Evaluation of different scheduling policies in IaaS applications by Mean Field analysis

Danilo Abbaldo
Dipartimento di Informatica
University of Torino
Corso Svizzera 185
Torino, Italy
Email: dnl.abboz@gmail.com

Marco Gribaudo
Dipartimento di Elettronica e Informazione
Polytechnic of Milano
Via Ponzio 34
Milano, Italy
Email: gribaudo@elet.polimi.it

Daniele Manini
Dipartimento di Informatica
University of Torino
Corso Svizzera 185
Torino, Italy
Email: manini@di.unito.it

***Abstract*—Cloud Computing is emerging today as a commercial infrastructure that through the use of virtualization aims to provide on demand computing resources. In particular, the Infrastructure as a Service (IaaS) is a cloud service that allows the user to perform and execute any OS and application in virtual environments. In this work we present an evaluation of different policies for the assignment of virtual machines that IaaS providers can adopt in order to both efficiently planning their infrastructures and guaranteeing the quality of service expected by customers. The study is based on the exploitation of a Mean Field Model, able to evaluate systems with a relevant number of interacting entities, that can provide interesting insights derived from the computation of different performance indexes such as the request loss rate, the mean number of executing/available resources, and the overall ratio of satisfied requests to mention a few.**

***Index Terms*—Mean Field Analysis, Performance Evaluation of Communication Networks, Cloud Computing**

I. Introduction

Cloud Computing is a commercial infrastructure that through the use of virtualization aims to provide on demand computing resources. The Infrastructure as a Service (IaaS) is a cloud service that allows the user to perform and execute any OS and application in virtual environments. The most known commercial examples available nowadays are the Amazon Elastic Compute Cloud (EC2), and the IBM Smartcloud.
In this work we evaluate different policies for the assignment of virtual machines that IAAS providers can adopt in order to both efficiently plan their infrastructures and supply the quality of service expected by customers. The correlation between the resources allocated and the performance offered is influenced by a number of factors such as the characteristic of the different class of requests, the capacity of the resources, and the workload sharing the same physical hardware.

The study is based on the exploitation of a Mean Field Model that is able to describe systems with a relevant number of interacting entities (for instance physical and virtual resources), and to compute the time evolution of all the entities forming the cloud system. Thanks to the underlying solution technique the resulting model can provide interesting insights derived from the computation of different performance indexes such as the request loss rate, the mean number of executing/available resources, and the overall ratio of satisfied requests to mention a few.
In the literature, there are several works that deal with the optimal allocation of resources in virtual environments. Several techniques and models focus on database consolidation, some like in [8] by means of workload monitoring for load balancing, others like in [10] using data migration and task scheduling. Other techniques, as in [2][3][9] are aimed to maintain acceptable application performance levels while minimizing the costs of migration/consolidation of resources. Many works propose different approaches to enable autonomic controller to satisfy service level objectives by dynamically provisioning resources, such as [6][12] [13]. In particular, in [5] the dynamic allocation of VMs in cloud environment is described.

The technique proposed in this paper is different from the one presented in [2] that uses queueing network to represent the studied systems, since our work exploits the object oriented representation to describe all the elements forming the systems, e.g., virtual machines, physical machines, and clients. Moreover, thanks to the underlying solution technique it is possible to perform the analysis for systems composed of a huge number of interacting objects, providing more accurate results as the number of component increases. In [12] the analysis is focused on CPU allocation and on the application response time, whereas in our work we take into account OS and application virtualizations to evaluate the load request response and the infrastructure efficiency by studying different ad-hoc performance indexes. In this paper we apply the same methodology used in [7] to model biological pathways to evaluate different scheduling policies in IaaS cloud applications.

The paper is organized as follows: in Section II we review the basic concepts of Mean Field Analysis, Section III presents the model and its validation, and in Section IV results from the experiments are reported.

II. Methodology

Current data-centers that provision cloud services are usually characterized by several hundreds (if not even thousands) of machines. To properly model such systems, we require a formal technique capable of dealing with such dimensions. The

VALUETOOLS 2012, October 09-12, Cargèse, France
Copyright © 2012 ICST
DOI 10.4108/valuetools.2012.250285

Mean Field Technique [11] is a good choice in this direction. An *Object-oriented Mean Field Model*, is a representation that describes the behavior of a system as a net composed of a large number of interacting objects. Objects are divided into *classes*: all the objects belonging to a given class have exactly the same behavior. Objects might be influenced by the distribution of the other objects in the system. Each object is modeled by a CTMC, whose transition rates may depend on the state of the whole system. All the objects that belong to the same class are characterized by exactly the same infinitesimal generator and the same parameters. If two objects perform the same actions at different rates, they must belong to different classes. In order to ease the description of complex systems, classes are further grouped into *meta-classes*. All the classes that derive from the same meta-class are characterized by the same structure, but different rates.

The number of objects in every class changes dynamically: new objects might be formed at a given rate (expressed as quantity of new objects created per unit of time), and each object has an exponentially distributed maximum lifetime. In the following we report the formal description taken from [7].

We call an *Object-oriented Mean Field Model* $\mathcal{M}$, a tuple:

$$\mathcal{M} = (MC, OC), \tag{1}$$

where $MC = \{mc^{(1)}, \ldots, mc^{(k)}\}$ is a set of k *meta-classes* and $OC = \{oc^{[1]}, \ldots, oc^{[m]}\}$ is a set of m *object classes*. Each meta-class $mc^{(i)}$ is in turn defined by a tuple:

$$mc^{(i)} = (c^{(i)}, n^{(i)}, L^{(i)}, \Lambda^{(i)}, \mathbf{C}^{(i)}), \tag{2}$$

where $c^{(i)}$ is a label corresponding to the name of the meta-class, $n^{(i)}$ is the number of states of the CTMC, $L^{(i)} = \{l^{(i)}\}$ is a set of labels (the names of the states) and $\Lambda^{(i)} = \{\lambda_1^{(i)}, \ldots, \lambda_{p_i}^{(i)}\}$ is the set of *formal parameters*. $\mathbf{C}_i = |c_{ul}^{(i)}|$ is the $n^{(i)} \times n^{(i)}$ infinitesimal generator of the CTMC where c_{ul} is the transition rate from state u to state l. The entries of $\mathbf{C}^{(i)}$ may depend on the actual values assigned to the parameters $\Lambda^{(i)}$. An object class $oc^{[j]}$ is also a tuple:

