Evaluation Of Optimization Control Parameters in Multi-Leveled Cloud Platform

Dmitrii A. Zubok, Tatiana V. Kharchenko, Aleksandr V. Maiatin, Maksim V. Khegai
ITMO University
St. Petersburg, Russia
{Zubok, Kharchenko}@mail.ifmo.ru, {mavr.mkk, MaksimKhegai}@gmail.com

***Abstract*—Cloud platforms demand the highest performance available. To achieve this different optimization methods were proposed and tested. Each method has its own set of parameters and settings. While consolidating them into one single hierarchy leads to increase in performance, this consolidated method is also more complex. Thus, it is almost impossible to foresee which parameters should be configured and which values they should have without a real testing environment. However even with such an environment a set of experiments need to be performed. Those experiments provide comparison data, allowing determine values for such parameters. In this paper such experiments were performed and values were gathered which will help to properly configure the consolidated method and gain increase in performance for the system.**

I. INTRODUCTION

Cloud platforms have become an integral part of present life. Since their creation they were actively researched and developed, leading to many solutions based on clouds, such as Infrastructure-As-A-Service or Software-As-A-Service models. Those models however, demand performance and stability during their functioning. Most often they use virtualization and several physical servers with a host system to implement some kind of an infrastructure. Without any optimizations it's impossible to provide decent performance in terms of long run: some virtual machines will unavoidably slow down during the work. There are a lot of researches made in this direction. For example, in [1] Son, Hak and Young Yeom propose use of fast storage devices in order to improve cloud platform data access. To improve calculation performance in [2] authors try to improve performance by combining modeling and search based paradigms.

In general there are three often mentioned methods of increasing processing power and optimizing server load.

Such methods include:

- Live migration of Virtual Machines
- Migration of applications (services) to another virtual machine.
- Incoming jobs scheduling

A. *Live migration*

Live migration is used when resources of one physical server are not enough to keep virtual machines performance on a decent level. The method consists of simply moving a virtual machine to another physical server, however, without stopping it or disconnecting a client from it. That means that downtime of the service is minimal and only occurs at the moment of swapping to a migrated virtual machine. Everything in the old machine will be copied to the new one, including memory and storage. The method has few disadvantages:

- Migration is a slow process, so it shouldn't be done too often or even with a few virtual machines at the same time.
- It loads the network and can decrease data exchange performance
- For a proper work it demands a physical server that can afford running the migrated virtual machine

Performance tests on live migration were performed by Akoush et al in [6] and the optimizations were researched in [8].

B. *Migration of applications*

Migration of applications is similar to migration of virtual machines in a way that it serves as a mean to decrease load on a physical server. However the difference is that applications may not only be migrated to another virtual machine on another physical server but also on another virtual machine on the same server. That helps balancing the load by migrating some applications and keeping other. However, the method can't be used if there is no virtual machines that can run this application. This may be solved by preinstalling all necessary for an application components in all virtual machines and saving it as a template. A way to migrate web-applications by serializing it and reconstructing on a target device was presented in [7]. In [9] authors apply live-services migration to a wireless cloud to increase performance.

C. *Incoming jobs scheduling*

Before a job is sent to be processed, the load may be optimized by the system by deciding which virtual machine should process a job. This happens in background, with the system having a jobs queue and a working algorithm that constantly checks status of each virtual machine. If performance of one virtual machine becomes too low for it to be able to process a new job, then it is sent to another, less loaded machine. Jobs scheduling was properly explained in [3] and [4]. This method is good for long term execution as it need additional data and statistics to optimize the load.

A different method is to use all three listed methods, consolidating them into one single hierarchy. This was partially described and implemented in [5]. All three methods are used one by one in case performance hasn't been improved. For this a set of testing scripts (intellectual agents) which check the current performance was created. The main controller decides if it's necessary to go to a next step.

