Functional Model of a Software System with Random Time Horizon

Dmitrii A. Zubok, Aleksandr V. Maiatin, Valentina E. Kiryushkina, Maksim V. Khegai
ITMO University
St. Petersburg, Russia
zubok@mail.ifmo.ru, {mavr.mkk, v.kiryushkina}@gmail.com, MaxHegai@rambler.ru

Abstract—**Virtualization technologies are being actively used to design infrastructure of cloud computing systems. In this case applications can be duplicated and hosted in different virtual machines on different physical nodes. That defines various performance of applications which causes the problem of managing performance of the entire heterogeneous system. There are different ways of solving this problem, including queuing theory methods. However research of the threshold discipline in scope of queuing theory is not complete because of difficulty of gathering precise analytic values and building of precise mathematic model of the system. Another feature of heterogeneous systems is the finite random time of system functioning which is defined by random endogenous and exogenous factors. This paper gives an overview on a functional model of the system with two heterogeneous devices with random functioning time and different service disciplines. In scope of simulation statistic experiments for different service disciplines at random time interval an average time needed to process a single request is measured. A comparison of service disciplines is conducted. Authors also provide a working software implementation of the heterogeneous system and experiments with use of service disciplines is performed.**

I. Introduction

Software systems are widely used in modern life. They are one of the mandatory things to have in computer science and anything related to it. Lately a type of such systems called clouds became very popular. This led to appearance of variety of optimization techniques for cloud based systems.

One of the most important research topics discussed in clouds optimization related papers and articles is the resource allocation. Verma et al. [2] formulated the problem of dynamic placement of applications in virtualized heterogeneous systems as a continuous optimization: performance is optimized at each time frame by optimizing the placement of virtual machines. In [3] Chaisiri et al. proposed a virtual machine placement algorithm based on stochastic integer programming. These works discussed VM consolidation and not resource allocation at application level. Song et al. [4] presented a resource allocation approach according to application priorities in multi application virtualized cluster. For this machine learning was used to define applications priorities in advance. This proposed approach is a static resource allocation, ignoring dynamic resource demands in cloud. Further studies led to the presentation of a prototype infrastructure by Appleby [5]. This infrastructure can dynamically allocate cloud resources for an e-business computing utility. In 2007 Xu et al. [6] introduced a two-level resource management system with controllers at VM level and a global controller at server level. Their works focus only on cloud resource allocation among VMs within a server and don't consider the resource optimization among applications.

Most clouds are built in the form of data centers. They consist of the master server, computing servers and the transmission server. To provide more powerful resource capacity these servers are composed by multiple virtual machines. The master server works as a scheduler, it receives requests and distributes them to computing servers. The waiting time before a request is sent to a corresponding service is determined by resource capacity of a computing server. The same resource capacity determines processing speed for the requests. After the processing the results will be sent back to users. Transmitting server acts as a gateway node and controls the traffic.

During the work, as a part of optimization algorithms, virtual machines may need to be migrated to another physical server. Migrations are not committed in equal time intervals but when an event occurs. Because of that we can't define the time when it will happen. This is what called *random time horizon*. Usually the agent, or a controlling server in the case of clouds, doesn't know the final time before it actually comes. Thus, we can't assume that T is a finite and fixed (deterministic) variable. If the time is very large, it is often assumed to be infinite. We will assume that the planning horizon (terminal time) T has a known distribution function and finite expectation. In 1965 Yaari [7] presented uncertain lifetime, where the distribution function $F_t(s)$ is the conditional probability of the consumer dying before time s, considering he is alive at time *t*, where $t < s$. Karp and Tsur (2011) [8] applied non-constant discounting to a problem of climate change, by assuming that a catastrophic climate event occurs at a random time T. Non-constant discounting reflects that preferences change with time: an agent that made a decision at time t will have different preferences when it makes a decision at time *s*. Agent that makes decision at time *t* is called the *t*-agent.

A *t*-agent has two behaviors: naive and sophisticated. Naive one makes the agent take decisions without thinking that it will change its parameters later. Then it modifies its calculated choices for the future. In general its decisions are time inconsistent. The solution to this is in solving the associated optimal control for such an agent and patching together the optimal decision rules at time *t*. To get a time consistent strategy we need a sophisticated *t*-agent; it should take into account the preferences of all *t'*-agents, for $t' > t$. To find time-consistent solutions for agents with non-constant discounting we use a dynamic programming approach, applying Bellman's optimality principle.