$$oc^{[j]} = (o^{[j]}, c^{[j]}, \Gamma^{[j]}, N^{[j]}, \pi_0^{[j]}), \tag{3}$$

where $o^{[j]}$ is a label representing the name of the class; $c^{[j]}$ is name of the meta-class from which the class derives; $\Gamma^{[j]} = \{\gamma_1^{[j]}, \ldots, \gamma_{p_i}^{[j]}\}$ is the set of *actual parameters* assigned to each of the formal parameters of the meta-class defined by $\Lambda^{(i)}$; $N^{[j]}$ is the initial number of objects; $\pi_0^{[j]}$ is a probability vector of size $n^{[j]}$ that defines the initial state probability for the objects belonging to this class. We define $n^{[j]}$ as the number of states of class j inherited from its meta-class, that is $n^{[j]} = n^{(meta-class \quad of \quad j)}$. The value of each actual parameter can depend on the distribution of the number of objects among the states of all the classes that compose the model. As a notation, we use round brackets in superscripts for elements corresponding to meta-classes and square brackets to denote elements belonging to classes.

A. Analysis

The model is analyzed using *mean field analysis* [1] which takes advantage of the result proposed in [4] to consider the evolution of each class separately. Initially, object classes are instantiated: matrix $\mathbf{C}^{[j]}(\cdot)$ is computed for each $oc^{[j]}$ by inserting the actual parameters $\Gamma^{[j]}$ in the definitions of $\mathbf{C}^{(i)}$. We call $\mathbf{N}^{[j]}(t) = |N_l^{[j]}(t)|$ a vector of size $n^{[j]}$, whose element $N_l^{[j]}(t)$ represents the number of objects of class j in state l at time t. Formal parameters can depend on the number of objects in each state, and thus we have: $\mathbf{C}^{[j]}(\mathbf{N}^{[1]}(t), \ldots, \mathbf{N}^{[m]}(t))$.

The evolution of the system can then be studied solving for $j = 1..m$:

$$\frac{d\mathbf{N}^{[j]}(t)}{dt} = \mathbf{N}^{[j]}(t)\mathbf{C}^{[j]}(\cdot), \tag{4}$$

with $\mathbf{N}^{[j]}(0) = N^{[j]}\pi_0^{[j]}$. The derivation of Eq. (4) can be summarized as follows. To simplify the presentation we drop the $^{[j]}$ superscript and the state dependencies $(\cdot)$. The number of objects of class j in state l at time $t + \Delta t$ can be approximated by:

$$\begin{aligned} N_l(t+\Delta t) \quad \approx \quad & N_l(t) + \sum_{u \neq l} N_u(t) c_{ul} \Delta t \\ & - N_l(t) \sum_{u \neq l} c_{lu} \Delta t. \end{aligned} \tag{5}$$

The second and third terms on the r.h.s. of Eq. (5) represent objects entering and leaving state l. By applying the definition $c_{ll} = -\sum_{u \neq l} c_{lu}$, rearranging the terms, and dividing by Δt we obtain:

$$\frac{N_l(t+\Delta t) - N_l(t)}{\Delta t} \approx \sum_u N_u(t) c_{ul}. \tag{6}$$

Eq. (4) can be obtained by letting $\Delta t \to 0$, and using vector notation. The experiments presented in this paper compute the solution of Eq. (4) with the Runge-Kutta method with adaptive step size discretization.

III. The IaaS model

We developed a model to describe resource allocation in IaaS infrastructures: services that allow the users to perform OS and applications in virtual environments. We consider a scenario where the user requires the execution of virtual machines (VMs). The provider assign such requests to physical machines (PMs) to satisfy the user requests.

The development of an object oriented model that exploits the Mean Field Analysis presented in Section II, permits us to evaluate the performance of the IaaS system with respect to the resource allocation policies, as functions of the different classes of requests, the capacity of the resources, and the workload sharing the same physical machines.

The modeled elements are clients, physical machines and the virtual machines hosted by the physical ones. Commercial cloud providers classify machines according to the resources they assign to the users. For example, Amazon classifies its

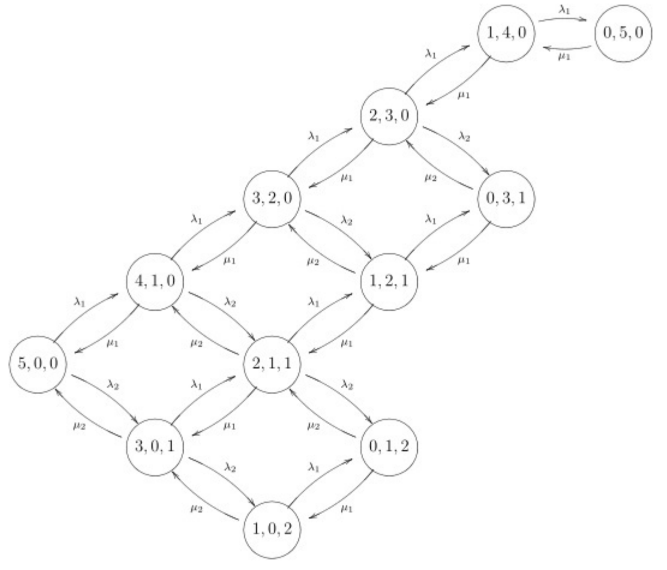


Fig. 1. CTMC of the meta-class *Physical Machine*

virtual machines in terms of ECU (EC2 Compute Unit), that roughly describes the number of cores, the speed of the CPU and the total memory available to the user. In this paper we focus on memory as the feature that characterizes the requests that the user can place. However the proposed methodology can be easily extended to consider other types of resources. In particular, we take into account a set of request types that is pre-defined and equal for all the PMs composing the infrastructure. Without loss of generality, we mainly focus on the maximum amount of memory required by a VM and we measure it in *slots* where each slot can correspond to a fixed amount of RAM (in our examples is 1 Gigabyte).