However, tests for applications and virtual machines migration need additional data in form of threshold values to be used in checking. This is done by checking performance of a virtual machine that performs migrations. The optimization algorithm also needs to decide if overall performance while migrating is not significant and that system will benefit from migration at all. For example if jobs stream is distributed then at some moments there will be increased intensity in jobs income, thus decreasing performance. The system needs to determine if it's a temporary decrease or if it cannot provide enough resources to keep performance level at optimal level and an optimization algorithm needs to be run.

The three methods as have been said are consolidated and separated to different layers. Each of them when started will decrease performance for a certain amount of time and thus needs to be started only when there are obvious benefits from doing so. The determination of moments when a next level is in order to be involved is not an easy task and cannot be decently solved by analytics only. So instead experiments are needed to determine performance at each level which will then be used as threshold values. Another problem is in testing environment itself. Simulation environments such as AnyLogic are not enough for this task since there are a lot of parameters that cannot be modeled with a decent accuracy. As a result a need to build a real testing platform with real hardware and software occurs.

In the present paper results of a research on performance of each method are presented. This will help us to predict performance changes and use this information to optimize the utilization of physical server's resources.

II. TESTING ENVIRONMENT

A. *Platform description*

To perform experiments, a simple testing platform was built. It consisted of two physical servers with 2GHz CPU, 2048 MB RAM and 20GB of HDD each. Those servers ran XEN hypervisor and supported operating-system-level virtualization, meaning that virtual machines that were deployed on servers share the kernel but have separate resources. Linux Debian was chosen as host and guest system for virtual machines. Each physical server has a controlling virtual machine deployed at all times. As a way to exchange data between virtual machines and physical servers, RabbitMQ was used. This means that servers must be placed in the same network, making connection from each virtual machine to another possible. RabbitMQ is used only to send or receive simple text data and doesn't act in sending virtual machine data. This task is performed by XEN hypervisor. The architecture of the platform is represented on fig 1.

On the figure 1, "Virtual Machine 1" and "Virtual Machine 3" are the controlling ones and have a controller inside. "RabbitMQ Receiver" in each virtual machine is the only way for them to connect to other machines, and can receive messages from "RabbitMQ Streamer" and send messages back. "RabbitMQ Streamer" works as a switch, connecting all virtual machines into one single network. Physical servers themselves are in the same network to be able to send virtual machines and applications to each other.

To measure the performance each virtual machine has a performance monitoring agent. Those agents are light-weight processes that check performance and send data to the controller on-demand. Storages for virtual machines and applications serve as main storages for every physical server to receive new machines and applications when in need of deploying new ones.

Each virtual machine is a Linux Debian based XEN container with a few components already preinstalled. They include:

- RabbitMQ
- PHP5
- MySQL server
- Python interpreter

As of now PHP is the language the controlling scripts were made with and is used to run them. Since some scripts are supposed to work in background, they are executed as daemons. Those scripts are made as lightweight and performant as possible, to reduce overheads to the level where their influence is almost nonexistent.

The infrastructure of the server is similar to previous one we presented in [5] but simplified. One of the servers is the main one, containing all controlling scripts. The second one is an additional one, existing to receive migrated virtual machine or application. It, however, runs a daemon that listens to requests or commands from the main one. There are four main scripts for experiments one script to generate load, one script to control data exchange and one script dedicated to control applications migration.

The testing platform as a whole allows to measure performance on each optimization level and even all of them at the same time. The measurements the platform does are not merely for live migration.

B. *Scripts description*

1) VM migration: This script calls Xen migration function and waits for an answer.

2) Application migration: This script starts an application migration by sending a command to controller on another machine.

3) Run experiments without load: This script runs experiments without load several times, depending on

parameters set when running the script. The load generator is turned off.

4) Run experiments with load: This script works the same as the previous one but the generator is turned on.

