Multi-Leveled Hierarchical Control to Optimize Workload of a Service-Oriented Platform

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

***Abstract*— nowadays, one of the directions of development of cloud systems is a creation of open platforms, providing resources for deploying third-party services. Users of such resources may deploy their own services, implemented with a wide range of technologies, receiving guaranteed performance and availability. To provide high availability and optimal utilization of hardware resources such platforms are built with a complex hierarchical infrastructure. As complexity of such systems increased, due to inability of control methods on different levels of hierarchy becoming ineffective, an issue with performance appeared. A complex approach to control is needed. This work is focused on researching the probability of making a hierarchical control system that allows choosing optimal methods and influence different system's elements on different levels, depending on current state of the system. A prediction of a probable workload of the system is also taken into consideration. The testing platform allowed conducting simulation experiments to determine threshold values that will influence system's state and to evaluate cost of control methods at certain levels of hierarchy. The received results show that it is highly probable to use the multi-leveled hierarchical control to achieve optimal performance level.**

I. Introduction

A performance control has been and will always be a very important part of every computing system. Especially this is crucial for systems that provide services to a lot of clients and even a small decrease in performance may, potentially, lead to short, almost uncontrollable, bursts of downtimes. For example, this is a common issue in cloud computing systems.

Modern cloud systems nowadays utilize complex system architecture and as a result have a lot of parameters for performance control. Multi-tenant systems are one of many applications of a cloud based architecture. A typical architecture for such systems has different services (applications) deployed in virtual machines. Those virtual machines in their turn are deployed in physical nodes. The architecture also contains performance optimizers that distribute incoming jobs to different services.

A lot of researches are centered around means of controlling different parameters of such systems. Virtual machines migration, for example, was researched in [1] and [2]. Even though they don't look at migration in scope of service-oriented architecture their ideas provide good foundation for further researches. In the first paper G. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu describe a way to optimize (not always minimize) time of live migration of multiple virtual machines. In the second paper T. Franco et al. present a research on seamless live migration of virtual machines over the Wide Area Network. As a result of their experiments a virtual machine was migrated between two countries with very low downtime.

Work [3] researched a way of paralleling the processing of several jobs without decrease in performance. In [4] a solution to multi-level scheduling is presented. Also, in [7] we described a way of controlling jobs distribution with use of threshold disciplines.

While all these works may help to significantly increase the overall performance, they focus on only one single parameter. And in each case there are areas definite areas for optimal usage of those methods. For example, threshold disciplines are effective only if jobs intensity doesn't exceed total performance of each processing application. Virtual machines migration supposes an existence of a time period when jobs intensity is constant after a few peaks. Simultaneous use of different optimization methods was not as widely researched [5].

An important area where a complex approach to performance control would be useful is service-oriented platforms with interacting services. The main and important distinction of such systems are unpredictable traffic changes: small and mostly insignificant changes in jobs stream intensity may lead to very significant changes in system's performance. This is due to complex distribution of secondary jobs while processing an original job by a sequence of applications.

II. Basic architecture of a system

Though every system that was researched in papers, mentioned before, is different, we can still base our architecture on their findings. In [8] we presented a typical model of a service-oriented system. This is our basis for the present work.

The system itself may be separated into few different parts (levels). There are five of them:

1) Physical level
2) Virtual level
3) Applications level
4) Jobs queuing level

5) Supplementary level

They compose a hierarchy of levels with only Supplementary level being outside of it. This level describes a set of additional applications that may be deployed on different levels themselves. The levels are represented on Fig. 1.

A. The Physical level

The Physical level is on the top of the hierarchy. This level is basically just a physical server that contains virtual machines hypervisors with virtual machines and applications.

B. Virtual level

Virtual level is the first step to jobs distribution. This level describes all virtual machines that have been deployed in it. These virtual machines may come with preinstalled software or they may be clean for an administrator or even user to install needed software manually. Virtual machines and applications storage is also included in this level.

C. Applications level

Applications level contains every application that was installed in a virtual machine. Each virtual machine has its own application level, making it a second step to jobs distribution.

D. Jobs queuing level

Jobs queuing level is the last step to jobs distribution. Every application has a jobs queue where jobs are being put when arriving to an application. This level also contains a common jobs queue. This queue holds every job that arrived before redistributing them to appropriate applications.