A. Basic model

The definition of a meta-class, denoted as Physical Machine, provides the general structure that describes the behavior of a physical machine. In this structure, each state is a tuple $(k, n_1, .., n_M)$ where k reports the number of available memory slots, and each n_i denotes the number of virtual machines in execution for each type of virtual machine i, where i represents the associated number of memory slots. The resulting CTMC depends on the maximum number of memory slots available in the physical machine and on the types of virtual machines it provides. Figure I reports the CTMC for a scenario where each PM has $5GB$ memory available and there are two types of virtual machines, with 1 and 2 Gigabytes respectively. Indeed, the CTMC structure is automatically computed by the framework interface for any parameter combination, allowing an easy analysis for different scenarios.

The meta-class describing the scenario of the CTMC reported in Fig. I, can be formalized as follow:
$mc^{(1)} = (\text{'}\textit{Physical Machine}\text{'}, n^{(1)}, L, \Lambda^{(1)}, \mathbf{C}^{(1)})$,

where $n^{(1)} = 12$ states, and:

$L = \{(5,0,0), (4,1,0), (3,0,1), (3,2,0), (2,1,1), (1,0,2), (2,3,0), (1,2,1), (0,1,2), (1,4,0), (0,3,1), (0,5,0)\}$

Other elements are: $\Lambda^{(1)} = \{\lambda_1, \lambda_2, \mu_1, \mu_2\}$, $\mathbf{C}^{(1)} = Q$.

In this case λ_1 and λ_2 represent the inter arrival rate of requests for VMs with 1 and 2 GB of memory respectively, on the other hand μ_1 and μ_2 denote the service rate for the execution of 1 and 2 GB VMs respectively. The infinitesimal generator Q of the CTMC (Figure I), is reported in Table I. This formalization provides the structure of a PM, the number of objects in any CTMC state accounts for the number of the PMs of the whole infrastructure that assume the associated status. Given this representation we can model different types of PM by deriving different classes according to the aspects we would describe.

B. Resource allocation policies

In order to consider physical machines performing different resource allocation policies, we derive a class (from the meta-class) for each case we aim to study. Four policies were selected:

- *Random*: VM requests are assigned randomly among all the PMs, neglecting whether a request is addressed to a machine without available resources.
- *Available First*: it considers only the set of PMs that have available resources, and then it choose randomly from that set.
- *Best Available First*: it considers only the set of PMs that have available resources, and it privileges the choice of the machines with a higher number of available resources.
- *Optimal*: it always assigns the requests to PMs with the lowest load.

By assigning different functions to the rates included in (Λ) it is possible to describe the different policies. In particular, each policy can be modeled by differentiating the state inter arrival rates (λ_i). The formalization of a class will result as:

$$oc^{[1]} = (\text{'}\textit{Policy Name}\text{'}, \text{'}PhysicalMachine\text{'}, \{\lambda_{v1}, \lambda_{v2}, \mu_1, \mu_2\}, N, |1, 0, \ldots, 0|)$$

where the first element defines the name according the policy it models, the second element indicate the meta-class from which it derives. It is followed by the parameter list that determines the behavior of the objects of this class, N stays for the initial number of objects, i.e., the initial number of PM performing the policy identified by this class, and the last vector denotes the initial distribution of the N objects in the CTMC states.

The total arrival rate of requests for given set of resources has been made proportional to the total number of available PMs to consider scalability issues. In the following are reported the functions computing the arrival rates for any case, given that N is the total number of PMs and $N * \lambda_i$ is the total arrival rate for requests of type i:

- *Random*: this policy is modeled by assigning, for any request type i, the same arrival rate λ_i for all CTMC states:

$$f_i(n, v) = \lambda_i, \tag{7}$$

where $n(v)$ is the number of objects in a CTMC state v. The idea behind this equation is that the total arrival rate $\lambda_i \times N$ is equally divided by the N available PMs, and thus each machine receives requests for i resources at rate λ_i.

- *Available First*: the arrival rate is divided for the sum of PMs that can satisfy the request. Let us call R_i the set of states that represents PMs with enough space to satisfy request of i resources. More formally, $R_i = \{v' = (k, n_1, .., n_M) : i \leq k\}$. Then we can compute the arrival rate in state v as:

$$f_i(n, v) = \frac{\lambda_i \times N}{\sum_{v' \in R_i} n(v')}. \tag{8}$$

- *Best Available First*: in this case the sharing algorithm assigns resources with a higher probability to machines with more free resources. To simplify the notation, let us call $K(v)$ the total number of resources available in v (more formally: if $v = (k, n_1, .., n_M)$, then $K(v) = k$). Then we have:

$$f_i(n, v) = \frac{\lambda_i \times N \times K(v)}{\sum_{v' \in R_i} n(v') \times K(v')}. \tag{9}$$

- *Optimal*: the requests are always sent to the machines with the lowest load first, that is machines that have the maximum number of available resources. If there are more machines with the same maximum number of available resources, then the choice is made randomly. Formally

$$\begin{aligned} f_i(n, v) \quad = \quad & \frac{\lambda_i \times N}{\sum_{v' : K(v') = K(v)} n(v')} \times \\ & \mathbf{1}\left(\forall v' : K(v') < K(v) \Rightarrow n(v') = 0\right), \end{aligned} \tag{10}$$

where $\mathbf{1}(x)$ is the indicator function that returns 1 if argument is true, 0 otherwise.

For what concerns the service time, we imagine that the PM executes all their jobs with a processor sharing scheme. If $1/\mu_i$ is the mean service time for machines with i resources, then in a state $v = (k, n_1, .., n_M)$ we have that the completion rate $\mu_i(v)$ for a request of i resources is:

$$\mu_i(v) = \frac{n_i}{\sum_j n_j} \mu_i. \tag{11}$$

In the following, we have supposed that $\mu_i = \mu, \forall i$. In this case, by considering the PM represented in Figure 1, we have that $\mu_1(v)$ and $\mu_2(v)$ are equal to:

$$\mu_1(v) = \frac{n_1}{n_1 + n_2} \mu \ ; \ \mu_2(v) = \frac{n_2}{n_1 + n_2} \mu, \tag{12}$$

respectively.

