A Multi-Agent Approach to the Monitoring of Cloud Computing System with Dynamically Changing Configuration

Dmitrii A. Zubok, Tatiana V. Kharchenko, Aleksandr V. Maiatin, Maksim V. Khegai
Saint Petersburg National Research University of Information Technologies, Mechanics and Optics
St. Petersburg, Russia
zubok, kharchenko@mail.ifmo.ru, mavr.mkk@gmail.com, MaxHegai@rambler.ru

Abstract—**Cloud based distributed systems rely on scheduling and resources allocation to function. In complex distributed systems a distribution of many jobs of different types is required. At the same time, a problem of virtual machines migration to physical servers must be solved. Therefore, configuration of a cloud system may be very dynamic, meaning that not only number of existing computational servers but also their location on physical servers might change. Optimal control strategies aimed to solve these problems are effective only when updated information about system's components is available. However, gathering this information from many distributed components of a cloud system, such as physical nodes or virtual machines may significantly decrease overall performance. These problems can be solved by applying different optimization techniques such as multi-agent approach. Agents decide if the information is outdated and needs to be updated by them. This paper describes a cloud system architecture that uses agents of different types. Agents' algorithms and their interaction schemes are defined. Software implementation in form of software environment is presented. Simulation experiments to compare performance of the system when using default monitoring methods and a multi-agent approach were conducted.**

I. INTRODUCTION

Performance and reliability of cloud computing systems has always been a reoccurring topic for researches. Since the first cloud system appeared in 2000s, a lot of techniques were implemented to ensure that downtime of a system stays as low as possible. Lately an idea of a multi-agent approach to monitoring appeared and was researched numerous times. Its main goal is to help keeping information about frequently changing (uncertain) environment up to date and respond to these changes according to their algorithms. In [1] the approach was used in smart grid building. Cloud architecture is used as a base, and agents are used to check status of the system and to coordinate the control of each node.

Work of Mauro et al [2] applies multi-agent systems to evaluating of reliability assessment of power systems. They use simulation to improve reliability of a system and demonstrate the potential of multi-agent systems. However, although agents perform evaluation and optimization, they are not applied to mass-service systems which are the scope of this paper.

In [3] an agent is defined to have next characteristics:

- Autonomy: in the absence of external intervention, through their knowledge or perception of the external environment, they can independently control their own behavior to accomplish certain tasks;
- Collaborative: different agents can collaborate together to complete the task or the other complex problems, provide information to other agents when needed, this feature is especially suitable for solving of distributed problems;
- Study: they take the initiative to learn and get information from external environment in order to constantly revise their own database, which makes them more reasoning and planning.

The multi-agent system includes methods from different disciplines, such as artificial intelligence and distributed computing. This approach as it is, however, may lead to a huge overload of a system and, thus, needs optimization. One such way is, for example, decreasing polling rate and distributing tasks between agents. Optimization is especially important when the multi-agent approach is applied to performance monitoring. The presented optimization method consists of swapping polling with on-demand data updating. That allows significant decreasing of overhead expenses, increasing overall performance of a system.

In recent years most frequently optimization techniques were created for cloud based computing systems. Cloud based computing is a kind of computing where resources and data are shared and provided on-demand. Typically a cloud is built as a data center and has a main controlling server (master server), a computational server and a transmission server. During its work the cloud may have a huge overload that will decrease its performance. In this case optimization techniques are used which, essentially, try to minimize number of allocated resources [4]. A simple technique involves constant polling of all computational servers in order to get up to date information about their performance, and alternate resources quotas according to this information [5].

II. A MULTI-AGENT APPROACH

A more complex technique is scheduling that uses different service disciplines such as conservative or threshold discipline [6]. Those disciplines have the same base algorithm but behave differently at the later stages. The conservative discipline receives a job and as soon as the controlling server finds the first free server, sends it for processing. The threshold discipline, instead, after receiving a job adds it to a queue of a computational server. When number of jobs in this queue becomes higher than a threshold value, it begins to search for a next computational server with free space in its queue. The threshold values are calculated based on the current performance of the system.

However, for the technique to work, additional parameters are required. Thus, as a part of scheduling, the constant polling is also presented. The reason is that required parameters are non-constant. Incoming jobs stream rate changes with time and varies randomly, with slight tendencies depending on time of day and other factors. Time needed to process jobs is also non-constant and can't be acquired at the time of job receiving. Processing of jobs with different processing time decreases the overall performance of a computational server so the controlling server must get information about performance to properly distribute load. At the same time resources quotas may change in time as well, whether because of a resource management system or an administrator. All this leads to the conclusion that in case of scheduling constant polling is essential for proper functioning of the method as updated information about resources and performance is essential.

