism which allows varying both the CPU supply voltage and the operating frequency. The adoption of DVS as a control variable is very promising, as power consumption is proportional to the cube of the operating frequency, whereas servers performance varies linearly with the operating frequency. Hence, under light load conditions, energy consumption can be effectively reduced by lowering CPU frequency without affecting the provided Quality of Service (QoS) level. In [2], [3] Linear Parametrically Varying (LPV) models for the performance control of Web services adopting DVS as control variable have been proposed.

On the other hand, virtualization technologies allow reducing the number of physical servers – and thus the associated energy costs of the overall service center – and enable the dynamic reconfiguration of the physical infrastructure by, e.g., migrating running virtual machines (VMs) between servers or turning some servers off when not needed. With server virtualization, physical resources (e.g., CPU and disks) are partitioned into multiple virtual ones, creating isolated VMs each running at a fraction of the physical system capacity. A virtual machine monitor (VMM), like Xen or VMWare is installed on each physical server and manages physical resources, providing service differentiation and performance isolation for multiple VMs. Usually, each VM is dedicated to a single Web service and has access to a time-varying fraction of the physical server capacity, depending on the Web service QoS requirements and the current workload conditions. However, virtualization introduces new problems related to resource multiplexing and scheduling of multiple VMs. The performance modelling of virtualized environments is a challenging task, as the impact of the choice of the VMM scheduler, its parameters and I/O management overhead is still only partially understood (see [4]). Note, in passing, that autonomic management today is implemented also in virtualization products (e.g., VMWare DR5 or Virtuoso VSched), which have the aim of equalizing the use of resources but cannot currently provide any performance guarantee.

To model such complex systems, in [5] optimization techniques which can determine the VMM scheduling parameters and VMs placement on physical servers have been proposed. Anyway, a queuing network model of physical servers is adopted and VM resource contentions issues are neglected. Furthermore, queuing network models usually assume that the system is at steady state. Hence, this type of approach is effective only on a long time horizon, e.g., half an hour. Vice versa, to implement genuine control-theoretic approaches, one needs to accurately model the system transients and adjust the system configuration within a very short time frame (e.g., minutes).

In this context, the aim of this work is to derive LPV models for the dynamics of a web server in a virtualized environment. Specifically, we propose to employ a multirate scheme in order to work out a new type of LPV model for the considered application, capable of describing the input-output system behaviour at a slow rate, while retaining parameter dependence at a fast rate. This is motivated by the fact that in [2] and [3] the best modeling performances were achieved with a very fast sampling time, i.e., few
seconds. However, such a sampling time is even too short for control purposes, whose expected appropriate time scale is one or few minutes. As such, to exploit the fast parameters variability which helps to better explain fast transients in the response time while working at the appropriate time scale for control purposes, the multirate LPV model class is proposed.

The paper is organised as follows. Section II states the problem setting and the LPV multirate modeling rationale. Section III illustrates the LPV identification algorithms. The LPV multirate model class is presented in Section IV, whereas Section V discusses the experimental results.

II. PROBLEM STATEMENT

This work considers CPU-bounded Web service applications which share a virtualized computing platform. In the paper, the following notation is adopted.

- $\Delta t$ denotes the sampling interval;
- $k$ is the discrete time index over the interval $[k\Delta t, (k+1)\Delta t]$;
- $\lambda^i_k$ denotes the requests arrival rate for Web service application $i$ in the $k$-th time interval;
- $s^i_k$ is the requests service time, i.e., overall CPU time needed to process a request for application $i$ in the $k$-th time interval;
- $T^i_k$ is the average server response time, i.e., the overall time a request stays in the system for application $i$ in the $k$-th time interval;
- $\phi^i_k$ represents the VMM scheduling parameter, i.e., the fraction of capacity devoted for executing the VM which hosts application $i$ in the $k$-th time interval.

