Using Alternating Decision Trees in Multi-Leveled Hierarchical Cloud Based System

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

***Abstract*—Cloud platforms are an essential part of modern world. Used in all kind of fields, from education to science, they became inseparable from information technologies and computing sphere. However the problem of performance optimization still exists and is not entirely solved even today. With introduction of machine learning and artificial intelligence this task became easier to solve. This paper presents a way to utilize decision trees to control performance of a computational system and to balance load on different nodes in attempt to increase quality of service.**

I. Introduction

Since its appearance cloud computing has been a very perspective direction of information technologies studies. Most often they are being used for service-oriented platforms due to easy deployment and control. However the architecture of such systems becomes more complex and a problem of performance control still exists. Evaluation of values necessary to solve this problem is difficult and can't be solved analytically. A multi-leveled hierarchical system is an attempt to separate system's entities into different levels and optimize them one-by-one. In [10] a prototype of such a system was presented, however the decision making algorithm was still not clearly determined. Recent researches show that machine learning gives good results when making a decision based on many different parameters.

Machine learning has been around for a very long time. The idea of a system that can change its behavior based on implicitly gathered external data is an attractive one. Having a system that can learn by itself is really convenient.

Such systems found their use in data processing and predictive analysis. Proposed and implemented algorithms largely evolved since their first appearance. Lately they are being used to solve optimization problems, making existence of autonomous high-performance computing systems possible.

As of today there are two popular approaches to machine learning commonly used:

- Genetic algorithm
- Neural network

The main idea of the genetic algorithm is survival of the fittest. In the algorithm we first decide which criteria for "fitness" will be used in selection step. Then random genomes, sets of data, are being created and then tested for fitness. The fitting one "survives" and creates so-called "offsprings" that later replace this genome. "Offsprings" may have "mutations", changed chunks of data in genome so they wouldnt be completely the same. Then the process repeats. "Mutation" rate is not fixed and may be adjusted in attempts to boost the algorithm performance as shown in [6].

This algorithm may be seen as an optimization algorithm in a way that it searches for the most optimal choice among all others. This has been proven by works that used the genetic algorithm. For example in [1] it was used to find the optimal placing of nodes for wireless sensor network.

Neural networks are loosely based on a real biological neural network and as such are extremely flexible. They dont have any prior knowledge about objects they research but gather data and develop their own set of parameters during work. Nowadays they are mainly used in image recognizing but there are other fields they are applied to. For example in 1996 a neural network was used to predict academic performance [2]. A more recent example is [3] where a network was used to forecast electricity demands.

Machine learning usage in performance optimization, however, demands a specifically designed architecture. In [8] we presented a basic architecture of our testing system, consisting of several virtual machines connected by a single virtual switch. In [11] an efficient monitoring system was presented. Based on intellectual agents this system instead of a classical real-time polling approach used simple background applications that were gathering data about virtual machines performance and sending it only when performance exceeded a threshold value. The architecture was upgraded to make use of this system and as a result, a knowledge base and ontologies were introduced.

With all that in mind an architecture was designed to make use of each and every system made by us and to take scalability and flexibility into consideration.

II. Architecture

The architecture itself is composed of several different layers; each having its own optimization technique. In total there are three main distinct levels with two additional nominal levels. The main levels are:

- Physical level
- Virtual level
- Applications level

The additional levels are:

- Jobs queuing level

- Supplementary level

A. Physical level

This level includes any physical server added to the system. Each of them contains virtual machines hypervisor, virtual machines, data storages and applications. This is the top of the system hierarchy and is represented by **physical servers**.

B. Virtual level

Virtual level is the next one in the hierarchy. Every virtual machine is added to this level, as soon as it is deployed. These machines may have any set of software packages whether preinstalled or clean. There is also application storage for automated deployment of applications. This level is represented by **virtual machines** and storages.

C. Applications level

This level contains applications that were deployed in a virtual machine. This level is essentially a sub-level, since every virtual machine has one. This level is represented by **applications**.

D. Jobs queuing level

This level is used in jobs distribution and is not really determined as a different layer. It contains job queues and a set of rules for distribution.