The main disadvantage of constant polling is a high load on CPU. Continuous polling must be done from a controlling server. While it isn't involved in computing, coordinating and control is done there, so it is equally important as computational servers. The on-demand data updating completely avoids this problem by not doing anything but simple calculations when the data doesn't need to be sent. This is achieved by placing an agent in each computational server and in the controlling server.

The proposed approach is based on decrease of monitored objects' load by placing an intellectual agent in each of them. Instead of performing a constant polling objects are monitored by those agents that decide if the changes were big enough and information about this should be made available for other controlling components. The infrastructure gives ways to subscribe new agents to information channels and information exchange between controllers and data storage. The agents are different for each type of system objects and have the same basic algorithm but differ in monitor and decision making algorithms. This way a multi-agent approach to monitoring also provides scalability and monitoring of a constantly changing system. The basic algorithm of a monitoring agent is shown on Fig. 1 and is explained further.

An agent is normally in stand-by mode, waiting for a data to be received or sent. When created, it subscribes to changes in information it has to check. Then it receives current threshold value for its computational server and enters the interrogation state when it gathers information about the server. At the same time it receives messages from the main controlling server and if it gets a signal to change threshold value does so. During the interrogation agent applies filtering algorithms to decide if the changes in parameter were big enough for the new value to be published. If they were, agent publishes the information and enters the interrogation state again. At the end of its work, agent unsubscribe from changes in information and closes. Those tasks are simple enough to not reduce server's performance.

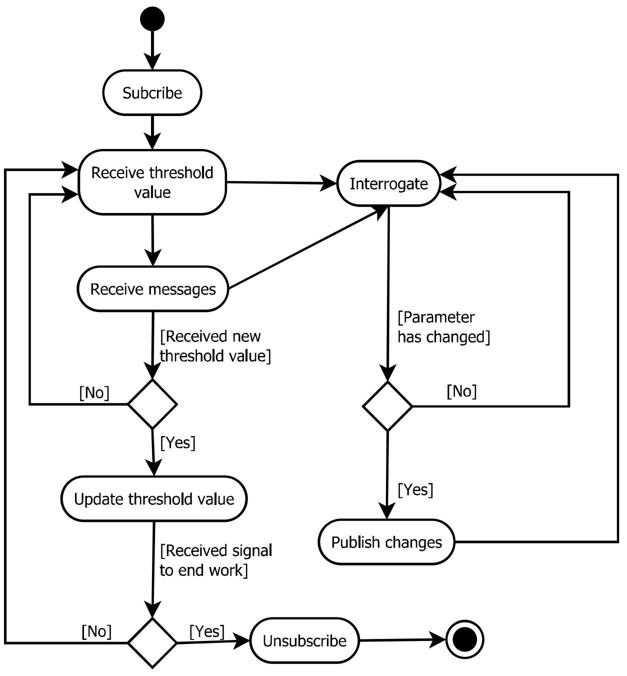

Fig. 1. Activity diagram of the basic agents' algorithm

To test the multi-agent approach a software environment is required. Next section describes the environment built as a mass-service system simulator.

III. SOFTWARE ENVIRONMENT

A. Virtual machines monitor

The developed software environment is based on Xen hypervisor – a hypervisor using a microkernel design. Xen supports running several virtual machines at the same time as well as sharing resources between them. Migration of virtual machines is also possible. All that allows separating different subsystems from each other, and quickly swapping them with other subsystems, providing a strong base for a module based architecture. Xen is often used in scheduling analysis and as a virtual machines monitor for cloud based systems [7], [8].

Each subsystem is a virtual machine with its own parameters and set of software, connected with each other through a single virtual interface that works as a router.

Shared resources pool ensures that the system is able to dynamically reallocate resources between virtual machines, providing basic means to control performance of the environment.

B. Communication

Communication between different subsystems is crucial for functionality of the environment. Since virtual machines may be located on different physical nodes, an efficient way to exchange messages through internet connection is required. The current architecture uses a message broker, a message-oriented middleware as a way to exchange information between different subsystems. A message broker is responsible for receiving and sending messages between other instances of itself.