In cases of non-constant discounting and random duration the *t*-agent tries to maximize the expected value of the agent's objective. The terminal time in this case is a random variable described by a distribution function $F(\tau)$. Assuming that T is independent on state and control variables, we assume that the distribution $F(\tau)$ has density function, $F(\tau) = f(\tau)$. The t-agent will then maximize next expression:

$$I_t = \int_t^{\infty} dF_t(\tau)\left[\int_t^{\tau} ds\theta(s-t)L(x,u,s)\right] + \qquad (1)$$

$$+\int_t^{\infty} dF_t(\tau)\theta(\tau-t)S(x(\tau),\tau)$$

In this paper we want to introduce readers to the area of mathematics behind these principles, show their complexity and explain them.

II. PRELIMINARIES

The problem of optimal jobs assignment to heterogeneous servers was discussed by many researchers and a lot of algorithms and models were proposed. For use in clouds the most noticeable are controllable queuing systems. It means that an arrived job is sent either to a computing server or to a queue where it waits to be computed. Which server the job is being sent depends on a service discipline. Let's consider an M/M/K/N–K $(K \le N < \infty)$ controllable queuing system. K is heterogeneous exponential servers of intensities $\mu_k (k = 1, K), N - K$ places in the buffer, and a Poisson input of jobs with the intensity lambda.

For modeling of the system operation, consider the controllable process $\{Z(t)\} = \{(X(t), U(t))\}$ with the observed process $\{X(t)\} = \{Q(t), D(t))\}$ and controlling process $\{U(t)\}$. Here $Q(t)$ is the queue length at time t, and $D(t) = (D_1(t), ..., D_k(t))$ describes the states of the servers at this time, $D_i(t) = 0$ *or* 1 depending on the idle or busy *i*-th server. The states space of the observed process is $E = N \times \{0,1\}^K$ with the set $E = \{0,1,...,N-K\}$. The decision set A consists of $K+1$ elements, i.e. $A = \{0,1,...,K\}$ and the decision "0" denotes not to occupy any server (send the arrived job to the queue), while the control "*k*" denotes to send the next job on the *k*-th server. Denote also by $L(t) = Q(t) + \sum_{1 \le k \le K} D_k(t)$ the random process of the number of jobs in the system.

For each state $x = (q, d_1, ..., d_K)$ denote by $d(x) = \sum_{1 \le k \le K} d_K$ the number of busy servers, by $l(x) = q(x) + d(x)$ the number of jobs in the system, by $J_0(x)$ and $J_1(x)$ the sets of busy and idle servers respectively,

$J_0(x) = \{j : d_j(x) = 0\}$, $J_1(x) = \{j : d_j(x) = 1\}$,and

by $M_0(x) = \sum_{j \in j_0} \mu_j$, and $M_1(x) = \sum_{j \in j_0} \mu_j$ the total service intensities of the idle and busy servers.

Notice that the decision depends on the state of the observable component and the decision set $A(x)$ at the state x equals $A(x) = \{0\} \cup J_0(x)$.

Under the considered assumptions, the process $\{Z(t)\} = \{(X(t), U(t))\}$ is a Markov decision one with transition intensities

$$\lambda_{xy}(a) = \left\{\begin{array}{l} \lambda, \text{for } y = x + e_a \\ \mu_j, \text{for } y = x - e_j + 1_{\{q(x>0)\}}(-e_0 + e_a), j \in J_1(x) \\ 0, \textit{otherwise} \end{array}\right\} (2)$$

where $e_i = (0, ..., 1, ..., 0)$ denotes the $K+1$-dimensional vector, the *i*-th component of which (beginning from 0-th) is one and all others are zeros.

As usual (see, for example, [9]) define a strategy δ, the probability distribution P_x^δ of the process $\{Z(t)\}$ given the initial state x and strategy δ, the expectation E_x^δ with respect to this probability distribution. Then in mathematical terms the problem can be represented as follows: minimize with respect to all admissible strategies the following expression:

$$g(x;\delta) = \lim_{t\to\infty} \frac{1}{t} E_x^\delta \int_0^t L(u)du \qquad (3)$$

In 2002 Rykov and Efrosinin [10] proved next theorem:

For the system under consideration an optimal policy is of a threshold type, i.e. for each state x, there exists some level of the queue length $q^*(x)$ (depending on the collection of busy servers $J_1(x)$) such that it is necessary to occupy some server only if $q > q^*(x)$; in this case the fastest of idle servers should be occupied. On the other hand, if in some state x the optimal decision is to allocate a job to the queue then the same decision is optimal for all y with $q(y) \le q(x)$ and the same collection of busy servers $J_1(y) = J_1(x)$.