E. Supplementary level

This level is not determined as well and only holds monitoring agents, controllers and a knowledge base. The level itself is crucial only for the optimization algorithm and not for functionality of the system as a whole.

The current architecture is presented on Fig. 1.

The load distribution algorithm, as presented in [10] is depending on parameters gathered by monitoring agents. The main problem here is number of those parameters: due to scalability we cant predict how many parameters each type of agents will gather. While we can determine this from agents configuration sent to the controlling server, the system should be able to use this data with little to none interaction from user. In other words the optimization algorithm should be able to adapt itself to a new set of parameters.

To make this task easier there are different algorithms for each level of the system. There is typically one single monitoring agent for each node on levels. For example there is one agent for each virtual machine. This allows us to compute one performance value per node based on data gathered by agent.

The algorithms for each level were presented in [8] and jobs distribution and virtual machines migration algorithms performance was researched in [10]. Although performance levels were high this could be further improved by implementing some sort of artificial intelligence that will decide when and on which level should an optimization be performed. The difference from the previous approach would be that AI will be learning, constantly changing parameters by itself, based on gathered knowledge of systems behavior and state at some points in time.

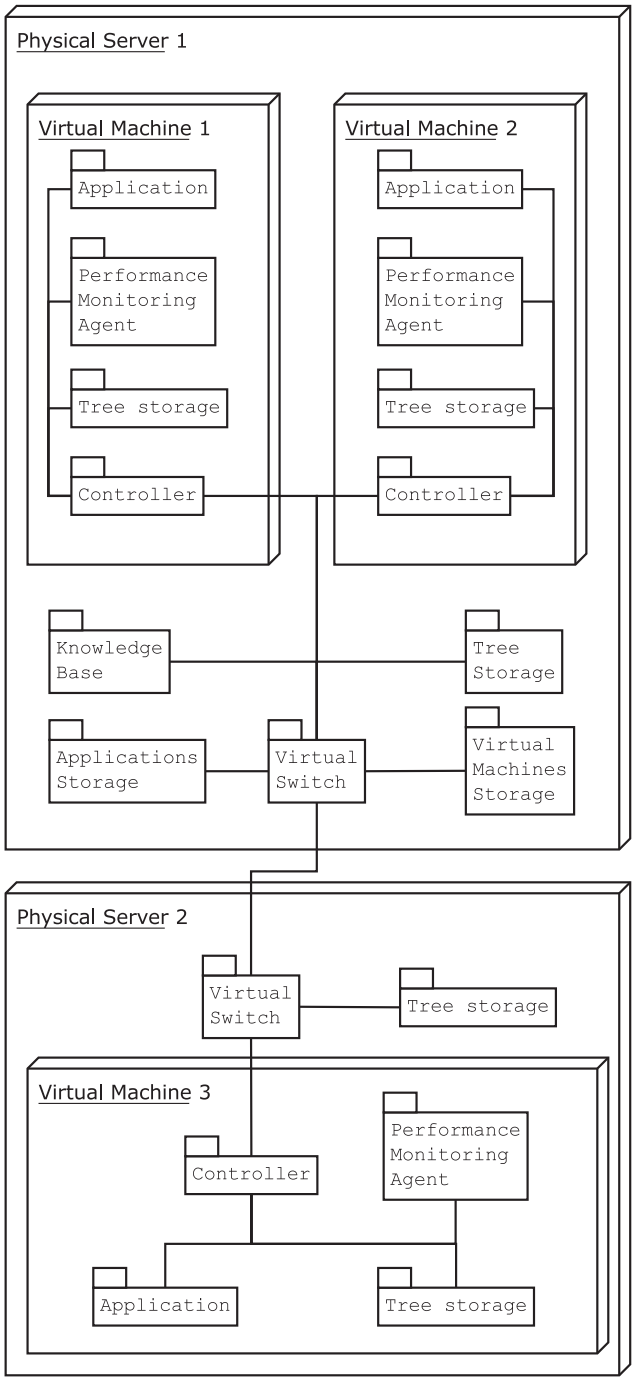

Fig. 1. Architecture of the system

III. Alternating decision trees

The algorithm we propose is based on alternating decision trees. Researched by Freund and Mason in 1999 [4] and further improved in 2002 by Holmes et al [5] this algorithm extended concept of decision trees by allowing for more than one choice coming from a single branch. More so a concept of a prediction was added, which can be seen as a weighted decision. There are several reasons behind not using any of the commonly used machine learning algorithms:

- They have longer learning time

- They have bigger overheads
- They are too complex for our tasks

In a high-performance system those points are crucial for a choice of an algorithm.