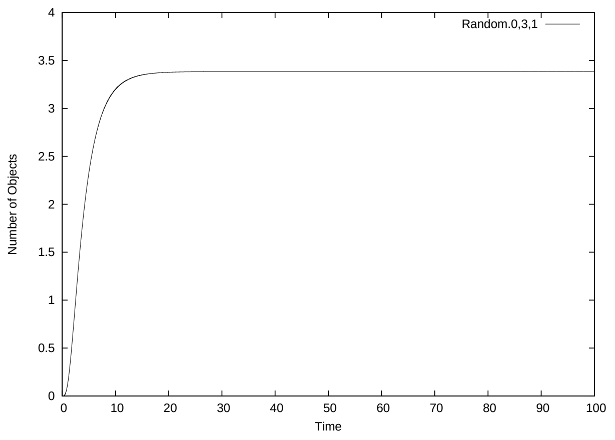


Fig. 2. Time evolution of the number of objects in the state (0,3,1) for Random class

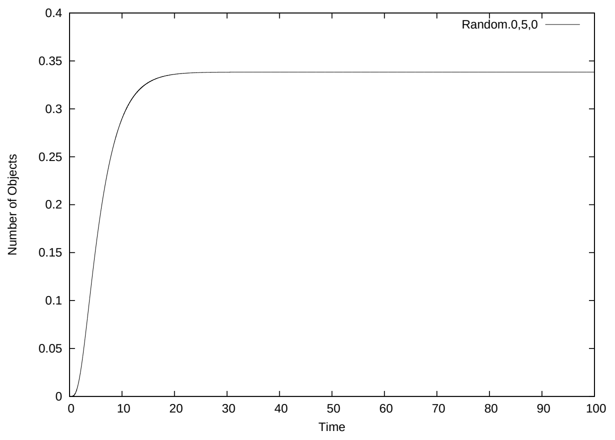


Fig. 3. Time evolution of the number of objects in the state (0,5,0) for Random class

C. Model output and performance indexes

Thanks to the solution technique described in Section II-A the model output computes the time evolution of the number of objects in all the states for each class, i.e., it provides the number of PM's for any possible state. In Figures 2 and 3 are reported the plots for some states of the class Random. Indeed, we exploit the output to compute a set of performance indexes to capture interesting insights related to both the quality perceived by users and the infrastructure planning. We derived the following indexes:

- Mean number of empty PMs (with all resources available);
- Mean number of executing VMs;
- Mean number of PMs without available resources;
- Mean number of available resources (without VM type differentiation);
- Mean number of busy resources;
- Mean overall throughput;
- Request loss rate;
- Mean number of executing VM with j resources;

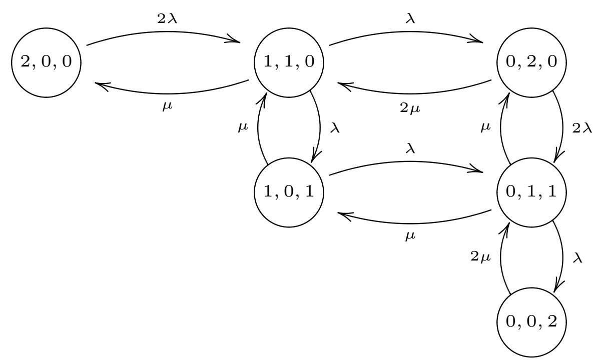


Fig. 4. CTMC for the Random policy, with $M = 2$ PMs, and two resources

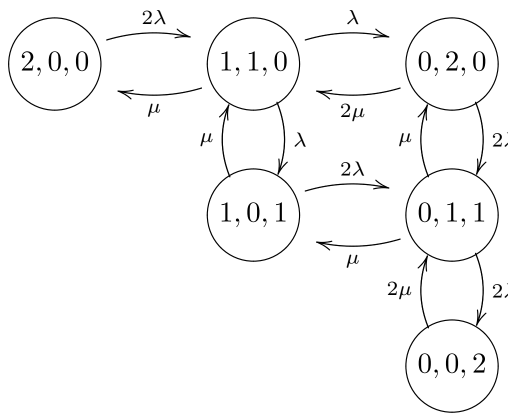


Fig. 5. CTMC for the Available First policy, with $M = 2$ PMs, and two resources

We used these indexes to compare the performance among the considered policies.

D. Model validation

As shown in [4], the Mean Field Analysis technique, which converges to the exact solution of the model as the number of considered objects tends to infinity, can provide very accurate approximations of the exact solution for finite populations. However the quality of the approximation depends a lot on the functions used to describe the evolution of the system population, and on the load of the system. In order to validate the proposed approach, we compared the solution of the mean field model with the one of the exact CTMC for a small number of resources and PMs. In particular we focused on two resources and from two to five PMs. Figure 4 shows the CTMC corresponding to the *Random* policy, and Figure 5 shows the CTMC of the *Available First* policy, both with $M = 2$. In order to reduce the state space, the CTMC describes a *lumped* version of the system: the state of the model is a tuple (c_0, c_1, c_2), where each component c_i corresponds to the number of PMs that have i resources used (and thus, since we consider at most two resources, $2 - i$ available). Both CTMCa are very similar, with the only differences in the *Available First* with respect to the *Random* is in states $(1, 0, 1)$ and $(0, 1, 1)$, where the exit from the state is at rate 2λ instead of λ, and there is no loss until the system is full (state $0, 0, 2$). This small difference is what characterize the policy, since the *Available First* do not send requests to PMs already full. Figure 6 shows the results for $M = 2$ and $M = 5$ PMs, for both the *Random* and *Available First* policies, and for the mean number of assigned resources, and the mean number of full PMs. As we can see, the steady state solution is matched very closely by the mean field model. We can thus conclude that the mean field model is appropriate to study the proposed system.