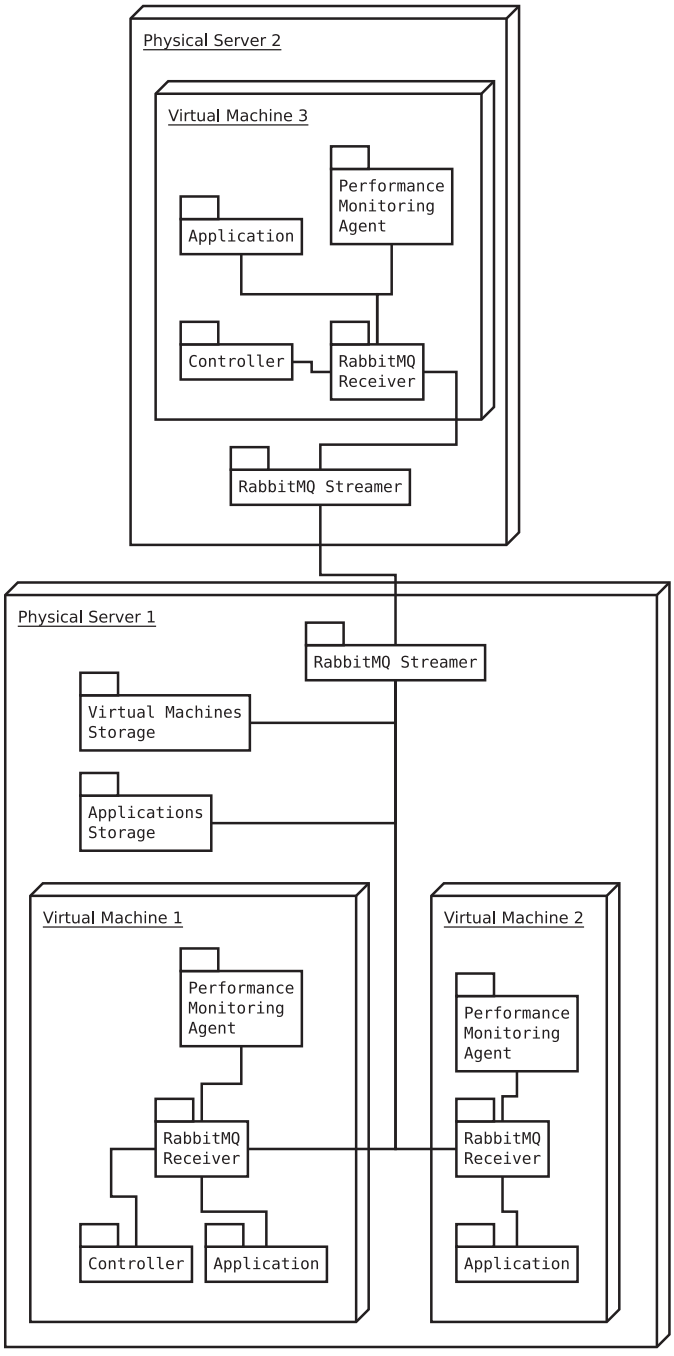

Fig. 1. Testing platform architecture

5) Application Migration Controller: This controller uses Data Exchange Controller internally. Essentially what it does is it sends a data about which application and to which virtual machine it should be migrated.

6) Data Exchange Controller: This controller's function is to receive data, determine which virtual machine it should be sent to, and send it, and runs in a controlling virtual machine on each physical server. In case an answer is required, it waits until it is received. When starting the script opens a messaging thread for each ip address found in the system. Ip addresses are added manually and are associated with each virtual machine existing in the system. The messaging threads work asynchronously and separately one from another, meaning that messages can come and go without blocking other messages. They also can come in a messages queue and stay there even if receiving end is not working. They will be sent after a connection is established. This behavior is defined in RabbitMQ.

7) Load Generator: The load is simulated by making a virtual machine calculate a function. The job to calculate the function is generated with an intensity distributed by the Poisson law. This creates a constant but controllable load where we can change how intense the calculation of one single job will be and how often those jobs will income. Since we don't test jobs queueing as it was done by us in previous works, we don't need scheduling to be active, thus each job that comes is sent to a virtual machine immediately.