In [9] middleware is defined to have next characteristics:

- Prioritization of messages;
- Protection of messages from loss and reordering;
- Support for large user base;
- Adaptability to user changes;
- Low transmission latency;
- Low latency variability;
- Large data throughput;

The most important characteristics for the environment are: protection of messages from loss and reordering, aadaptability to user changes, low transmission latency and low latency variability. To provide them RabbitMQ was used.

RabbitMQ is an asynchronous message queuing framework that works with many protocols and is optimized for high performance message handling. This framework was successfully used in [10] as a message broker. In [11] a comparison between ActiveMQ and RabbitMQ was made and the superiority of RabbitMQ was proven. As an exchange protocol to communicate with every subsystem the software environment uses Advanced Message Queuing Protocol (AMQP) which focuses on a message-oriented communication and encryption of messages. The current specification (AMQP 1.0) provides needed features; particularly, a real-time feed of constantly updated information. Thus, the AMQP protocol and RabbitMQ are appropriate for communication between agents.

C. Architecture description

Each virtual machine is limited in its resources. Maximal CPU usage and maximal RAM amount are set during creation of a machine and can be changed later if such a need appears. Virtual machines may be hosted on different physical servers but they are still in the same virtual network, having access to each other, provided that they were allowed to interact with other virtual machines. In general a virtual machine is a subsystem.

There are two types of subsystems:

- Controlling subsystems
- Computational subsystems

Controlling subsystems control the behavior of different parts of the environment and each of them normally has at least one corresponding computing subsystem. The software environment relies on this type of subsystems to exchange required data from one subsystem to another.

Computational subsystems compute data that incomes from a corresponding subsystem and return a result. These are the main computational subsystems and are essential to the purpose of the environment: simulation experiments using distributed computing.

If there is more than one physical server in the environment, one of them takes the role of a main server, controlling others. The main server contains one extra controlling subsystem that distributes jobs to computational subsystems and decides if they need to be sent to another physical server. If there is only one physical server this server becomes the main one.

The architecture is presented on Fig. 2.