IV. Results

In this section we discuss the results obtained executing the model of each allocation policy and deriving the performance index described in the previous section. We perform the comparison on some of these indexes for different policies to capture insights about the proposed schemes. The *Mean number of PMs without available resources* and the *Mean number of executing VMs* can be useful for both infrastructure planning and efficiency improvement; the analysis of the quality of service perceived by the clients can be instead estimated computing measures such as the *Mean number of available resources*, and the *Request loss rate*.

A. Policy evaluation as function of the workload

The analysis is performed for a scenario with 100 PMs hosting 5 resources (e.g. 5 GBytes of RAM) each, where clients can request VMs requiring either 1 or 2 resources (e.g. 1 or 2 GBytes of RAM). The overall request arrival rate ranges from 0.1 to 1 requests per day, whereas the mean service time μ is set to 1 day. Figure 7 plots the *Mean number of PMs without available resources*. On one hand it can be noted that the *Available First* policy has the larger number of unavailable PMs as λ increases, on the other hand the *Optimal* one has always PMs available. It is also interesting to remark that from this point of view the *Random* policy outperforms the *Best Available First* when λ become greater than at least 0.87.

The same result is supported by the graph depicted in Figure 8 where is reported the *Mean number of available resources*. Indeed, this indexes is strongly correlated with the *Mean number of PMs without available resources* that is complementary but accounts also for the total number of resources.

The curves of Figure 9 reports the *Mean number of executing VMs*. By looking at this graph together with the one reported in Figure 10 it is possible to find an interesting indication, indeed, while the latter shows that the *Random* policy is the only to lose requests (it assigns request to any machine regardless wether it has resources available), the *Available First* and the *Best Available First* have a mean number of executing VMs that is higher with respect to the Random one (this is always true for the *Available First*, it is true for λ greater than 0.6 in the *Best Available First* case).

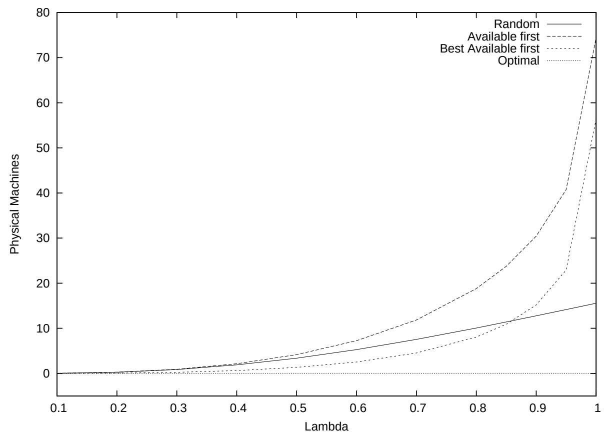


Fig. 7. Mean number of PMs without available resources

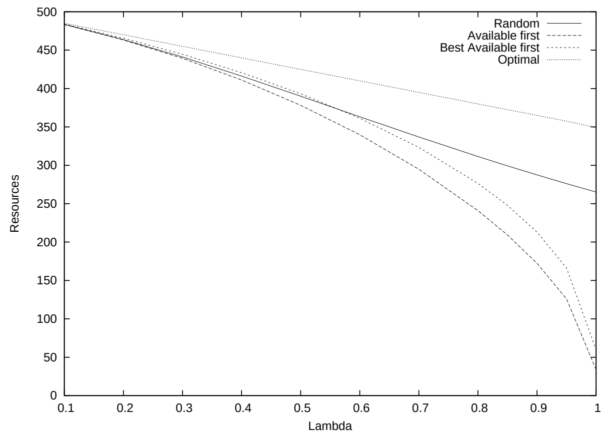


Fig. 8. Mean number of available resources (without VM type differentiation)

The *Optimal* has the lowest value since it selects always the PMs with the lower load and hence it is able to rapidly execute the VMs.

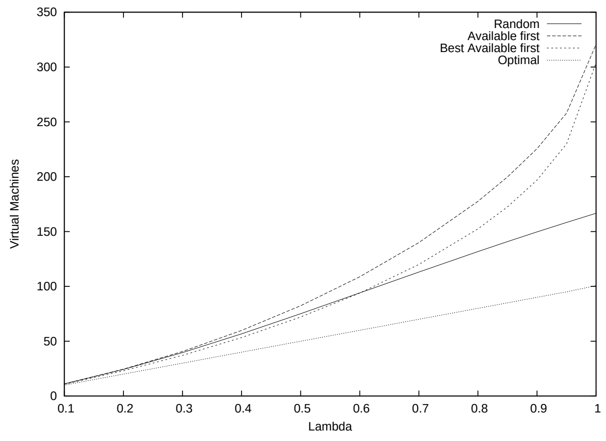


Fig. 9. Mean number of executing VMs

Figure 11 plots the *Mean overall throughput*, the *Random* one decreases due to the losses whereas the others are linear

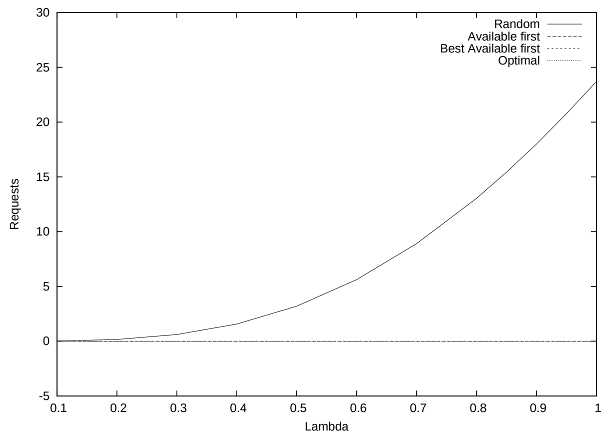


Fig. 10. Request loss rate

and depend on the ratio among the μ and λ parameters.