The VMM allocates CPU to competing VMs proportionally to the weights $\phi^i_k$ that each VM has been assigned in the $k$-th time interval. For example, if two VMs have been assigned the same fraction of capacity, they are guaranteed to receive 50% of the CPU each. Furthermore, in this work the VMM has been configured to support work-conserving mode, [4]; this means that if one of the two aforementioned VMs is blocked for an I/O operation (or simply because no requests are running in the system) the other one has access to the entire CPU. VMM scheduling parameters can be updated by using VMM commands which are actuated in few milliseconds without introducing any system overhead. Hence, the VMM scheduling parameters $\phi^i_k$ can be adopted effectively as control variables to adapt the application performance to the incoming workload variations on very fine-grained time scales. Finally, note that the system is configured – as is the usual choice in this context – so that $\sum_{i=1}^{n} \phi^i_k = 1, \forall k$, i.e., the entire system capacity is devoted to application execution, [6].

In this context, assuming that $n$ applications are present due to the machine virtualization, the available control variables are the $n-1$ VMM scheduling parameters, i.e., $u^i_k = \phi^i_k, i = 1, \ldots, n-1$. In fact, due to the constraint $\sum_{i=1}^{n} \phi^i_k = 1, \forall k$, only $n-1$ VMM scheduling parameters are linearly independent. Moreover, if the server is endowed with DVS capabilities, we can assume that $s^i_k$ is inversely proportional to the physical server CPU frequency. This means that the effect of – say – lowering the CPU frequency when a light workload is present in the system causes an increase of the effective CPU time needed to serve a request. This assumption is supported by current technology trends, since in modern systems (e.g., AMD Operon 2347HE Barcelona core) CPUs and RAM clock can be scaled independently. Thus, denoting with $f_k$ the ratio of the frequency applied during the time interval $k$ to the physical server maximum frequency, the effective service time for each application can be defined as $s^i_{f,k} = s^i_k/f_k$. However, as the CPU frequency will be the same for all VMs, the resulting control variables will still be $n-1$, in this case defined as $u^i_k = \phi^i_k f_k, i = 1, \ldots, n-1$.

Further, as incoming requests – independently of the application they may try to access – wait in a queue before accessing the physical server, the system dynamics have a feedthrough term. Thus, the response time $T^i_k$ is given by

$$T^i_k = s^i_k + \xi^i_k,$$

where $\xi^i_k$ is the waiting time, i.e., the time that the request spends in the queue, [7]. As such, the real dynamics of the system are to be found in the variable $\xi^i_k = T^i_k - s^i_k$. In what follows, MIMO LPV models will be identified for the dynamics of $\xi^i_k$ only, and the final response times for each application will then be retrieved by means of (1).

As previously mentioned, the final aim is to adopt a multirate LPV model class for describing the dynamics of the considered virtualized system, capable of describing the input-output system behaviour at a slow rate, while retaining parameter dependence at a fast rate.

Specifically, to exploit the fast parameters variability which helps to better explain fast transients in the response time while working at the appropriate time scale for control purposes, we aim at realising the modeling scheme depicted in Figure 1, where the real virtualized system $S$ has input $u^i_k = \phi^i_k, i = 1, \ldots, n-1$, output $y^i_k = T^i_k$ and is dependent on the workload parameters $p(r)$. Note that, due to the stochastic nature of the workload, the arrival times of the single requests $i_{req}^u$ do not uniformly span the sampling interval, but can be modeled as samples of an exponential distribution with parameter $\gamma$, i.e., $i_{req}^u \sim \mathcal{E}(\gamma)$. As such, the real system dynamics can be captured by a multirate LPV model with input $u^i_k$ and output $y^i_k$, with $u^i_k$ representing the scheduling decisions made by the VMM and $y^i_k$ giving the actual computational workload generated by application $i$. The multirate LPV model class for describing the dynamics of the real virtualized system is introduced in the next section.
to identify an LPV model the measured parameters and the system output need to be re-sampled via time averaging by post-processing the measured data according to the time stamps of each actual request. Thus, to construct a multirate LPV model we intend to consider inputs and outputs sampled at the control-oriented sampling time of 1 min ($T_1$ in Figure 1), whereas the fast parameters variability is maintained by sampling the requests rate and the respective service times with a faster sampling interval of 10 s ($T_2$ in Figure 1). In so doing, we expect to achieve better identified models with respect to those obtained by sampling both input and output and the parameters at the same slow time scale of 1 min to be used for control design.