That allowed them to formulate number of job minimization threshold formula:

$$q_j^* = \left[\frac{1}{\mu_j} \sum_{k=1}^{j-1} \mu_k \right] - (j-1) \qquad (4)$$

However this formula gives us an estimated value. Because of that we look not only on the values it gives us but also on their variations. This will allow us to get an optimal threshold value.

III. Queuing Models and Experiments in Simulation Environment

In this section we introduce two queuing models to compare system performance when different service disciplines are applied. AnyLogic Professional 7.1 is used as a modeling tool. Simulations are widely used in researches related to queuing models and optimization. For example, in [11] simulation is used to evaluate resource allocation schemes. There authors perform experiments with different scenarios and get charts as a result.

Basic model is shown on the Fig. 1. The model consists of one source, one infinite queue and two servers with different service rates.

It is known that a request on a web server can be modeled by a Poisson process. So we consider that requests arrive with an average rate λ. Servers have an exponential service time with average service rate μ_1 and μ_2 respectively, such as $\mu_1 < \mu_2$ and $\mu_1 + \mu_2 = M$.

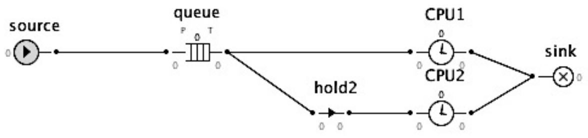

Fig. 1. Conservative service discipline model

Source element produces objects with rate λ of the class *Entity*, which was modified by adding field *startTime* for storing time when a request arrives.

Element *Queue* describes the queue. Its capacity is set up at maximum.

CPU1 and *CPU2* are the standard library *Delay* elements, which delay requests (one per moment) for a specified time. We use function *exponential()* with arguments $1/\mu_1$ and $1/\mu_1$.

In the simulation environment we implement three different service disciplines: conservative discipline, threshold discipline and discipline based on calculating minimum average service time subsequently referred as mintime discipline. Conservative discipline is FIFO discipline, where requests are served by any available server. Threshold discipline for two-server queue implies that initially we use only one server with maximum service rate, and we have threshold value q_2. When size of the queue reaches this value we should start using the second server. Threshold value is calculated according to the equation (4).

To measure conservative and threshold discipline performance we use the basic model. hold2 element used for blocking CPU2 in threshold mode. It works as follows. When a request enters the queue the unblockCPU function is called. This function checks if size of the queue reaches the threshold value and if it is unblocks CPU2 by setting hold2 blocked. When the request exits the queue the blockCPU function is called, which checks if size of the queue is less than the threshold value and if it is blocks CPU2.

Mintime discipline implies following. For each incoming request we calculate expected service time on each server t_1, t_2. If the corresponding server is available $t_i = \mu_i$, otherwise $t_i = (n_i + 1) \cdot \mu_i + t_r$,where n_i is size of the queue for corresponding server, t_r is a remaining service time for current processing request and

$t_r = t_{enter} + \mu_i - t_{current}$, t_{enter} is time when a request starts being processed and $t_{current}$ is current time. Then we compare t_1 and t_2 and put the request to a server that has minimum t_i. To measure this discipline performance we use a model similar to the basic one: model has one source, two servers, but every server has its own queue. The model for mintime discipline is shown on Fig. 2.

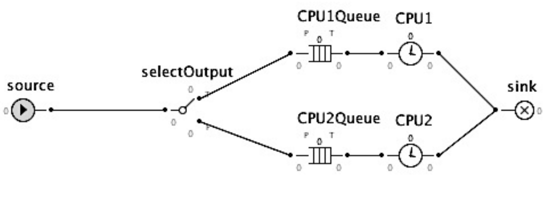

Fig. 2. Mintime discipline model

We took several experiments using these models to compare described service disciplines. General experiment structure is next. We set initial model state that implies an empty queue, two available servers and no processed requests. Then we set the model parameters: rate λ, average service rate μ_1 and μ_2 and experiment time T. As the indicator for the experiment to stop we use the function that is shown on Fig. 3. We set the threshold value Q, the number of processed requests, indicating that the experiment should stop.