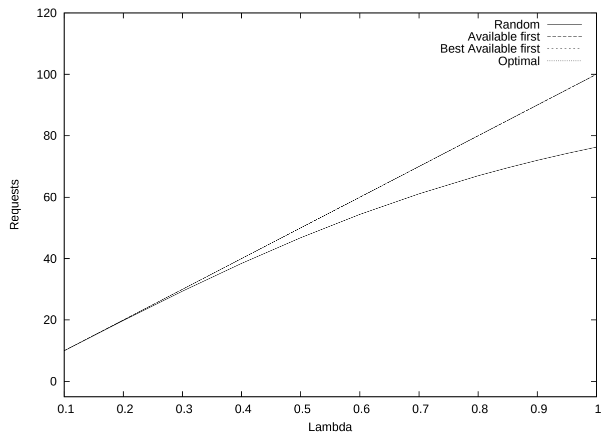


Fig. 11. Mean overall throughput

B. Analysis for different initial numbers of resources

As expected, by increasing the number of available resources for each PM also the policies that do not use the optimal approach improve their performance as illustrated in Figure 12. Moreover, this graph points out that when K is greater or equal to 15 the *Best Available First* has a lower number of executing VMs than the *Random* that tends to behaves like the *Available First* as K increases. The *Optimal* maintains the same performance for all depicted configurations since with these workloads the policy is always able to rapidly finish off the requests.

C. System response to bursts

One of the main advantages of Mean Field analysis, is the possibility of performing transient analysis, We exploit this feature by modulating (with a deterministic time dependent process) the arrival rates to describe *workload bursts*, i.e., relevant amount of VMs requests that arrives in a small time intervals. Burst are thus modeled by increasing the overall inter-arrival rate in fixed time periods. In Figure 13 is reported the system response to bursts for each policy, λ switches from 0.5 to 1.2 and vice versa. Note that during the burst, the arrival rate is much higher than the service rate, and VMs tends to accumulate in the system. During the peak's period the *Mean number of executing VMs* of the *Optimal* is lower with respect to the others, whereas the *Best Available First* has the greatest value. The *Random* and the *Available First* behave at the same way.

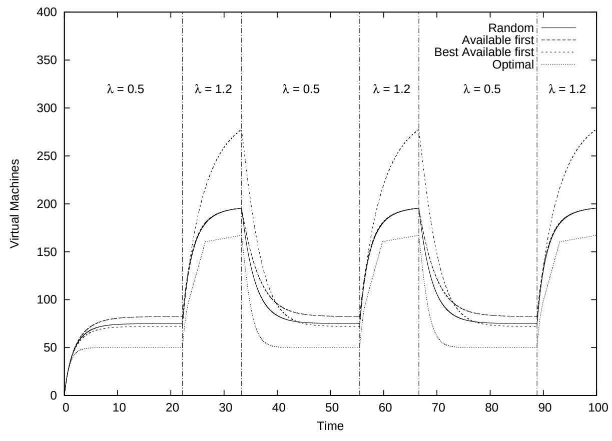


Fig. 13. Mean number of executing VMs with workload bursts

V. Conclusions

In this work we evaluate different policies for the assignment of virtual machines in an IaaS cloud infrastructures. The developed model, thanks to the Mean Field Analysis, is able to easily describe different topologies characterized by relevant numbers of components. We compared four different scheduling policies and in the considered scenarios the results highlight that the *Optimal* policy outperforms the others, indeed, it has always the lower number of executing VMs and such as the *Available First* and the *Best Available First* does not lose client requests. Moreover, results show that despite the *Random* causes the loss of requests, even the system could serve them, its mean number of executing VMs can be lower than the *Available First* and the *Best Available First* ones, providing insights about the cost planning (number of resources) versus the expected quality of service (e.g. loss requests rate).

Nevertheless some outcome could be quite intuitive, this study proposes further investigations, for instance, a model refinement devoted to take into account the time required to compute the scheduling policies could evaluate how this impacts on the overall performance. The model can also be effortlessly extended to represents PMs classes with both different CPUs and different VM sets.

References

[1] M. Benaim and J.-Y. L. Boudec. A class of mean field interaction models for computer and communication systems. *Performance Evaluation*, 65(11-12):823–838, 2008.

[2] F. Benevenuto, C. Fernandes, M. Santos, V. Almeida, J. Almeida, G. Janakiraman, and J. Santos. Performance models for virtualized applications. *Lecture Notes in Computer Science*, 4331:427–439, 2006.

[3] M. Bennani and D.Menasce. Resource allocation for autonomic data centers using analytic performance models. *Autonomic Computing*, pages 229–240, 2005.

[4] A. Bobbio, M. Gribaudo, and M. Telek. Analysis of large scale interacting systems by mean field method. In *5th International Conference on Quantitative Evaluation of Systems - QEST2008*, St. Malo, 2008.

[5] N. Bobroff, A. Kochut, and K. Beaty. Dynamic placement of virtual machines for managing sla violations. In *Proc. of Integrated Network Management, 2007. IM '07, 10th IFIP/IEEE International Symposium*, 2007.

[6] O. Bushehrian. The application of fsp models in automatic optimization of software deployment. In *In: Proceedings of the ASMTA*, 2011.

[7] F. Cordero, D. Manini, and M. Gribaudo. Modeling biological pathways: an object-oriented like methodology based on mean field analysis. In *the Third International Conference on Advanced Engineering Computing and Applications in Sciences(ADVCOM)*, pages 193–211. IEEE Computer Society Press, 2009.

[8] C. Curino, E. Jones, S. Madden, and H. Balakrishnan. Workload-aware database monitoring and consolidation. In *In: Proceedings of the 2011 international conference on Management of data*, 2011.

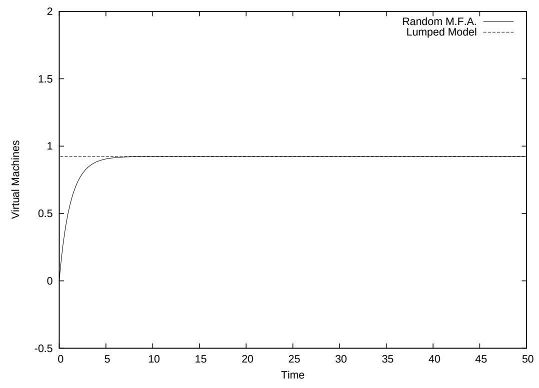
[9] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. *NOMS*, 4331:373381, 2006.