III. LPV STATE SPACE MODELLING AND IDENTIFICATION

LPV systems are linear time-varying plants whose state space matrices are fixed functions of some vector of varying parameters. LPV model identification algorithms are available in the literature both for input/output and state space representations of parametrically-varying dynamics. In particular, in the recent works [8], [9] an input-output modelling approach was adopted. If, however, the aim of the identification procedure is to eventually work out LPV models in state space form for control design purposes, one should keep in mind that the usual equivalence notions applicable to LTI systems cannot be directly used in converting LPV models from input-output to state space form, as the time-variability of LPV systems ought to be taken into account, [10]. Bearing this in mind, in this work we focus on state space LPV models in the form

$$
x_{k+1} = A(p_k)x_k + B(p_k)u_k,
$$

$$
y_k = C(p_k)x_k + D(p_k)u_k,
$$

where $p \in \mathbb{R}^s$ is the parameter vector and $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $y \in \mathbb{R}^l$. It is often necessary to introduce additional assumptions regarding the way in which $p_k$ enters the system matrices: in this work we focus on affine and input-affine models, defined as follows: i) Affine parameter dependence (LPV-A):

$$
A(p_k) = A_0 + A_1p_{1,k} + \ldots + A_sp_{s,k}
$$

and similarly for $B$, $C$ and $D$, and where by $p_{i,k}, i = 1, \ldots, s$ we denote the i-th component of vector $p_k$. This form can be immediately generalised to polynomial parameter dependence; ii) Input-affine parameter dependence (LPV-IA): this is a particular case of the LPV-A parameter dependence in which only the $B$ and $D$ matrices are considered as parametrically-varying, while $A$ and $C$ are assumed to be constant, i.e., $A = A_0$, $C = C_0$.

As far as LPV model identification is concerned, it is usually convenient to consider first the simplest form, i.e., the LPV-IA one, as its parameters can be retrieved using Subspace Model Identification (SMI) algorithms for LTI systems by suitably extending the input vector. In this work the MOESP class of SMI algorithms has been considered, [11]. LPV-IA models also provide a useful initial guess for iterative methods which can be used for the identification of fully parameterised models in LPV-A form, along the lines of [12]. The classical way to perform linear system identification is by minimizing the error between the real output and the predicted output of the model. A similar approach can be used for LPV state-space systems of the form (2). Letting the system matrices of (2) be completely described by a set of parameters $\theta$, identification can be carried out by minimizing the cost function

$$
V_N(\theta) := \sum_{k=1}^{N} ||y_k - \hat{y}_k(\theta)||^2_2 = E^T_N(\theta)E_N(\theta),
$$

with respect to $\theta$, where

$$
E^T_N(\theta) = \begin{bmatrix} (y_1 - \hat{y}_1(\theta))^T & \ldots & (y_N - \hat{y}_N(\theta))^T \end{bmatrix},
$$

$y_k$ denotes the measured output and $\hat{y}_k(\theta)$ denotes the output of the LPV model to be identified.

IV. MULTIRATE LPV MODELS

Multirate schemes arise in digital control and digital signal processing whenever it is necessary to sample the outputs and/or update the inputs with different rates. As is well known (see, e.g., [13], [14]), multirate sampled-data mechanisms applied to linear time-invariant systems generate periodicity. In this work, as mentioned in Section II, the goal is to use a multirate scheme in order to work out a new type of LPV model for the considered application, capable of describing the input-output system behaviour at a slow rate, while retaining parameter dependence at a fast rate (see also Figure 1).

To this aim, we concentrate on LPV-IA systems, which proved to be the best suited for the considered application, see [2], [3]. Hence, we need to obtain an LPV multirate model, with slow inputs $\hat{u}_k$ and slow outputs $\hat{y}_k$ – piecewise constant over $N$ sampling intervals – and fast parameters $p_k$.