The built system contains only those scripts and doesn't have anything else, thus minimizing chances of overheads and giving us clearer results.

III. EXPERIMENTS

There were two sets of experiments performed:

- Experiments without load
- Experiments with load

Both experiments were done for two cases: migration of a virtual machine and migration of an application. Each experiment was performed 10 times and had 100 iterations.

A. Experiments without load

Experiments without load were performed to get base values to compare other results with. Jobs generating script was disabled and VM migration and applications migration were tested. Next parameters were researched:

- Time since process began
- Downtime

Here "time since process began" is self-explanatory and means how much time has passed since migration started.

"Downtime" is for how long the application that was running on virtual machine (or application that was migrating) couldn't process a job. To measure this, a low costing job was generated and sent to application without involving the jobs generator. The only thing this job did was requesting any answer from application.

An experiment to check calculation performance without migration was also performed. The values are needed to compare with calculation performance with migration later.

B. Experiments with load

Experiments with load showed us how much processing time each task requires while processing jobs at the same time.

Next parameters were researched:

- Time since process began
- Downtime
- Calculation performance while migrating

The first two parameters are the same as before. And the third one is needed to get actual performance changes while migrating.

IV. RESULTS

The results are presented on figures 2 – 12. After gathering the values were filtered by using moving average method. "Average" values used here to show tendency in changes were gathered by applying the filtering several times.

The calculation time without load was measured to have something to compare to. These values by themselves don't give us any information. The result is presented on fig. 2 and the average time is 3000ms. The "Average" values show the tendency of performance and it is obvious that with time performance stabilizes around average value.

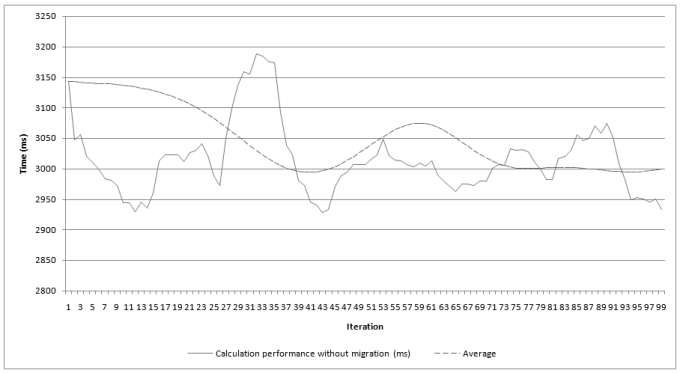

Fig. 2. Calculation time without load and with application migration

The migration time is, as supposed, higher with VM migration, staying at average on 175000ms. The time when migrating application is significantly lower, at 7500ms, as can be seen on figures 3 and 4. Tendency in application migration time here is slowly decreasing, however in VM migration it is increasing.