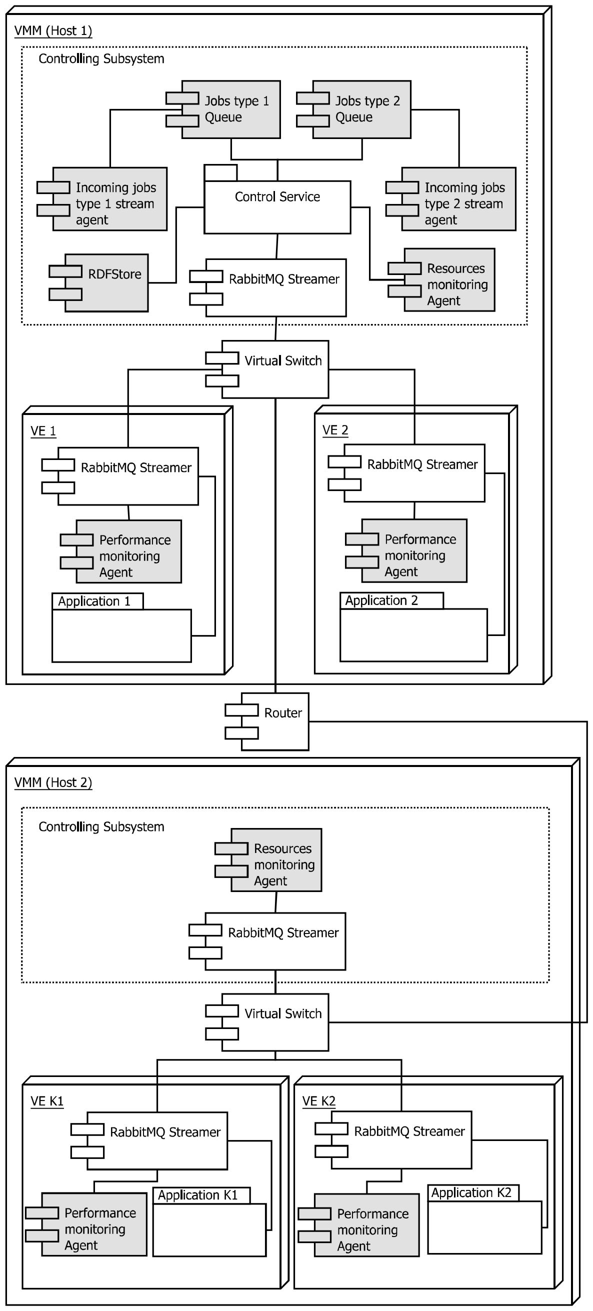

Fig. 2. Multi-agent system architecture

D. Subsystems

Each controlling subsystem has an agent, a jobs queue, a producer that sends jobs and other data to several subsystems and a consumer that receives data from subsystems. One of the subsystems that it sends data to is the main controller. The main controller has a common jobs queue where jobs await for decision, which computational subsystem a job should be sent to, to be made. There may be more than one computational subsystem corresponded to a controlling subsystem.

A generic computational subsystem includes an agent, a producer that sends jobs and other data to several subsystems, a consumer that receives data from subsystems and an application that processes received data and returns a result. A computational subsystem has one single corresponding controlling subsystem.

In the software environment agents are background processes that perform their own task. Each subsystem has its own agent distinct by their task:

- Computational subsystems have performance monitoring agents that evaluate performance of a subsystem it works on.
- Controlling subsystem has incoming jobs stream monitoring agent. This agent evaluates incoming jobs stream intensity.
- The main server has a resource monitoring agent. It gets information about resources quotas and their changes if they were changed by an administrator.

E. Agents

This subsection describes behavior of the agents that exist in the software environment. Every agent has the same base for the algorithms - independent decision making. Each agent decides for itself if information it has should be posted or not.

1) Performance monitoring agent: The agent that monitors the performance checks a computational server's performance at regular intervals and decides if it should be sent to the controlling server. The performance is evaluated by using moving average method [12]. This method smoothens up fluctuations and allows defining a trend in performance changing. If the average value exceeds a threshold value the agent send information about current performance of the server to the controlling server. In other case it just continues the evaluation. Fig. 3 shows the algorithm.

Using the calculated average value different parameters of a system, such as maximal CPU time, are changed.

2) Incoming jobs stream monitoring agent: This agent monitors rate at which jobs income to the queue. Its main goal is to check if there is a free computational server that is able to process the job and if there is no such server waits for its appearance. If there is a free server, the agent begins to evaluate incoming stream rate and if the average number of jobs was different from the last one for a certain period of time, sets it as a new rate. Since the agent is placed in the main controller, there is no controlling agent. The agent's algorithm is presented at the Fig. 4.

3) Resources monitoring agent: This agent gathers the information about current resources quotas. Those are allowed CPU time for each computational server, memory usage and allowed free space on hard drive. If any quota was changed it saves this information for a future use in jobs distribution.

Each agent is also able to update their threshold values by receiving data from the main controlling server. To provide scalability and flexibility ontology based data storage was implemented. ARC2 RDF system was used to create and manage RDF triplets and the data itself is stored in a database. Since the multi-agent approach doesn't suggest a frequent update of data, on-memory database will not give us any gain in speed. Thus, a relational database will be sufficient.