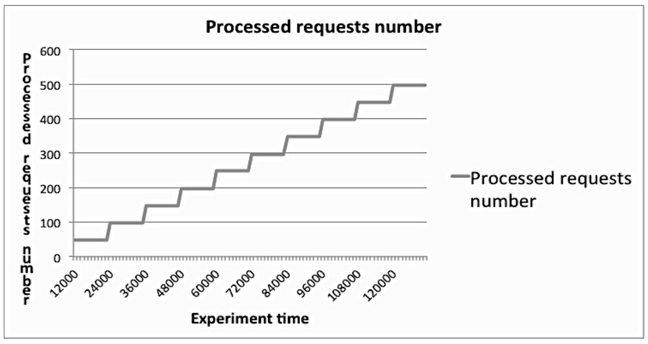

Fig. 3. Function to stop the experiment

We decided to run the model 500 times. If we run it 100-200 times we obtain sparsed average values, however 500 times turned out to be more sufficient in obtaining not so sparsed values.

Average time is calculated by calculating average service time for each run and then calculating average of average values from previous step.

Experiment results are written to files and then processed by Python script and Excel.

We took statistical experiments varying λ, ratio of μ_1 and μ_2, and threshold q_2. Each experiment we measured request service time as an average time of a request staying in the system.

We obtained that for $\lambda \geq M$ threshold discipline doesn't work, it degenerates into conservative, so we consider only for $\lambda < M$.

Let $M = 0.083(3)$. First we set a fixed $\mu_2 / \mu_1 = 0.5$ and set λ equals to 2M, 1.5M, M, 0.8M, 0.5M. But for $\lambda < M$ threshold discipline turned out to be more efficient, average service time is less by 25% compared to conservative.

Then we set $\mu_2 / \mu_1 = 0.33(3)$ for $\lambda < M$ and obtained that threshold discipline is more efficient when $q_2 = \frac{1}{\mu_2} \cdot \mu_1$.

But the usage of conservative discipline gives final queue size smaller by 20%.

Then we measured system performance for mintime discipline. Average service time for this discipline is less by 25% comparing with conservative and queue for rate $\lambda < M$ and queue size is more than four time lesser than for conservative and threshold disciplines.

Average service time for this discipline is less by 25% comparing with conservative and queue for rate $\lambda < M$ and queue size is more than four times lesser than for conservative and threshold disciplines. The obtained value for the conservative discipline's average service time is 385,5897 and for the improved discipline is 333,7046 and $\mu_2 / \mu_1 = 0.243$. Other obtained values are listed in the table 1, where q$_2$ is threshold value for queue size.

TABLE I. AVERAGE PROCESSING TIMES OF VIRTUAL MACHINES

q₂	Threshold average time, seconds	Conservative/ Threshold, seconds	Improved/Threshold, seconds
1.0	309,2085	1,2470	1,0792
2.0	313,9540	1,2281	1,0629
3.0	326,7411	1,1801	1,0213
4.0	343,0303	1,1240	0,9728

We took another experiment studying queue size at the end of experiment, and the end time is random: we stop the experiment when the number of processed requests equals 1000, which is an optimal value we gathered after a number of tests. The reason for that is that at 1000 requests the system is already in stationary state and shows all characteristics of Poisson process. We obtained the following ordering of experiment end time for each discipline $\tau_c < \tau_{mst} < \tau_t$, where τ_c is an average experiment end time for conservative discipline, τ_{mst} is average experiment end time for discipline with min service time choice, τ_t is average experiment end time for threshold discipline. Conservative discipline has a minimal average queue size and discipline with min service time choice has the maximal average queue size.

The experiment also shown that for an exponentially distributed random termination time the processing time for threshold discipline was lesser than for conservative discipline.

IV. Architecture of the software environment

To gather data from a software environment we need a proper system. Such a system can be built by using virtualization to simulate different physical servers. Xen-hypervizor allows several virtual machines to be created and run simultaneously and share hardware resources with them. It also allows us to limit some of the parameters such as maximal CPU usage for each virtual machine which will help us simulate real servers by setting different performance levels. Those virtual machines are connected to each other via single virtual interface that works as a router, thus giving access to any virtual machine from each one of them. The architecture is presented at the Fig. 4.