[10] P. Kokkinos, K. Christodoulopoulos, A. Kretsis, and E. Varvarigos. Data consolidation: A task scheduling and data migration technique for grid networks. In *In: Proceedings of the 8th IEEE Int. Symposium on Cluster Computing and the Grid*, 2008.

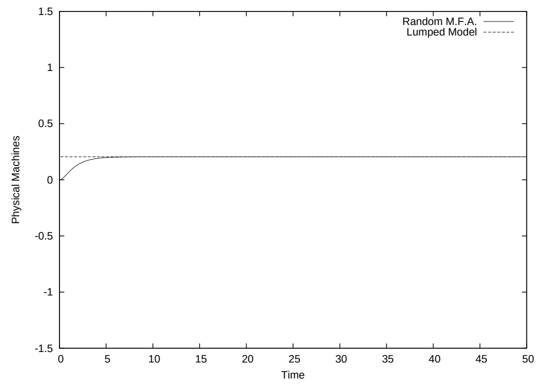
[11] T. Kurtz. Strong approximation theorems for density dependent markov chains. *Stochastic Process. Appl.*, 6:223–240, 1978.

[12] P. Padala, K. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *In: Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007.

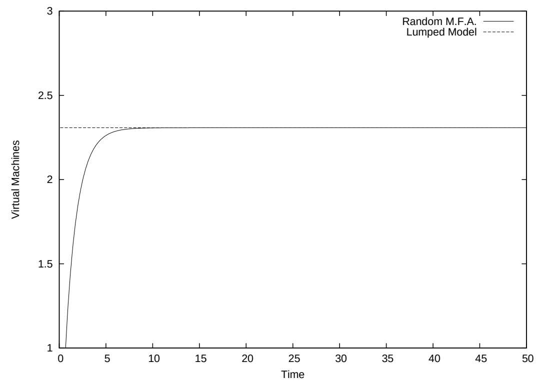
[13] B. Watson, M. M. A. Gmach, Y. Chen, M. Arlitt, and Z. Wang. Probabilistic performance modeling of virtualized resource allocation. In *In: Proceedings of the 7th international conference on Autonomic computing*, 2010.



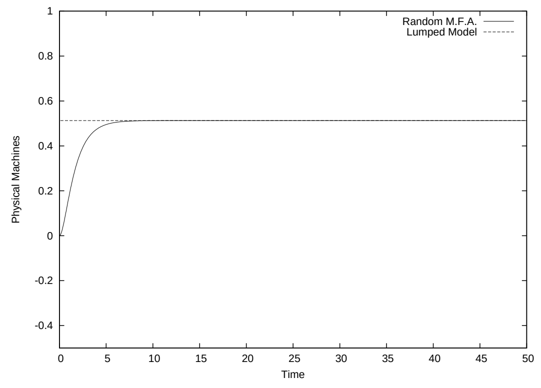
a)



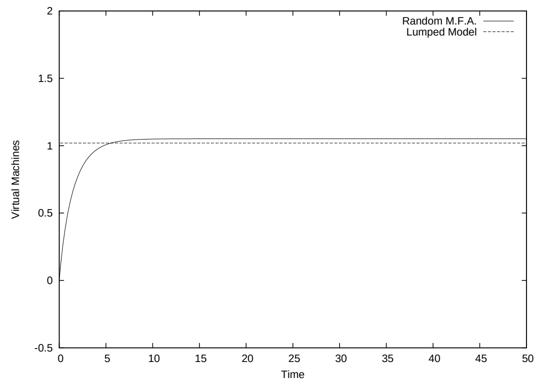
b)



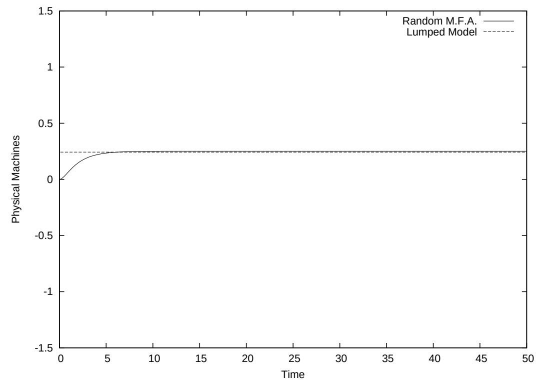
c)



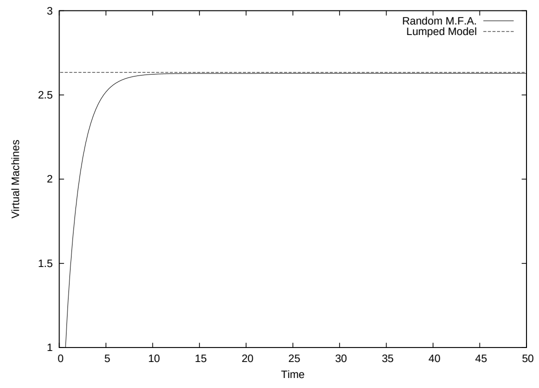
d)



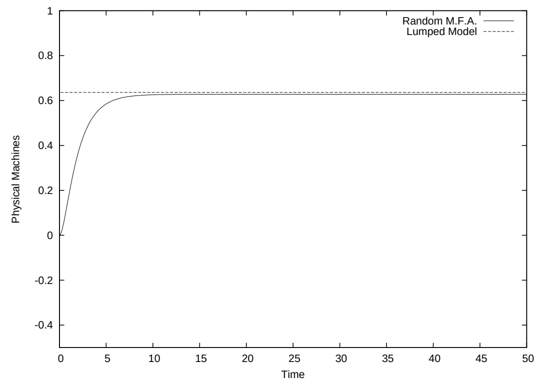
e)



f)



g)



h)

Fig. 6. Comparison of results between Mean Field Analysis and exact solution of the underlaying CTMC with the following parameters: $N = 2$, $\lambda = 0.4$ and $\mu = 1$. Random policy is considered in a)-d), while Available First in e)-h). The total number of PMs is $M = 2$ in a), b), e) and f), and it is $M = 5$ in c), d), g) and h). The mean number of assigned resources is shown in a), c), e) and g), while the mean number of full machines is plotted in b), d), f) and g).