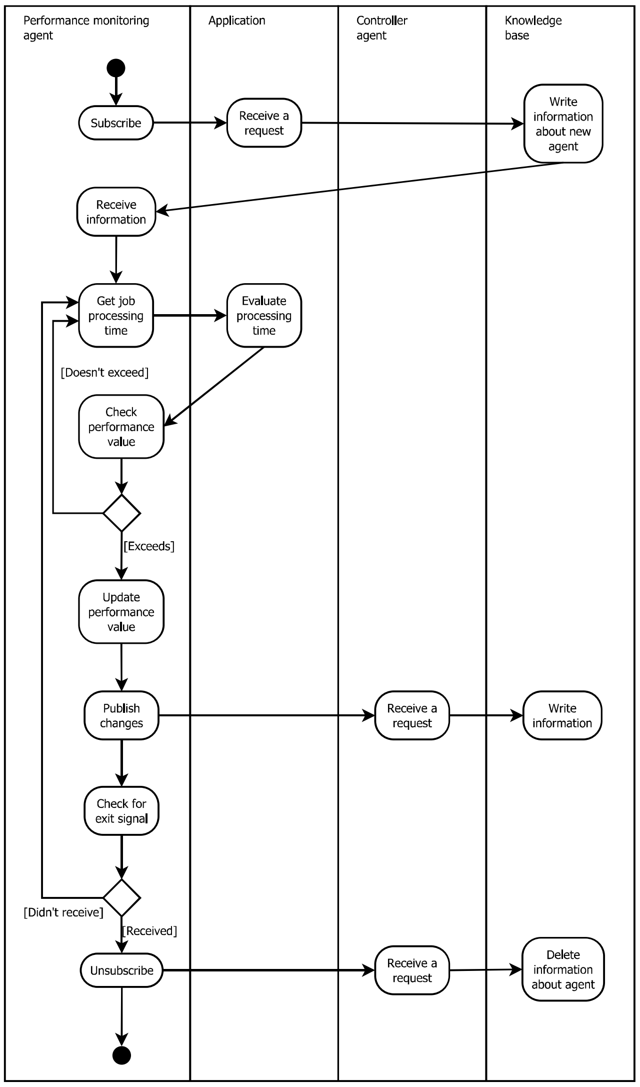

Fig. 3. Performance monitoring agent's algorithm

F. Software environment and algorithms description

The current software environment consists of one physical server, one main controlling server (virtual machine), two controlling servers and two corresponding computational servers. The basic algorithm is next: the main controlling server generates a job (in a real-life scenario it is done by a user), then the job is added to the common jobs queue. After that the system applies a service discipline to the queue and according to it distributes jobs between computational servers. As soon as a job is processed, computational servers sends an answer to let the main controlling server know that it is done.

The servers were sorted by performance level: highest to lowest. Currently there is only one parameter that characterizes this: maximal CPU usage (CPU quota).

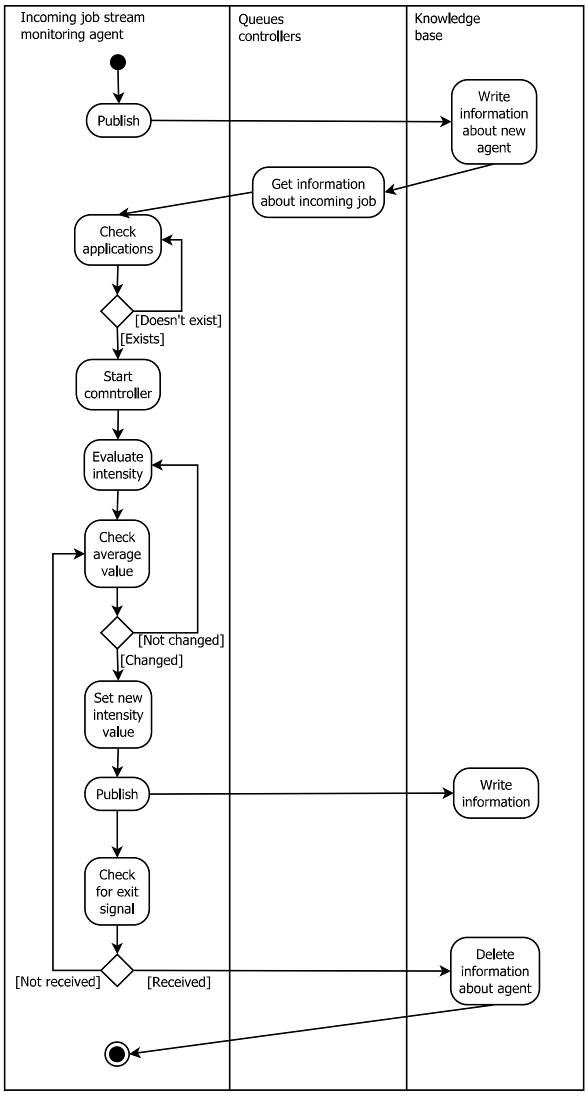

Fig. 4. Incoming jobs stream monitoring agent's algorithm

Software environment has two computational servers. Each of them has a memory quota set to 256 mb of RAM. The CPU time quota for the first server was set to 55% of the hardware CPU and the second one was set to 25%. The controlling server has a CPU quota set to 20%. In the next section the results of experiments are presented.

The conducted experiments were aimed to find out which initial parameters of a system allow increasing the performance by using the described approach.

IV. Simulation experiments

A. Experiments description

To evaluate the performance cost of multi-agent approach three experiments were conducted with both existing computational servers used. Two of them use the polling algorithm: the main controlling server constantly interrogates every computational server and gathers required information. Two different time intervals were selected: 1 second and 10 seconds. The third experiment used the multi-agent approach.