First of all, starting from a discrete-time fast model, we need to introduce the hold device, represented by the holding selector matrix

$$
S_k = \begin{cases} 0 & \text{if } (k \mod N) = 0 \\ I & \text{else}. \end{cases}
$$

To express the state and output equation of the original LPV-IA fast model with respect to the slow input and output variables, we need to augment the state vector to include the holding mechanism. Hence, the system state becomes $\bar{x}_k = [x^T_k, v^T_k]^T$ and the resulting state equation yields

$$
\bar{x}_{k+1} = \bar{A}(p_k)\bar{x}_k + \bar{B}(p_k)\hat{u}_k,
$$

where

$$
\bar{A}(p_k) = \begin{bmatrix} A & B(p_k)S_k \\ 0 & S_k \end{bmatrix}, \bar{B}(p_k) = \begin{bmatrix} B(p_k)(I - S_k) \\ I - S_k \end{bmatrix}.
$$

Similarly, the output equation can be written as

$$
y_k = \bar{C}(p_k)\bar{x}_k + \bar{D}(p_k)\hat{u}_k,
$$

where

$$
\bar{C}(p_k) = [C \ D(p_k)S_k], \bar{D}(p_k) = [D(p_k)(I - S_k)].
$$
Lengthy calculations show that the overall input-output behaviour of the system can be written as
\[ y_{k+h} = \bar{C}(p_{k+h}) \prod_{l=0}^{h-1} \bar{A}(p_{l+k}) \bar{x}_k + \bar{C}(p_{k+h}) \times \]
\[ \times \sum_{l=0}^{h-1} \prod_{j=k+l+1}^{k+h-1} \bar{A}(p_j) \bar{B}(p_{k+l}) \bar{u}_{k+l} + \bar{D}(p_{k+h}) \bar{u}_{k+h}, \]
for \( h \in [0, N] \). The above expression can be significantly simplified by noting that the state space matrices take peculiar explicit forms according to the value of the holding selector. Namely, when \( S_k = 0 \) one gets
\[ \bar{A}_{S_0} = \begin{bmatrix} A & 0 \\ 0 & 0 \end{bmatrix} \quad \bar{B}_{S_0}(p_k) = \begin{bmatrix} B(p_k) \\ I \end{bmatrix}, \]
\[ \bar{C}_{S_0} = \begin{bmatrix} C & 0 \end{bmatrix} \quad \bar{D}_{S_0}(p_k) = \begin{bmatrix} D(p_k) \end{bmatrix}, \]
whereas for \( S_k = 1 \) one finds
\[ \bar{A}_{S_1} = \begin{bmatrix} A & B(p_k) \\ 0 & I \end{bmatrix} \quad \bar{B}_{S_1} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \]
\[ \bar{C}_{S_1}(p_k) = \begin{bmatrix} C & D(p_k) \end{bmatrix} \quad \bar{D}_{S_1} = \begin{bmatrix} 0 \end{bmatrix}. \]
Evaluating (8) in the light of the above matrix expressions, the free and forced motion of the fast output take the form
\[ y_{r,N+k}^{F} = \left[ \begin{array}{c} CA^h \\ 0 \end{array} \right] \bar{x}_{r,N} \]
\[ y_{r,N+h}^{F,0} = \sum_{s=0}^{h-1} \left[ \begin{array}{c} \sum_{n=1}^{N-1} CA^s B(p_{r,N+s-1} + D(p_{r,N+h}) \bar{u}_{r,N}. \right. \left. \quad (13) \right. \]
\[ (14) \]
Finally, the slow output is computed by averaging the fast one over a period of \( N \) samples, namely
\[ \bar{y}_{r,N} = \frac{1}{N} \sum_{h=0}^{N-1} y_{r,N+h} = \frac{1}{N} \sum_{h=0}^{N-1} y_{r,N+h}^{F} + y_{r,N+h}^{F,0}. \]
By inspecting (13)-(15) it can be easily recognised that this model relates the slow input and output while taking into account the fast variation of the parameter, as desired.

V. EXPERIMENTAL RESULTS
We now present the experimental set-up and illustrate the experiments designed for collecting the data both for identification and validation purposes. Further, the obtained results are discussed and evaluated by means of quantitative performance metrics.

A. Testbed setting and experiment design
In the experimental framework, a workload generator and a micro-benchmarking Web service application have been used. The workload generator is based on a custom extension of the Apache JMeter 2.3.1 workload injector, which allows to generate workload according to an open model with a Poisson arrival process. The micro-benchmarking Web service application is hosted within the Apache Tomcat 6.0 application server. The Web service is a Java servlet designed to consume a fixed CPU time. The application has been instrumented to accurately determine the service time of each request. Note that this is not a limitation as there exist several techniques to assess the number of CPU cycles consumed by requests both at application level or at operating system level. The employed VMM monitor is Xen 3.0, which runs on an AMD Operon 2347HE Barcelona dual socket quad-core system. Up to four instances of the micro-benchmarking Web service applications have been considered hosted in Linux Fedora 8 VMs. The VMM monitor is assigned to a dedicated core, while the VMs are assigned to the same core by setting CPU affinity.