The choice of Xen is motivated by its often usage in scheduling analyze and high-performance communications [12].

The main goal of the experiment is to look at the system's behavior when we use gathered data and adjust parameters for the software environment to find the best outcome. We then will compare both performance outputs from the simulation environment and software environment and see if they lie within our expectations. For this to be possible we implemented the model described in the previous section with only input and output changed to be controllable by us.

Its algorithm is:

- A request is created by user (or, in case of the presented architecture, it's created in different intervals, generated by random numbers generator);
- This request is added to the common queue on the controlling virtual machine;
- Following a service discipline the request is sent to appropriate virtual machine;
- The request is processed and an answer is sent back to the controlling virtual machine.

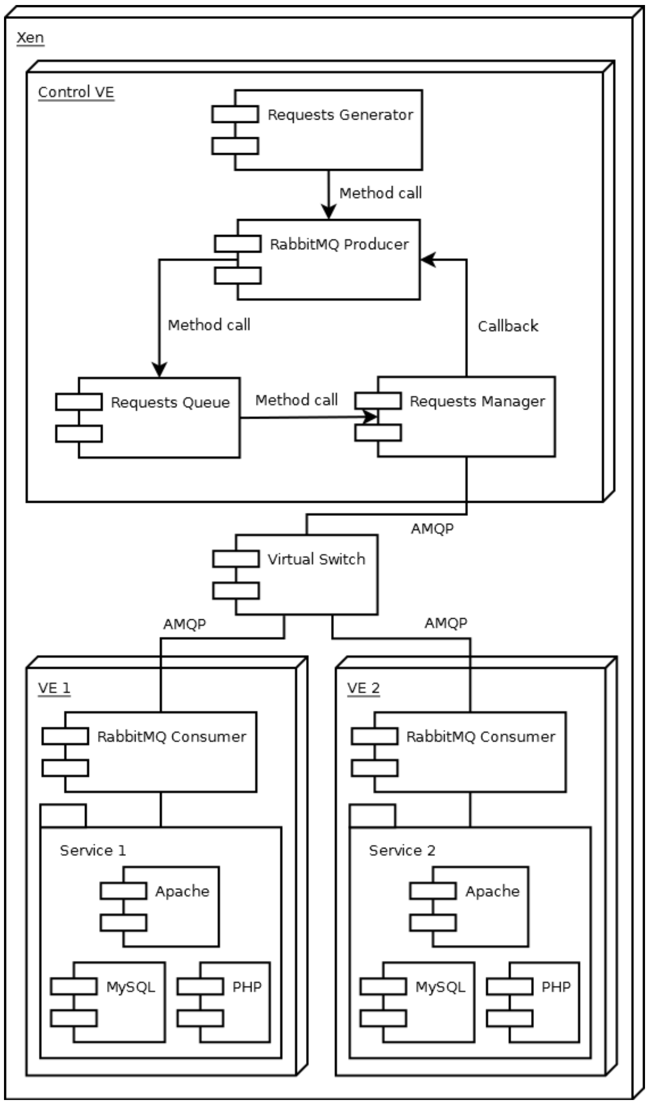

Fig. 4. Architecture of the system

There are currently two service disciplines implemented in the system: conservative and threshold. Both of them use the same architecture and may be quickly swapped. Conservative discipline is very simple: as soon as a request is added to the common queue, the system looks for a first free server (virtual machine) and sends it there. If no free servers is found the request stays in the common queue.

Threshold discipline is generally an upgraded conservative discipline. Here, instead of immediately sending a request in a first free server, we wait until the number of requests in the common queue becomes higher than threshold value and only then begin to send requests to the next free server. Until it happens every request is sent to a previous server, waiting for the previous request to be processed. The functional model of the system is presented on Fig.5.