In all three experiments jobs were generated by the main control script. To simulate a real–life case, intervals between jobs generation were randomly distributed by the Poisson's law based on average jobs per second value. There also were some variations. The whole time, during which the experiments were conducted, was separated into four periods. Intervals during those periods were different with the highest (and lowest jobs rate) being at the last one and the lowest (the highest jobs rate) being at the third one.

When a job is generated and sent to a server, time when it happened is saved. Computational servers receive jobs and run a script that simulates load on the system. It requires a sufficient CPU time to be processed and its processing time is guaranteed to be at least slightly different with each run. After the job was processed, a signal about this event is received by the main controlling server and the finishing time is saved. The processing time is calculated based on these values.

Experiments were conducted 10 times each and each run lasted for 1000 jobs. As soon as the 1000th job was processed, one run was stopped and the next one began.

B. Experiments results

Results of each run were saved to different text files. After each experiment values from those files were averaged and saved to another file. This resulted in 1000 values that represent average processing time of each job that was sent to a computational server in microseconds (µs). Then the results were filtered by calculating average for every three values.

The runs using constant polling have shown that processing time slowly but surely increase, as a run continues. This is due to the fact that because there is a high overload on the system, less and less jobs are actually being sent to be processed. This concludes that high frequency polling significantly decreases performance. The results for the first experiment are shown on Fig. 5, where *n* is a number of an experiment in a run.

Fig. 6 shows results of the experiments with polling frequency set to 10. As can be seen the system is stable and keeps performance on a higher level than when the frequency is set to 1. *n* is a number of an experiment in a run.

The last experiment was conducted using agents. Fig. 7 shows that at first there was a short increase in processing time however it then decreased and kept being stable until the end of the experiment. The average processing time was also lower than in any previous experiment which proves that the offered approach is more effective. *n* is a number of an experiment in a run.

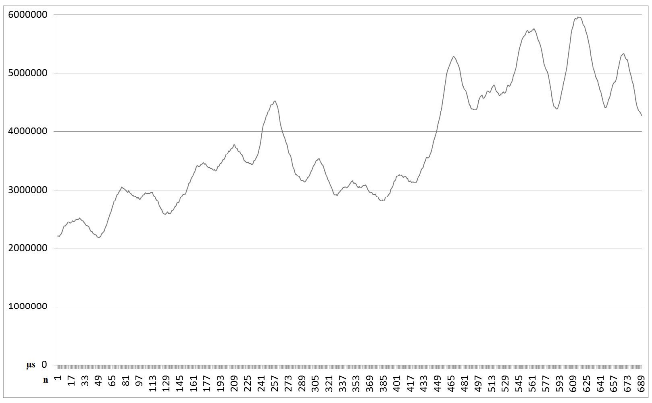

Fig. 5. Processing time with polling (1 sec. interval)

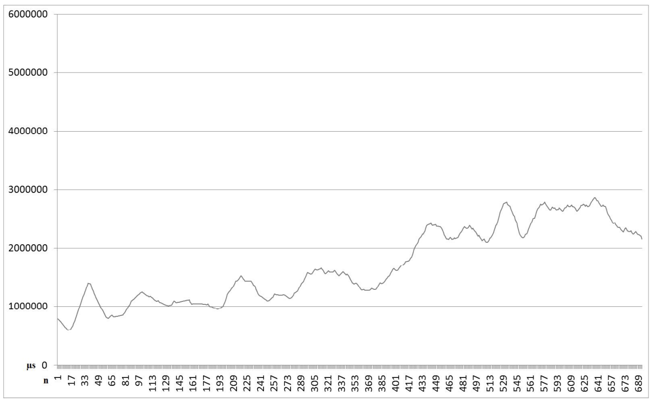

Fig. 6. Processing time with polling (10 sec. interval)

Table I shows the comparison of the average processing time in each experiment.