E. Supplementary level

Supplementary level holds monitoring agents, controllers and a knowledge base. Monitoring agents, depending on their purpose, may be put inside of an application, virtual machine or even physical server. That's why it is impossible to put this level in the hierarchy. Some elements from this level were researched in [9] as well as their use in real cloud systems. Thus, this level is crucial for proper functioning of optimization algorithm.

The controlling node always has a knowledge base and a monitoring agent inside. Since this node is a virtual machine itself, meaning that it lies in the Virtual level, it also has applications. Those are service applications that have no use in computational virtual nodes, and may not be deployed in those.

The service applications monitor the performance of the system and also hold common jobs queue and send a job to appropriate application. In general there may be a lot of such applications and their scope is not limited to just those two tasks.

Other two virtual machines are computational ones. They contain an application (or a few applications) that process incoming jobs and a monitoring agent. There is nothing else to prevent performance from degrading with time.

Every virtual machine, even the controlling one, has a controller inside. This controller takes next tasks:

- Receiving a job
- Deciding on which path a job must take
- Sending data to application
- Receiving data from application
- Sending a result

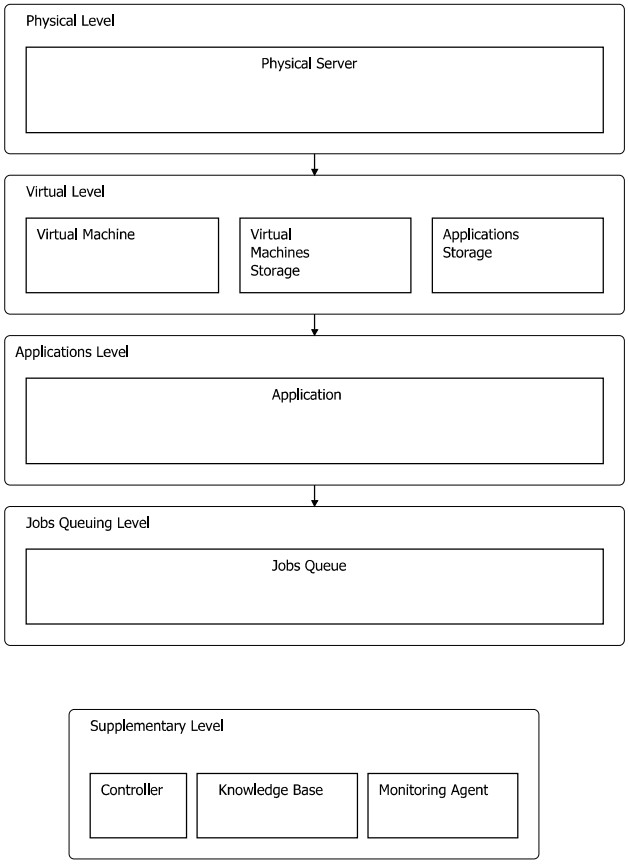

Fig. 1. Levels of the basic system

Those controllers are the only way for virtual machines to interact with each other. This was made to keep loose coupling of nodes and provide higher scalability.

Controllers are connecting with each other through a virtual switch.

The knowledge base is required for keeping information about existing virtual machines, applications and every other component of the system. To provide efficient update of information about system's elements, it holds data as RDF triplets and connects it into one ontology. However this is not the scope of this paper, so it will not be presented here. Detailed explanation of this approach is presented in [8].

III. JOBS QUEUES AND PATH FINDING

Since a virtual server may contain several applications, that means it can process different types of jobs at the same time. However doing so will significantly decrease the performance. That is the main reason a controller must choose the correct path for a job. Choosing the correct path is a non-trivial task that takes into consideration a lot of parameters such as current

performance of the node, current performance of a physical server, allocated node resources, allocated physical resources etc.

The performance, however, may change with time. That happens because of two main reasons: number of processed jobs may vary with time very drastically; placement of applications in virtual machines may not be optimal as well as placement of virtual machines in physical servers. As a result performance optimization is a complex process that is composed of a multiple tasks. In order to optimize performance we, according to [8] need:

1) Monitor performance and available resources of each server, virtual machine and application.
2) Quickly search for a path of job processing that will include a set of instances of applications.
3) Store knowledge about systems' objects and efficiency of decisions made earlier.

These tasks were solved in [7] and [8], but separately. Consolidating solutions of these tasks will give us a working algorithm for performance optimization and will increase the overall performance of the system.