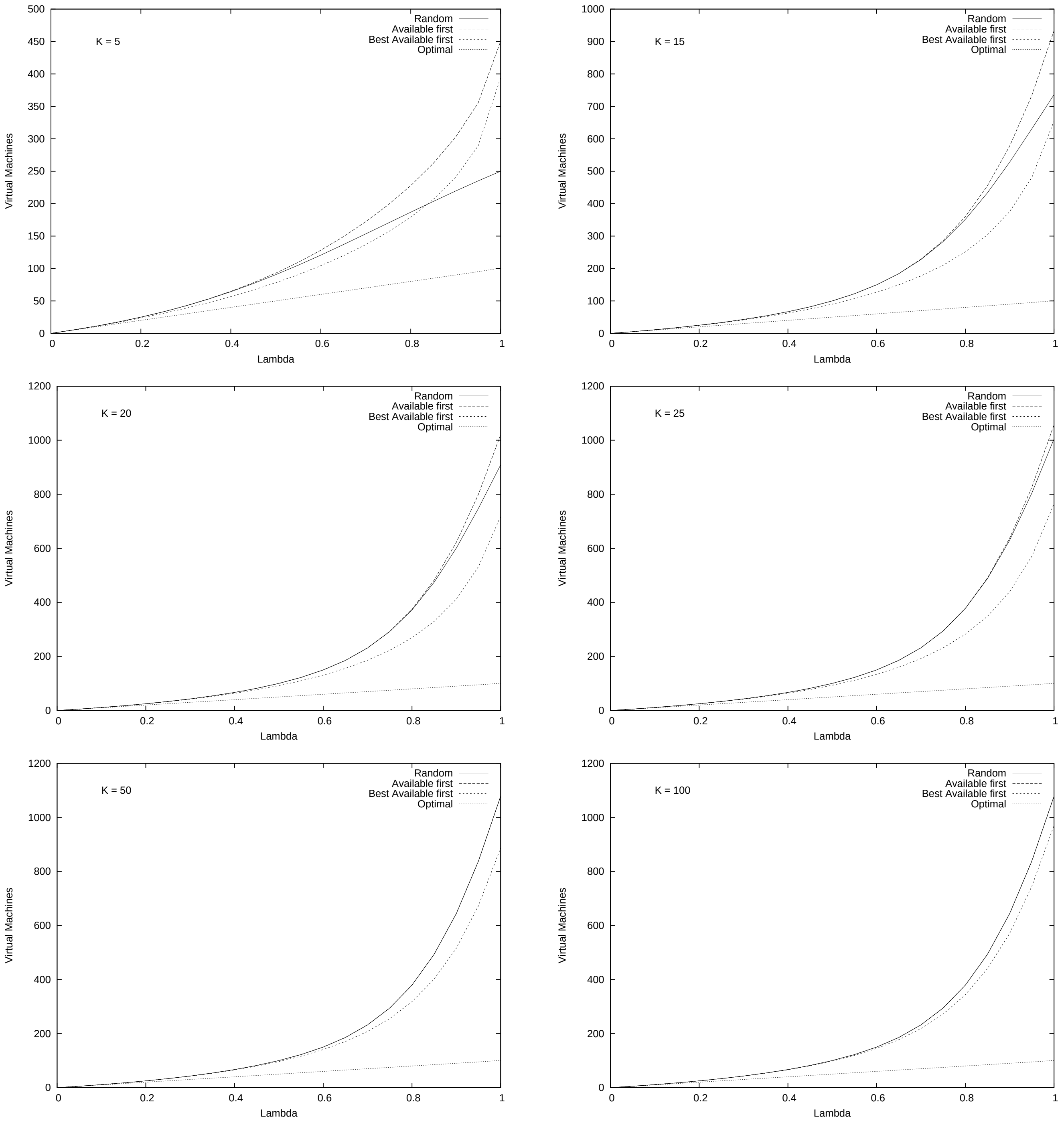


Fig. 12. Mean number of executing VMs for different K (initial number of resources)

states	5,0,0	4,1,0	3,0,1	3,2,0	2,1,1	1,0,2	2,3,0	1,2,1	0,1,2	1,4,0	0,3,1	0,5,0
5,0,0	-$(\lambda_1 + \lambda_2)$	λ_1	λ_2	0	0	0	0	0	0	0	0	0
4,1,0	μ_1	-$(\lambda_1 + \lambda_2 + \mu_1)$	0	λ_1	λ_2	0	0	0	0	0	0	0
3,0,1	μ_2	0	-$(\lambda_1 + \lambda_2 + \mu_2)$	0	λ_1	λ_2	0	0	0	0	0	0
3,2,0	0	μ_1	0	-$(\lambda_1 + \lambda_2 + \mu_1)$	0	0	λ_1	λ_2	0	0	0	0
2,1,1	0	μ_2	μ_2	0	-$(\lambda_1 + \lambda_2 + \mu_1 + \mu_2)$	0	0	λ_1	λ_2	0	0	0
1,0,2	0	0	μ_2	0	0	-$(\lambda_1 + \mu_2)$	0	0	λ_1	0	0	0
2,3,0	0	0	0	μ_1	0	0	-$(\lambda_1 + \lambda_2 + \mu_1)$	0	0	λ_1	λ_2	0
1,2,1	0	0	0	μ_2	μ_1	0	0	-$(\lambda_1 + \mu_1 + \mu_2)$	0	0	λ_1	0
0,1,2	0	0	0	0	μ_2	μ_1	0	0	-$(\mu_1 + \mu_2)$	0	0	0
1,4,0	0	0	0	0	0	0	μ_1	0	0	-$(\lambda_1 + \mu_1)$	0	λ_1
0,3,1	0	0	0	0	0	0	μ_2	μ_1	0	0	-$(\mu_1 + \mu_2)$	0
0,5,0	0	0	0	0	0	0	0	0	0	μ_1	0	-μ_1

TABLE I
CTMC INFINITESIMAL GENERATOR OF THE META-CLASS *Physical Machine*