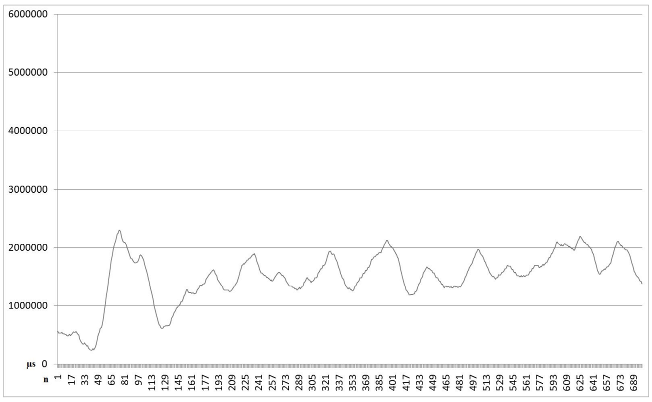

Fig. 7. Processing time with multi-agent approach

TABLE I. AVERAGE PROCESSING TIME OF JOBS

Experiment	Average processing time (µs)
Polling (1 sec. interval)	3685955.08
Polling (10 sec. interval)	1708525.6
Multi-agents approach	1429489.32

V. CONCLUSION AND FUTURE WORK

Effective control of complex cloud based distributed systems requires an updated information about distant nodes state. When constant polling is in action, shared resources are used, disallowing big part of it to be used by computational servers. The presented approach makes it possible to keep the required information up to date while keeping overheads at their minimum. The base of this approach is decision making agents that evaluate different parameters and send the information only when the changes are radical. The software implementation provides a good flexibility and scalability of the system.

Conducted experiments proved advantage of the offered approach in comparison with polling of servers when the incoming jobs stream rate is dynamic. The results characterize a system with independent applications, processing jobs. The proposed approach, however, has drawbacks. The first one is detection of a failure in components functioning. Unlike when polling is used, in case of using intellectual agents it is impossible to detect that one of components stopped responding. This raises the difficulty of using this approach in case of unstable functionality of components. The solution to this is in raising complexity of the main controller algorithm, making it possible to detect unusual pauses during the processing and receiving data from agents. The second problem may occur after heavily increasing number of components and, as a result, intensity of data streams that are to be sent to the knowledge base. In this case using a relational database management system may become a bottleneck. The solution is in shifting to a database management system that uses key-value pairs or to a distributed database. In the next work agents' algorithms will be improved and a possibility to apply this approach to interacting applications will be researched. Other service disciplines will be used and the approach will be improved to decrease time until a system enters the stationary state.

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] X. Jin, Z. He, Z. Liu, "Multi-Agent-Based Cloud Architecture of Smart Grid", *in Energy Procedia, volume 12,* 2011, pp. 60-66.

[2] M. A. da Rosa, A. M. Leite da Silvac, V. Miranda, "Multi-agent systems applied to reliability assessment of power systems", *in International Journal of Electrical Power & Energy Systems, volume 42,* Nov. 2012, pp.367–374.

[3] S.D.J. McArthur, E.M. Davidson, V.M. Catterson, A.L. Dimeas, N.D. Hatziargyriou, F. Ponci, T. Funabashi, "Multi-Agent Systems for Power Engineering Applications-Part I: Concepts, Approaches,

and Technical Challenges", *in Power Systems, volume 22,* Nov. 2007, pp. 1743 - 1752.

[4] L. Donatiello, R. Nelson, "Performance Evaluation of Computer and Communication Systems", *Springer-Verlag,* 1993, pp. 630-650.

[5] A. Quarati, D. D'Agostino, A. Galizia, M. Mangini, A. Clematis, "Delivering Cloud Services with QoS Requirements: An Opportunity for ICT SMEs", *in Economics of Grids, Clouds, Systems, and Services, Volume 7714,* 2011, pp. 197-211.

[6] D.A. Zubok, A.V. Maiatin, V.E. Kiryushkina, M.V. Khegai, "Functional model of a software system with random time horizon", *in Proceedings of the 17th Conference of Open Innovations Association FRUCT,* 2015, pp 259-266.

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

[8] D.C. Marinescu, *Cloud Computing*. Elsevier, 2013, pp 131-161.. Albano, L.L. Ferreira, L.M. Pinho, A.R. Alkhawaja, "Message-oriented middleware for smart grids", *in Computer Standards & Interfaces*, 2015, pp. 133-143.

[9] S. Satunin, E. Babkin, "A multi-agent approach to Intelligent Transportation Systems modeling with combinatorial auctions", *in Expert Systems with Applications, Volume 41*, 2014, pp. 6622 -6623.

[10] A. Aizstrauts, E. Ginters, M. Baltruks, M. Gusev, "Architecture for Distributed Simulation Environment", *in Procedia Computer Science 43*, 2015, pp. 18-25.

[11] K. Mivule, C. Turner: Applying Moving Average Filtering for Non-interactive Differential Privacy Settings. In: Procedia Computer Science, Volume 36, 2014, pp 409–415.