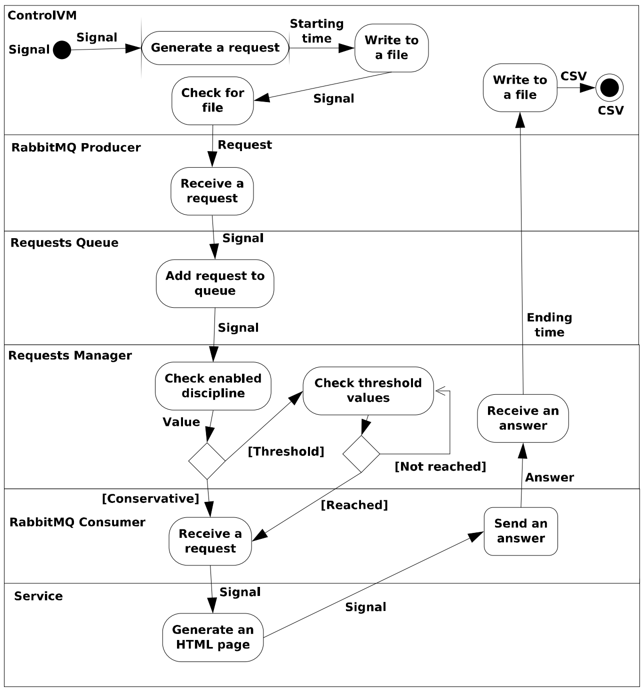

Fig. 5. Functional model of the system

The servers are sorted by performance level: highest to lowest, which is characterized by one parameter:

- Maximal CPU usage.

This parameter can be set using default tool available in Xen-hypervizor, called *xm*. Its parameter *sched-credit* can set CPU cap at needed level. In current system there are two servers, each with 256 mb of RAM. The maximal CPU usage for the first one is 55% of the overall hardware CPU time and the second one is 25%. The maximal CPU usage for the controlling server is 20%.

To make queuing possible we add two additional parameters:

- Number of simultaneously processed requests;
- Threshold.

They are set in the control script that we wrote. Each server has to have different threshold values. However, if we use a conservative discipline this value is completely ignored. Number of simultaneously processed requests is a value that defines how many requests a server is guaranteed to process before they will be sent to the next server (giving that a request cannot be processed or a threshold value has been reached).

Requests are sent by using RabbitMQ framework. It is an easy-to-use framework made for exchanging messages between a server and a client. What is important for us is that it works asynchronously, it has support for message queues and it works with a variety of protocols. We use AMQP as a protocol and PHP as a programming language to use RabbitMQ.

To test the performance an HTML page is generated by PHP when a request is received. The page is the same on every server; otherwise we can't evaluate the performance properly. When a request is added to the common queue, the control node writes receiving time to a variable. Then a page is generated and when it's done finishing time is written to another variable. The difference between these variables gives us the processing time. The smaller it is the higher the overall performance of the system. After it is calculated the processing time is written to a CSV file to be used later.

V. Experiment in the software environment

The experiment that was conducted consisted of two parts. First the random numbers generator was disabled and a constant interval T = 3 seconds is set. Such interval is needed to get an adequate precision of comparison of values. Only one computing server was turned on. Then the system was started and requests were created and added to the common queue. The control server worked for 50 cycles after which the system was shut down. The gathered data was then used to calculate average processing time of the server. The chart is presented on Fig. 6.

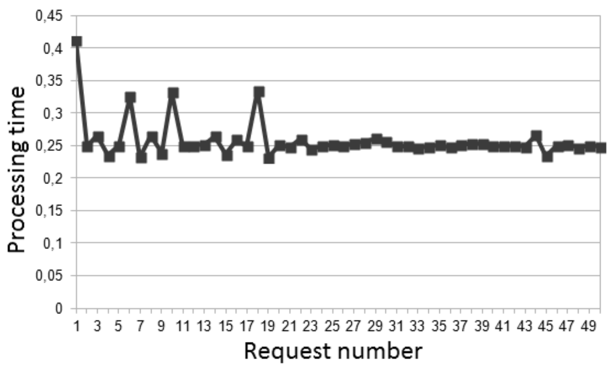

Fig. 6. Processing time of requests sent with 3 seconds interval on first VM

Then a constant interval T = 1 seconds was set and the system started again. The control server worked for 50 cycles and then the system was shut down. Again, the data was collected to calculate average processing time. The chart is presented on Fig. 7.

These average processing times that we gathered in these experiments were used to calculate the general average processing time of the first computing server. The same was done for the second server, the collected data is presented on Fig. 8 and Fig. 9.