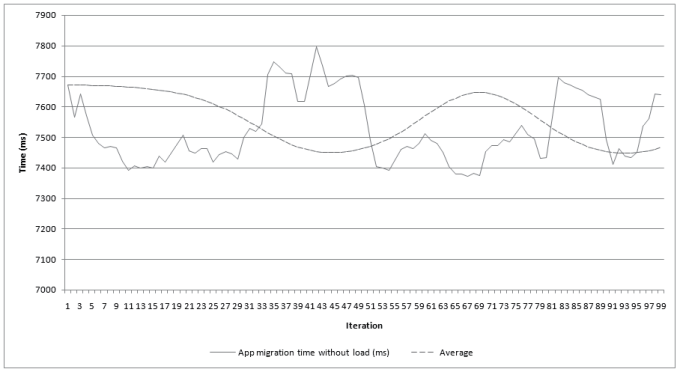

Fig. 3. Application migration time without load

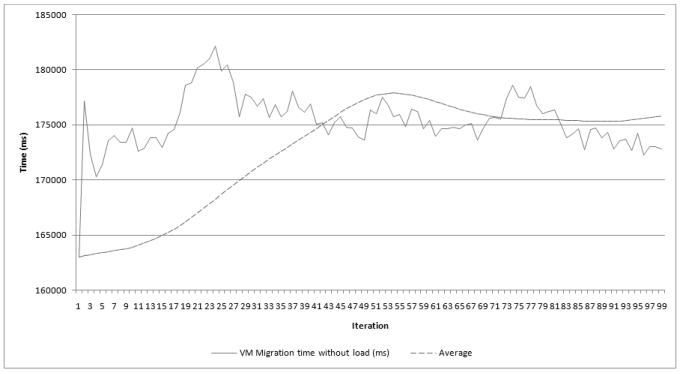

Fig. 4. VM migration time without load

The downtime with VM migration is also a little higher but mostly the same as with application migration, they are at average 4600ms and 3800ms relatively. Those can be seen on fig. 5 and 6. Tendency in both cases is increasing.

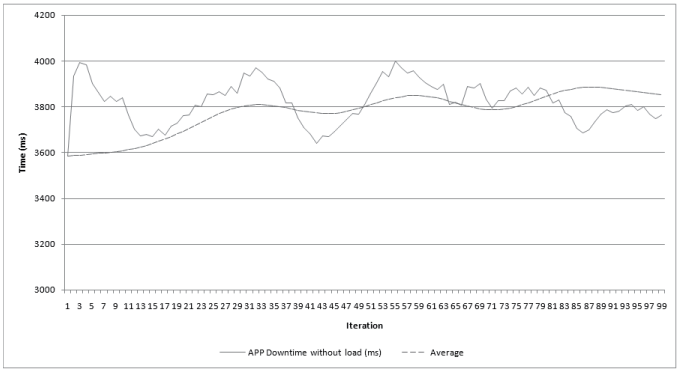

Fig. 5. Downtime while migrating application without load

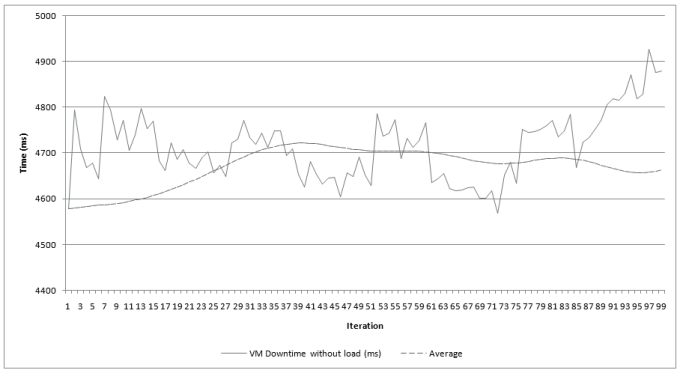

Fig. 6. Downtime while migrating VM without load

Calculation time when migrating was supposed to be much higher, however the results were quite optimistic. With application migration held, at average, at 4300ms, while with VM migration it held at 7100ms with occasional peaks at 7900ms. This may be explained with some overheads due to system processes working. The results are presented at fig. 7 and 8. Tendencies in both cases are increasing.