Each level has its own tree with branches defining an action, each of them having a set of prediction nodes. The parameters these prediction nodes check are different for each level and are gathered by appropriate intellectual agents. Additionally there is a decision tree for the outer level, where a decision on which level to optimize is made. There are a lot of distinctively similar sub-trees that can be generalized and instantiated when needed to provide flexibility and scalability. For example there may exist a lot of virtual machines each having a sub-tree. These sub trees are the same with only parameters for prediction nodes being different. By allowing instantiation of sub-trees we remove the necessity to recreate the whole tree to add a new virtual machine, we simply need to add a new sub-tree and fill its prediction nodes. The resulting multi-leveled tree doesn't have high overheads as was evaluated in [9] (for multi-leveled trees in general).

An example of such a tree is shown on Fig. 2.

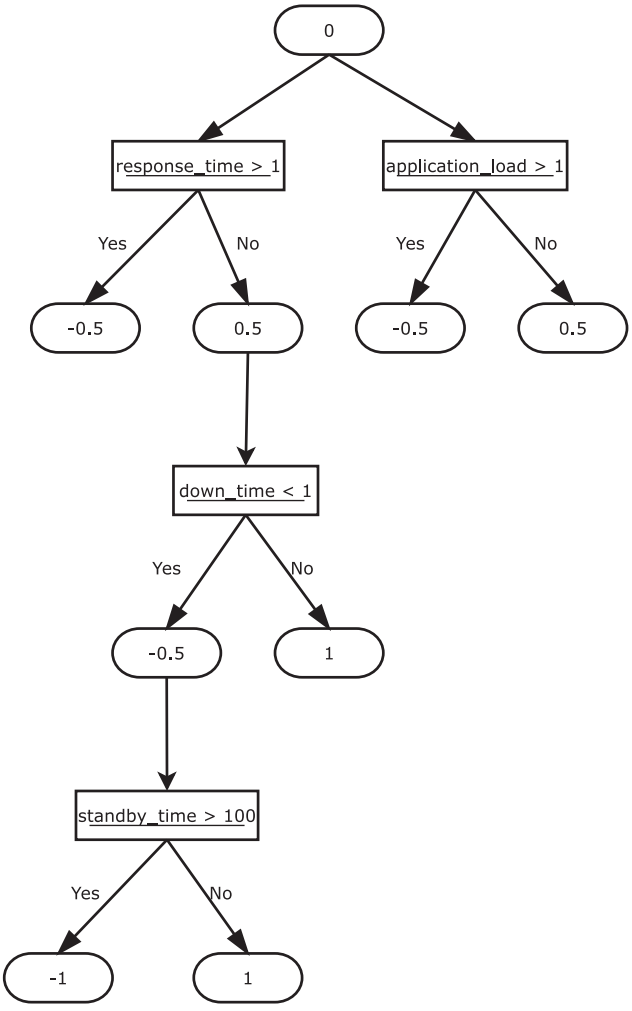

Fig. 2. Example of an alternating decision tree

IV. Optimization algorithm

The optimization algorithm itself was updated with decision trees and is more flexible now than before. Up until now the algorithm had only one or two threshold values and values could not be changed or added while the system is running; it demanded a restart. Now however its just a matter of creating a new sub-tree and instancing it, or even adjusting values "on-fly" which is the main advantage here. The system holds a set of values which describe state of a sub tree at different day times. Using these values it can behave accordingly, freeing resources when load is low and vice versa.

The algorithm is separated into several levels according to levels of the system. In our case there are three of them; the supplementary and jobs queuing levels are not being optimized as they are not real layers and thus not included in the hierarchy. The job queuing level is involved in optimization as a whole however.