The first task was solved by integrating intellectual agents into applications and virtual machines. They monitor performance of machines and send the data to the main controlling node. In order to keep performance on a proper level, those agents only send data when the changes were significant.

The second task was solved by implementing complex jobs. Those are jobs that may require several applications to process them in an order. Thanks to that we can try finding an optimal route for a job to take which will give the smallest decrease in performance.

The third and the last task provides us with information on which routes a job took earlier and for how long has it been processed. As a result we don't need to recalculate a path each time: we can assume that in a short period of time there were no significant changes and just use the same route.

IV. Architecture of the system

To find an optimal path for a job the system has to be in a stationary state for some time. However even slightest changes in jobs stream intensity may lead to big changes in performance and may demand some reconfiguration of the system to avoid downtime. Since the design is hierarchical, this process can happen on each level an can demand different complexity of algorithms for each of them.

As was said in section 2 the basic model of a system consists of five levels: Physical level, Virtual level, Applications level, Jobs queuing level, Supplementary level. All of them are necessary for system functioning.

We have our system as two physical servers with virtual machines hypervisors on it. These hypervisors deploy at least three virtual machines, with one of them being a controlling one and having the connection controller that transfers data. One of these two physical servers is the main one that holds database, knowledge base, and storages. It also has a main controller. Applications are sets of interpretators or runtimes and are deployed to different virtual machines. In our case a virtual machine with applications is a service.

Each virtual machine has an intellectual agent placed inside. These agents monitor the performance of a virtual machine and send this data to the main controller. As a connection controller computing virtual machines use Jobs Distributing Controllers that handle data transferring. All data is transferred through a virtual switch that connects every node and virtual machine into one local network.

The whole system is based on Xen Virtual Hypervisor. That means that machines use less processor time since they all use the same kernel of a host. We also use AMQP protocol to transfer data, as a part of the RabbitMQ framework.

The architecture is presented on Fig. 2.

V. The algorithm

The optimization itself may be separated into four different steps:

1) Jobs queuing step. During processing application have performance that changes with time. Having several copies of an application and monitoring its performance allows to achieve optimal average response time.

Advantages of this step are in low overhead costs and quick response to performance or jobs stream intensity change. However, the step is effective only when total performance of all applications is comparable to jobs stream intensity.

If we can predict when jobs stream exceeds the total performance and the period is long, we need to get to the next step.

2) Applications redistributing step. We can deploy another application in a virtual machine. Applications may also be redeployed to other virtual machines to decrease workload of a highly loaded virtual machine.

There are some limitations to this. Web-applications that process jobs may be developed using different technologies. For example a web-application may be a PHP script, a Python program or even a Java program. Those applications may also need a working database to keep track of used data. This restricts some virtual machines to use of only some certain applications by software versions, necessary dependencies, operating system.

The advantages are in ability to change total performance of applications and in a little higher that in the previous step but reasonable overhead costs. This step also has a quick response time, provided that there is applications storage.

There are disadvantages though. This step will be effective only if virtual machines can afford providing resources for all applications to work. It also demands a storage and preliminary creation of applications templates.