Both identification and validation experiments have been carried out for two, three and four VMs, respectively. For system identification purposes, incoming request rates \( \lambda_i, i = 1, \ldots, n \), accessing each of the \( n \) VMs vary stepwise every 1 minute, with values between 0.1 req/s and 1.2 req/s.

During each time interval, each request \( \lambda_i \) consumes a fixed deterministic service time \( s_i \) varied between 0.06 s and 1.1 s. Overall, 1,440 intervals (24 hours) have been considered and the average utilization of the physical server (defined as \( \rho_k = \sum_{i=1}^{n} \lambda_i s_i k \) ) has been varied from 0.05 to 1 in order to analyze the behavior of the physical server under both light load and saturation conditions. Finally, the VMM scheduling parameter \( \phi_i \), i.e., the fraction of capacity devoted for executing the VM which hosts the \( i \)-th application, has been selected as a realization of a uniform random variable with values between 0.1 and 0.9. Note that the \( n \)-th application is then forced to have access to a capacity of \( \phi_i = 1 - \sum_{i=1}^{n-1} \phi_i \).

Fig. 2. Time history of the request rates applied during a validation test with four VMs.

For model validation, a synthetic workload inspired by a real-world usage has been employed. The incoming workload reproduces a 24 hour trace obtained from an on-line banking application. The workload injector is configured to follow a Poisson process with request rate changing every minute according to the trace. Figure 2 reports the request rates applied to the four VMs case during a validation test, which follows a bimodal distribution with two peaks around 11.00 and 16.00. The application service time follows a log-normal distribution such that, in each 1 minute time interval, the standard deviation is four times larger than the average service time, [15].
B. Experimental results: LPV single-rate models

Based on the collected identification data, LPV-IA models for the virtualized server dynamics have been identified. Specifically, the VMM parameters are used as inputs, i.e., $u_k = [\phi_k^i], i = 1, \ldots, n - 1$, whereas the service times $s_k^i$ and each application utilization $\rho_k^i = s_k^i \lambda_k^i$ are employed as scheduling parameters, i.e., $p_k = [s_k^i, \rho_k^i], i = 1, \ldots, n$. The system outputs are the service response times obtained for each application, i.e., $T_k^i, i = 1, \ldots, n$. The model order – after some comparative analysis – was set to 2 for the case of two VMs and to 4 for the cases of three and four VMs, and the sampling time set to $\Delta t = 1$ min.

Based on the discussion in Section II, an LPV-IA model for the dynamics of $\xi_k^i, i = 1, \ldots, n$ was estimated, and the overall response times computed according to Equation (1).

To quantitatively evaluate the models, both on identification and validation data, two metrics based on the results obtained in simulation (note that the focus on prediction would not be appropriate as the aim is to evaluate control-oriented models) will be considered: the percentage Variance Accounted For (VAF), defined as

$$VAF = 100 \left(1 - \frac{Var[y_k - y_{sim,k}]}{Var[y(k)]}\right),$$

where $y(t)$ is the measured signal and $y_{sim,k}$ is the output obtained by the identified model and the percentage average error $e_{avg}$, computed as

$$e_{avg} = 100 \left| \frac{E_{\xi}[y_k - y_{sim,k}]}{E_{\xi}[y_k]} \right|.$$  

This choice allows us to assess the system performance both in a 1- and in a 2-norm sense.

Figure 2), the light load data are those in the time interval $t \in [0, 8.30) \cup (13.30, 14.45) \cup (17.30, 24]$ h, while the heavy load data in the time interval $t \in [8.30, 13.30) \cup [14.45, 17.30]$ h. We are now interested in comparing such performances with those obtained with the proposed multirate LPV model class.