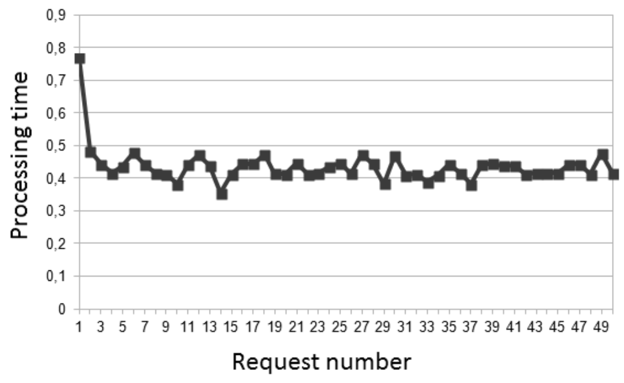

Fig. 7. Processing time of requests sent with 1 seconds interval on first VM

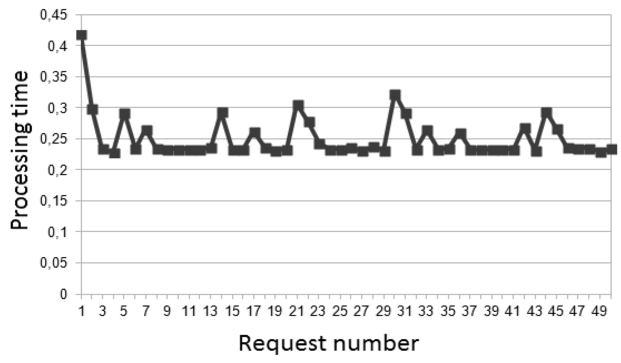

Fig. 8. Processing time of requests sent with 3 seconds interval on second VM

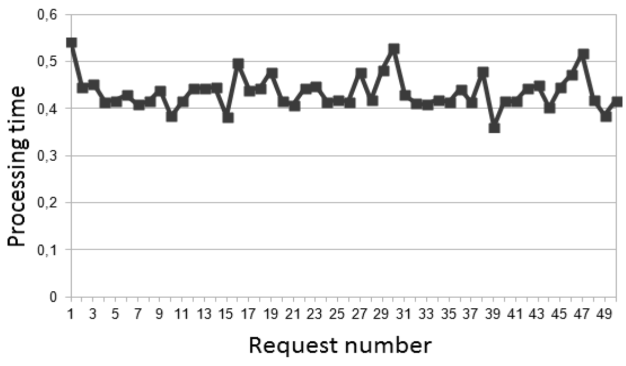

Fig. 9. Processing time of requests sent with 1 seconds interval on second VM

Using those averages processing times an average number of requests per second was calculated. The generator was turned on to calculate a random interval. After that the second part of the experiment was conducted for two service disciplines: conservative and threshold. The system worked for 1000 cycles. Thresholds for the threshold discipline were 5 and 10 requests. The resulting charts are presented on Fig. 10 and Fig. 11. All gathered values were added to the tables 2 and 3.

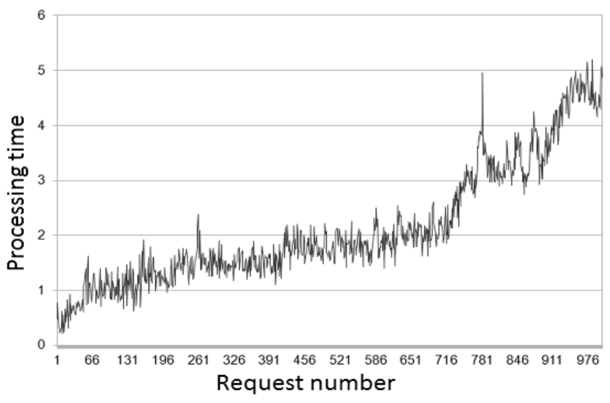

Fig. 10. Processing time of requests using conservative discipline

On the Fig. 10 which shows how processing time of requests with conservative discipline changed it is obvious that with time processing time increases. That is a known effect which means that requests begin to stay in the queue for a longer period of time. This effect is less visible on the Fig. 11.

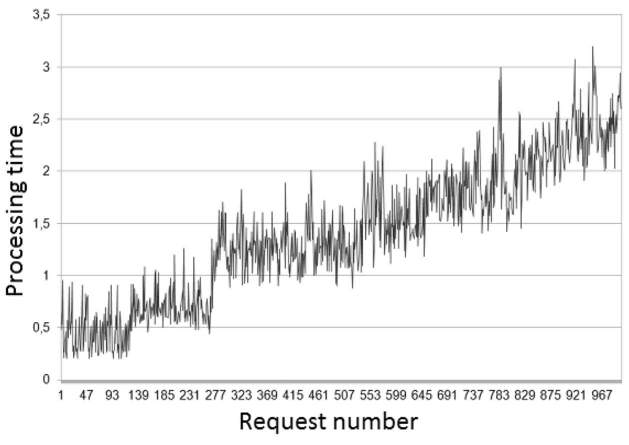

Fig. 11. Processing time of requests using threshold discipline

TABLE II. AVERAGE PROCESSING TIMES OF VIRTUAL MACHINES

VM	Interval	Average Processing Time
VM1	3 s	0,251110625
VM1	1 s	0,258107819
VM2	3 s	0,435131251
VM2	1 s	0,436286227