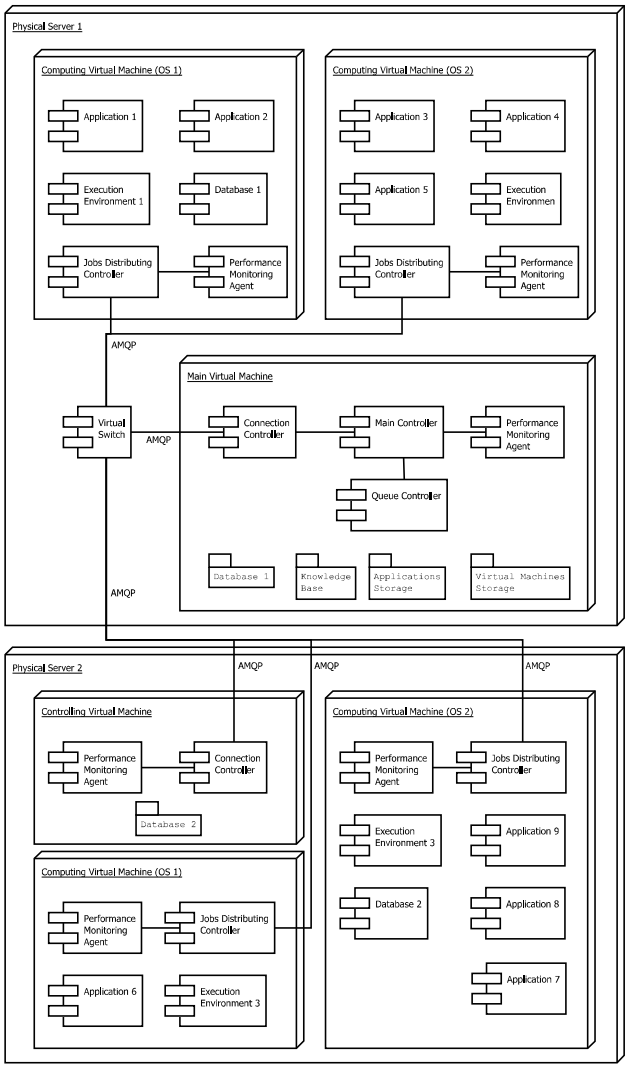

Fig. 2. Architecture of the system

3) Migration and creation of virtual machines. A migration to another physical server or a creation of a new virtual machine is required when all available resources are allocated and there is not enough of them to keep total performance on a reasonable level. This is mainly used when it is not possible to redeploy applications due to the limitations.

Overhead cost is significantly higher than before and the performance will always be a little lower due to another virtual machine getting processor time (this is not the case when a virtual machine is migrated to another server). However this step drastically increases ability to control performance level by providing additional resources and is limited only by physical resources of a system.

4) Starting or stopping physical servers. When no other step helps a last attempt to control performance is to start a new server or to stop a server that doesn't do anything at the moment. This will either use free physical resources to increase total performance of applications or free resources for other servers to use.

The activity diagram of all four steps is presented on Fig. 3.

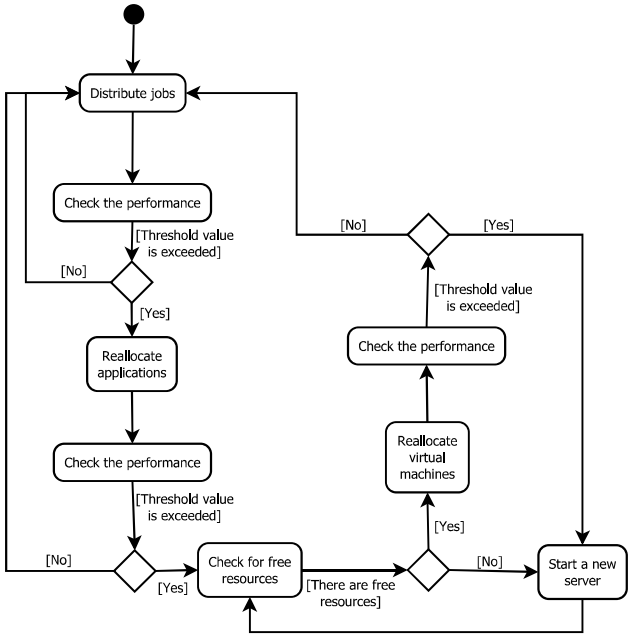

Fig. 3. Activity diagram of the applications reallocating algorithm

Again, the first step already was researched by us in [7] and has shown that in most situations it increases the overall performance. The third step is too complex to include it in this work, so instead we focus on consolidating steps one and two. We already have a threshold discipline from [7] and in this work we will describe an algorithm to find an effective new placements for applications.

This time, however, the first step is updated and extended with use of the second step and composite jobs. Here, we have a jobs queue that is filled with jobs, coming from users. This queue is used in jobs distribution: the system decides which applications may process any jobs and sends them there. But it does so only if a certain threshold value was not exceeded by response time.

The second step demands another threshold value. In our previous work the system just stopped sending jobs to one application when a certain value was exceeded and began sending them to another. This time, when the last threshold value is exceeded, applications redistributing begins. Search for an appropriate application is decomposed into these steps:

- Determine an application with lowest performance.
- Determine a virtual machine with highest performance.
- Determine a virtual machine with maximum free resources.
- If those virtual machines are the same machine: check if there are enough resources to reallocate application and if this virtual machine can run the application. If it doesn't then do steps 2 through 4 for virtual machine with second to highest performance and with second to maximum amount of free resources.

- If an appropriate virtual machine was found: reallocate the application.

The activity diagram of this algorithm is presented on Fig. 4.