C. Experimental results: LPV multirate models

To assess the performance on the LPV multirate models, we first of all derived LPV single-rate models at the fast time scale, i.e., with $\Delta t = 10$ s. Note that, as in both identification and validation tests the VMM scheduling parameters $\phi_k^i$ were actually varied every minute, for fast models identification the inputs are kept piecewise constant over six time intervals, whereas the measured workload parameters and response times are resampled at $\Delta t = 10$ s by suitable averaging of the measured data record relative to each single request (which contains the actual arrival time of the request and the associated service and response times). Once the fast models have been obtained, they have been embedded in the appropriate multirate LPV model according to the rationale described in Section IV.

Specifically, the free and forced motion of the fast output as functions of the slow input have been computed according to Equations (13) and (14) and the overall slow system output according to (15).

To inspect the achieved modeling performance, Figure 4 shows a detail of the response time obtained with the single-rate LPV model and with the multirate LPV model in a validation experiment with three VMs. Figure 4 shows that the multirate LPV models more accurately explain the peaks in the response time, as it was expected by exploiting a fast parameters variation. Finally, Table II reports the detailed validation performance – in both metrics – for the multirate LPV identified models (the data partitioning into light and heavy load is the same as described for the single-rate case). Comparing the results in Table I and Table II, note

![Fig. 3](image-url)  
Fig. 3. Detail of the measured (solid line) and simulated (dashed line) response time obtained with an LPV-IA model for the first (top plot) and for the second (bottom plot) application in a 2-VMs test on validation data at high load.

![Fig. 4](image-url)  
Fig. 4. Detail of the measured (solid line), simulated with the single-rate LPV model (dashed line) and simulated with the multirate LPV model (dash-dotted line) response time obtained with an LPV-IA model in a 3-VMs test on validation data.
that the multirate LPV models consistently provide better performance – especially in terms of VAF which is the most important indicator for control design purposes – than those obtained with the LPV single-rate ones. Further, it is worth pointing out that the multirate LPV models provide a more equalized modeling performance among the different VMs, which is more significant as the number of VMs increases.

VI. CONCLUDING REMARKS

This paper presented the control-oriented multirate modeling of LPV systems for performance control of web servers in virtualized environments. Specifically, MIMO LPV models have been estimated both at slow and fast time scales. Further, the fast LPV model has been embedded into an LPV multirate scheme to exploit the fast parameters variability and better account for system transients while producing a final model with inputs and outputs evolving at a slow time scale appropriate for control purposes. The suitability of the proposed multirate LPV model class for the problem at hand has been favorably witnessed by experimental data analysis.

Future research directions will address both customised identification algorithms for the multirate LPV model class and the use of the proposed multirate models for controller synthesis.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge Paolo Sala and Mauro Speroni for their help in the test activities and Dr. Massimo Leoni of IBM Italy for providing real applications traces.

REFERENCES


TABLE I

<table>
<thead>
<tr>
<th>2 VMs</th>
<th>VAF on 24h</th>
<th>VAF light-load</th>
<th>VAF heavy-load</th>
<th>eavg on 24h</th>
<th>eavg light-load</th>
<th>eavg heavy-load</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM1</td>
<td>82.21%</td>
<td>82.8%</td>
<td>82.48%</td>
<td>5.46%</td>
<td>1.43%</td>
<td>13.4%</td>
</tr>
<tr>
<td>VM2</td>
<td>73.41%</td>
<td>72.49%</td>
<td>82.22%</td>
<td>2.63%</td>
<td>6.42%</td>
<td>13.18%</td>
</tr>
</tbody>
</table>

TABLE II

<table>
<thead>
<tr>
<th>2 VMs</th>
<th>VAF on 24h</th>
<th>VAF light-load</th>
<th>VAF heavy-load</th>
<th>eavg on 24h</th>
<th>eavg light-load</th>
<th>eavg heavy-load</th>
</tr>
</thead>
<tbody>
<tr>
<td>VM1</td>
<td>89.73%</td>
<td>89.31%</td>
<td>91.01%</td>
<td>8.3%</td>
<td>6.99%</td>
<td>9.77%</td>
</tr>
<tr>
<td>VM2</td>
<td>78.66%</td>
<td>77.46%</td>
<td>86.68%</td>
<td>2.7%</td>
<td>0.96%</td>
<td>4.73%</td>
</tr>
</tbody>
</table>