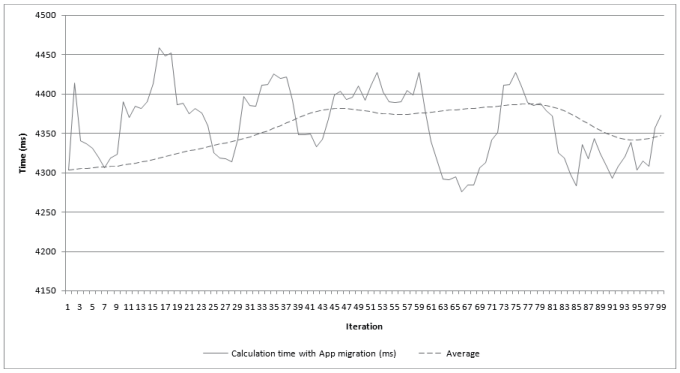

Fig. 7. Calculation time with application migration

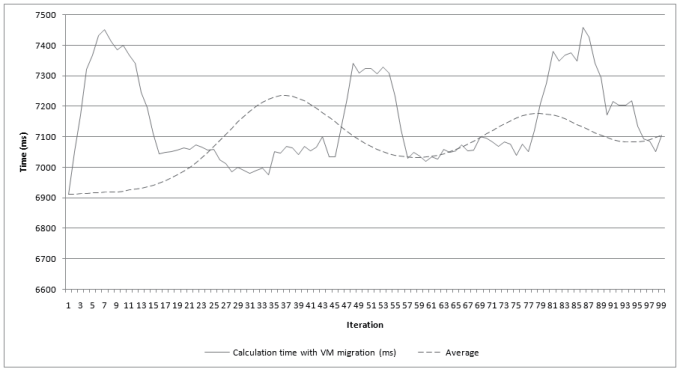

Fig. 8. Calculation time with VM migration

Application migration time with load stayed mostly at 37000ms and VM migration time stayed at 500000ms at average, with peaks at 600000. The explanation is that while copying big chunks of data the time is not always constant due to different factors like system load or network load. The results can be seen at fig. 9 and 10. Tendencies are increasing, however, slowly than in case of migration without load.

The downtime however was higher with VM migration, almost at 7500ms with peaks at 8200ms. While migrating an application the time was at 4300ms. Fig. 11 and 12 show this. Tendencies are around average values.