The three levels are:

- Applications level
- Virtual level
- Physical level

A. Applications level

Applications level optimization happens the entire time the system is working. There are two steps to this: jobs queuing and applications migration. The jobs queuing was researched and implemented in [7] and applications migration was researched in [12]. The main idea is to try optimizing applications load by distributing jobs according to a service discipline and if there is be no increase in performance in foreseeable future try to migrate applications to another virtual machine. The decision to start migration is based on current performance values of virtual machines and system as a whole.

B. Virtual level

When applications migration is done the system takes some time to evaluate its performance. If it has not been improved then the next step of optimization is used. On virtual level there is only one step: virtual machines migration. This is a long process; however thanks to live migration functionality we dont need to stop physical server as we perform it. This means that while one virtual machine is being migrated the other ones will keep functioning properly and will keep participating in optimization algorithm. At this point during migration only the first step, jobs queuing, is active.

C. Physical level

This level is the last available step in optimization. When it comes to this that means that load is so high, the resources available to the server are not enough and thus system needs more resources. When activated this step starts a new physical server and deploys the necessary virtual machine on it. The newly created server is included into optimization algorithm and after successful deployment is able to accept jobs.

As mentioned before each of these levels is represented in the decision tree as a set of connected branches. Each time

an intellectual agent decides to update performance data on its node prediction values for these trees are being recalculated. The sub-tree branches are being removed or added as soon as a change in system infrastructure is detected. For example if a new virtual machine is deployed, its sub-tree is added to virtual level and included into algorithm.

When deciding if optimization should be performed or not the algorithm bases its decision on current day time and current performance values. Due to this the tree is updated several times during a day. This happens according to schedule when day time is changed. The borders of day times are not fixed and may be changed even while the algorithm is functioning.

Below is an example of the algorithm in pseudo code for virtual machine migration (Listings 1, 2, 3, and 4). The procedures listed in the listing are the main ones this part of the algorithm. **OptimizeVirtualMachine** shows the main procedure that runs every other, mainly **RecalculatePredictionValues** which is the main procedure for learning. When no decision can be made the algorithm decides that it was a false-positive check and updates current values in the tree according to the current ones gathered from intellectual agents. Sometime later it will find optimal values for fast determination of slowdowns.

Algorithm 1 GetDecision procedure

```
1: PROCEDURE GetDecision(node, data, decisions)
2: BEGIN
3: DOUBLE result := 0;
4: ARRAY branches := GetBranches(node);
5: INT branchesNumber := SIZEOF(branches);
6: FOR i FROM 0 TO branchesNumber
7: BEGIN
8: ARRAY branch := branches[i];
9: INT predictionType := branch[predictionType];
10: IF data[predictionType] LESS branch[prediction] THEN
11: result := result + branch[left];
12: ELSE
13: result := result + branch[right];
14: ENDIF
15: GetDecision(branch[node], data);
16: END
17: INT decisionsNumber := SIZEOF(decisions);
18: FOR i FROM 0 TO decisionsNumber
19: BEGIN
20: IF result LESS decisions[i] THEN
21: RETURN decisions[i];
22: ENDIF
23: END
24: RETURN NULL;
25: END
```

In general the algorithm for a single virtual machine can be described as next: in the beginning the system just gathers data from virtual machine's intellectual agent. At this point only job queuing is enabled. Then the decision algorithm has to decide if changes in performance are big enough to go to the next step of optimization. It does so by assuming a threshold value based on current day time. If this value is exceeded then the virtual machine is considered to be overloaded and the next phase is started. In a tree we have several decisions that may be chosen based on current performance parameters and previous decisions the algorithm made. Since this response time peak may be temporary the algorithm checks previous decision and if last time it decided to migrate virtual machine to a different server then chances are it should do so again. This is supported by current day time during which, as we assume, the average load is not changed. If a decision that virtual machine doesn't need to be migrated or stopped is made then prediction values for the whole tree are recalculated and a new tree snapshot is saved. Similar decisions are made for physical node deployment.