VI. Experiments

To check the efficiency of the algorithm we conducted three types of experiments. The first is similar to one we already did in [7]: we checked the performance of the system when using the first step of the optimization. This is needed to compare results of the next experiments. The gathered values represent response time changes at each iteration.

The second type of experiments is experiment where we only reallocate applications to optimize the performance. The gathered values should have peaks at times of reallocation but essentially should be the same as at the previous step. The jobs queuing in this case is disabled: as soon as a job is received, it's sent to a first free application that can process it.

The last type consolidates two other. To implement a change from one step to another, we have a threshold value. When this value is exceeded, the optimization step is changed to applications reallocating.

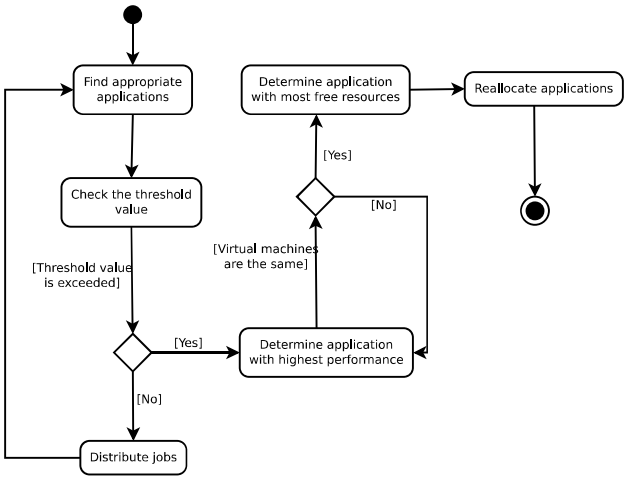

Fig. 4. Activity diagram of the applications reallocating algorithm

To be able to compare the experiments, all three of them had the same parameters: each experiment was conducted 100 times. During the experiments 1000 jobs was sent to the system with 3 seconds intervals. Then we smoothened up the values by using *Moving Average Method*, using all 100 iterations. The number of iterations should give us good smoothened values since there are a lot of input values. In [10] we used 10 and even then the averaged value was showing the dynamics of the chart without too many fluctuations. Having more iterations will only further smoothen the values.

As we can see at Fig. 5, with threshold discipline the response time slowly grows as there is not enough resources for all jobs to be processed (here N is the number of a job). However the performance is optimized to the point where this growth is not as fast as it could be with conservative discipline or no disciplines at all. At this point the response time grows up to almost 1 millisecond.

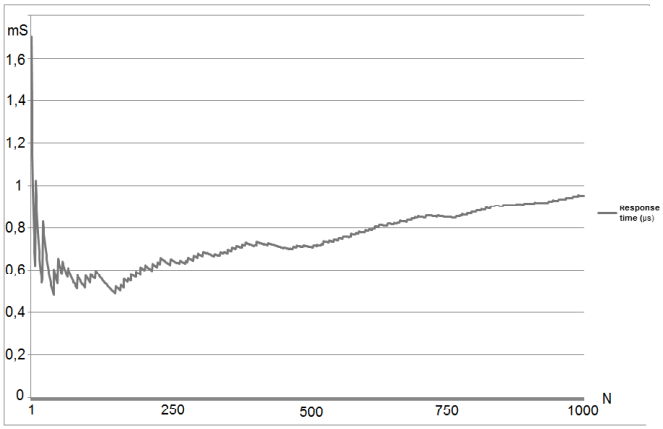

Fig. 5. Experiment results with threshold discipline

On Fig. 6 we can see that as soon as response time exceeds the threshold value reallocating of applications begins. There are two peaks but after those the performance increases for a while, before slowly decreasing again. These short peaks occur at the moment when applications are being reallocated since this process demands additional resources. The response time is seen to be quite high at the end of the experiment: about 0.6 millisecond.