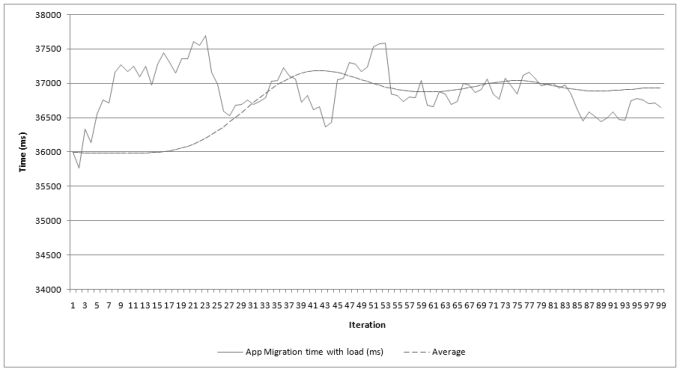

Fig. 9. Application migration time with load

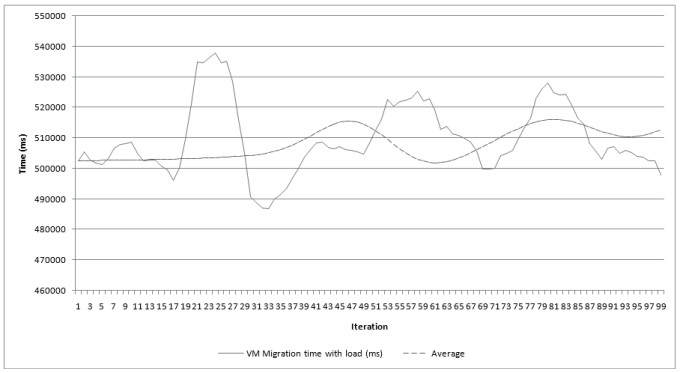

Fig. 10. VM migration time with load

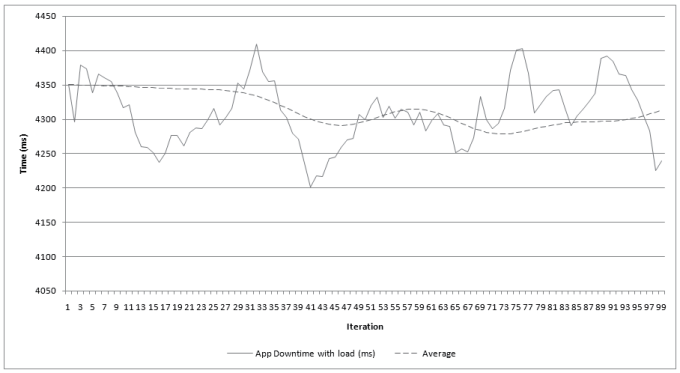

Fig. 11. Downtime while migrating application with load

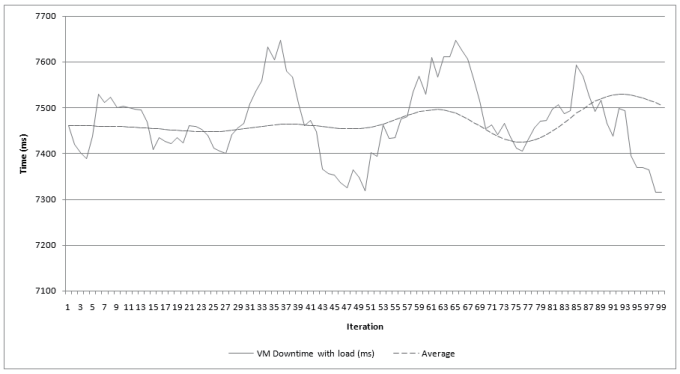

Fig. 12. Downtime while migrating VM with load

As a result the next comparison table was formed:

TABLE I. COMPARISON OF EXPERIMENTS RESULTS

	No load (ms)	With load (ms)
VM		
Calculation performance		7100
Migration time	175000	500000
Downtime	4700	7500
App		
Calculation performance		4300
Migration time	7500	37000
Downtime	3700	4300

V. Conclusion and future work

The results of the experiments will allow us to properly compare performance of our system after optimizations described in our later work are done. As of now the results show that there performance doesn't decrease that much while migrating a virtual machine or an application, which leads to a conclusion that there may not be much overheads during the optimization process. Tendencies that were gathered during migration with load also support this.

In the next work experiments with all levels of optimization working will be performed as well as improvements to the platform will be done.

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] S. Yongseok, H. Hyuck, Y.Y. Heon, "Optimizing file systems for fast storage devices", Proceedings of the 8th ACM International Systems and Storage Conference, May 2015

[2] E.K. Donia, F. François, N. Grégory, M.A. Jorge, A. Michel, L.T. Yves, "Generic cloud platform multi-objective optimization leveraging models@run.time", Proceedings of the 29th Annual ACM Symposium on Applied Computing, Mar. 2014

[3] D. A. Zubok, T. V. Kharchenko, A.V. Maiatin, and M. V. Khegai, "Functional model of a software system with random time horizon", in Proc. FRUCT Conf., 2015, pp. 259-266.

[4] D. A. Zubok, T. V. Kharchenko, A.V. Maiatin, and M. V. Khegai, "Ontology-based approach in the scheduling of jobs processed by applications running in virtual environments", Knowledge Engineering and the Semantic Web, vol.518, 2015, pp. 273-282.

[5] D. A. Zubok, T. V. Kharchenko, A.V. Maiatin, and M. V. Khegai, "Multi-Leveled Hierarchical Control to Optimize Workload of a Service-Oriented Platform", in Proc. FRUCT Conf., 2016, pp. 279 - 284.

[6] A. Sherif, S. Ripduman, R. Andrew, A.W. Moore, A. Hopper," Predicting the Performance of Virtual Machine Migration", MASCOTS'10, Aug 2010.

[7] K. Jin-woo, M. Soo-mook, "Web application migration with closure reconstruction", Proceedings of the 26th International Conference on World Wide Web, April 2017, pp 133-142

[8] S. Sangeeta, C. Meenu, "A three phase optimization method for precopy based VM live migration", Springerplus, 2016

[9] M. Andrew, W. Shiqiang, K.L. Kin, J.K. Bong, S. Theodoros, " Live Service Migration in Mobile Edge Clouds", IEEE Wireless Communications 99, August 2017