The Table II consists of intervals between requests and an average processing time with those intervals for every virtual machine we have.

TABLE III. AVERAGE PROCESSING TIMES OF THE SYSTEM

Discipline	Average Processing Time	Requests left in queue
Conservative	2,117474206	15
Threshold	1,401167783	9

The table 3 shows how many requests were left on the queue at the end of the experiment and an average processing time for both disciplines that were used.

The experiments made it obvious that the threshold discipline is a better choice than the conservative one. Not only it gave us lower processing time but also less requests left in the common queue.

VI. CONCLUSION

This paper provided an analysis of data gathered during experiments in simulation environment and experimental environment. We also built a functional model of the system and used it in the experiment in experimental environment. The conducted experiments allowed us to compare two service disciplines that we used, and analysis of data showed that while using the threshold discipline processing time is lower than while using the conservative discipline. That means that the threshold discipline is a better choice, almost doubling the performance.

Further we will perform other experiments in the experimental software environment we built. Those experiments are:

- Experiment with an upgraded threshold discipline which will take into account average time needed to process a request and send it to different virtual machines accordingly;
- Experiment with more than two virtual machines;
- Experiment with chain of virtual machines that a request needs to be processed in;
- Experiment with virtual machines that have different applications.

ACKNOWLEDGMENT

This work was partially financially supported by the Government of Russian Federation, Grant 074-U01.

The presented result is also a part of the research carried out within the project funded by grant #15-07-09229 A of the Russian Foundation for Basic Research.

REFERENCES

[1] G.W. Ainslie, *Picoeconomics*, UK: Cambridge University Press, 1992.

[2] L. Karp, T. Tsur, "Time perspective and climate change policy", *Journal of Environmental Economics and Management*, 1992, pp. 1-14.

[3] A. Verma, P. Ahuja, A. Neogi, *pMapper: power and migration cost aware application placement in virtualized systems*, Springer J. Middleware (2008).

[4] S. Chaisiri, B. Lee, D. Niyato, "Optimal virtual machine placement across multiple cloud providers", in: *Proc. IEEE Asia–Pacific Services Computing Conference*, 2009, pp. 103–110.

[5] Y. Song, H. Wang, Y. Li, B. Feng, Y. Sun, "Multi-Tiered On-Demand resource scheduling for VM-Based data center", in: *Proc. of IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 148–155.

[6] K. Appleby, S. Fakhouri, L. Fong, G. Goldszmidt, M. Kalantar, S. Krishnakumar, D. Pazel, J. Pershing, B. Rochwerger, "Oceano-sla based management of a computing utility", in: *Proc. of IEEE International Symposium on Integrated Network Management*, 2001, pp. 855–868.

[7] M. E. Yaari, "Uncertain lifetime, life insurance, and theory of the consumer", *The Review of Economic Studies*, 1965, pp. 137-150.

[8] J. Xu, M. Zhao, J. Fortes, R. Carpenter, M. Yousif, "On the use of fuzzy modeling in virtualized data center management", in: *Proc. of IEEE International Conference on Autonomic Computing*, 2007, pp. 25–25.

[9] M. Yu. Kitaev, V.V. Rykov, *Controlled queueing systems*, New York: CRC Press, 1995.

[10] Xiaoming Nan, Yifeng He, Ling Guan, "Queueing model based resource optimization for multimedia cloud", *J. Vis. Commun. Image. R*, 2014, pp. 928-942.

[11] V. Rykov, D. Efrosinin, "Numerical Analysis of Optimal Control Policies for Queueing Systems with Heterogeneous Servers", *Information Processes*, Vol . 2, 2002, pp. 252-25

[12] A. Nanos, N. Koziris, "Xen2MX: High-performance communication in virtualized environments", *The Journal of Systems and Software*, 2014, pp. 217-230.