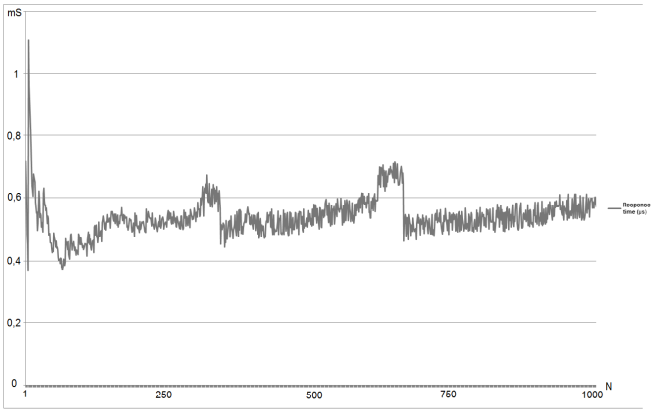

Fig. 6. Experiment results with application reallocating

Fig. 7 shows improved performance when using both methods. At the end of experiment the response time was much lower: about 0,5 milliseconds. There are still peaks when applications are being reallocated, however, the performance does not decreases so fast. During the experiment it was smooth and response time increased very slowly.

Conclusion

Control of an open cloud platform to provide optimal performance level is a complex task. Such system has a multi-leveled hierarchical architecture: elements on different levels contest for common resources, and that influences other elements that may not need additional resources at the moment.

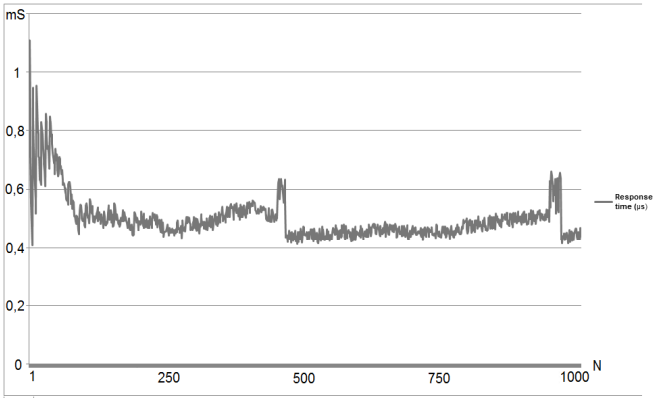

Fig. 7. Experiment result with threshold discipline and applications reallocating

In this work we updated our existing algorithm for performance optimization in such systems by adding another step. The resulting algorithm was tested in simulation experiments and has improved system's performance compared to the use of jobs queuing exclusively. An example of architecture for use with the algorithm was also presented.

The future researches lie in implementing two last steps of optimization, further increasing performance in cases of high jobs stream intensity. However even now the proposed algorithm shows its potential.

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. Sun, D. Liao, V. Anand, D. Zhao, and H. Yu, "A new technique for efficient live migration of multiple virtual machines", *Future Generation Computer Systems*, vol.55, Feb.2016, pp. 74-86.

[2] F. Travostinoa, P. Daspitb, L. Gommansc, C. Joga, C. de Laatc, J. Mambrettid, I. Mongaa, B. van Oudenaardec, S. Raghunatha, and P. Y. Wange, "Seamless live migration of virtual machines over the MAN/WAN", *Future Generation Computer Systems*, vol.22, Oct.2006, pp. 901-907.

[3] X. Liu, Y. Zha, Q. Yin, Y. Peng, and L. Qin, "Scheduling parallel jobs with tentative runs and consolidation in the cloud", *Journal of Systems and Software*, vol.104, Jun.2015, pp. 141-151.

[4] A.J. Rubio-Montero, E. Huedo, F. Castejón, and R. Mayo-García, "GWpilot: Enabling multi-level scheduling in distributed infrastructures with GridWay and pilot jobs", *Future Generation Computer Systems*, vol.45, Apr.2015, pp. 25-52.

[5] L. Chunlin, and L. Layuan, "An agent-oriented and service-oriented environment for deploying dynamic distributed systems", *Computer Standards & Interfaces*, vol.24, Sep.2002, pp. 323-336.

[6] 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.

[7] 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.

[8] D. A. Zubok, T. V. Kharchenko, A.V. Maiatin, and M. V. Khegai, "Ontology for Performance Control in Service-Oriented System with Composite Services", *Knowledge Engineering and the Semantic Web*, vol.519, 2016, pp. 42-55.

[9] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, T. Varvarigou, "A Self-adaptive hierarchical monitoring mechanism for Clouds", *The Journal of Systems and Software*, vol.85, 2012, pp. 1029-1041.

[10] D. A. Zubok, T. V. Kharchenko, A.V. Maiatin, and M. V. Khegai, "A Multi-Agent Approach to Monitoring of Cloud Computing System With Dynamically Changing Configuration", *in Proc. FRUCT Conf.*, 2016, pp. 410-416.