Algorithm 2 MigrateVirtualMachine procedure

```
1: PROCEDURE MigrateVirtualMachine
2: BEGIN
3: INT currentDecision := GetCurrentDecision();
4: INT lastDecision := GetLastDecision();
5: IF lastDecision EQUALS currentDecision THEN
6: Migrate();
7: ENDIF
8: END
```

Algorithm 3 RecalculatePredictionValues procedure

```
1: PROCEDURE RecalculatePredictionValues(tree, data)
2: BEGIN
3: ARRAY branches := GetBranches(tree);
4: INT branchesNumber := SIZEOF(branches);
5: FOR i FROM 0 TO branchesNumber
6: BEGIN
7: ARRAY branch := branches[i];
8: INT predictionType := branch[predictionType];
9: branch[predictionType] := data[predictionType];
10: END
11: SaveTreeSnapshot(tree);
12: END
```

Algorithm 4 OptimizeVirtualMachine procedure

```
1: PROCEDURE OptimizeVirtualMachine
2: BEGIN
3: ARRAY data := GetDataFromAgent();
4: DOUBLE currentPerformance := data[performance];
5: DOUBLE thresholdValue := GetCurrentThresholdValue();
6: INT dayTime := GetCurrentDayTime();
7: TREE tree := GetTree(daytime);
8: NODE rootNode := GetRootNode(tree);
9: ARRAY decisions := GetDecisions();
10: IF currentPerformance GREATER thresholdValue THEN
11: INT decision := GetDecision(rootNode, data, decisions);
12: SWITCH decision
13: CASE 0: MigrateVirtualMachine(); BREAK;
14: CASE 1: StopVirtualMachine(); BREAK;
15: CASE DEFAULT: RecalculateThresholdValues(tree);
    BREAK;
16: ENDSWITCH
17: ENDIF
18: END
```

Fig. 3 represents part of the algorithm responsible for virtual machines migration.

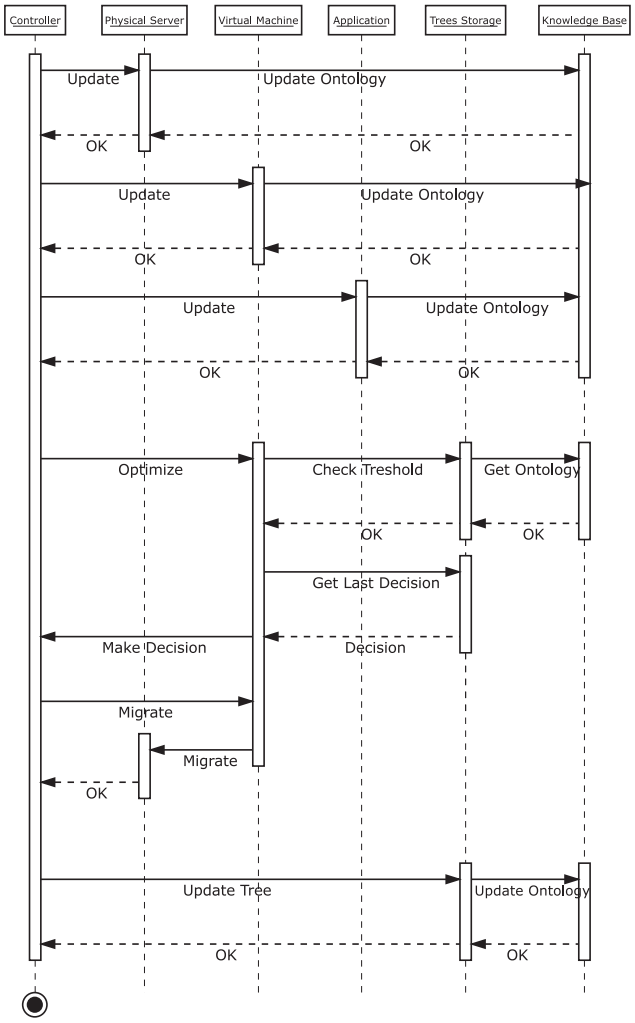

Fig. 3. Sequence diagram of virtual machines migration algorithm

This algorithm is common for all virtual machines and is controlled at the main tree which is being extended with new branches and sub-trees when configuration changes.

V. Experiments

By performing the experiments we had two goals:

- Check the credibility of updated architecture in terms of functioning
- Confirm the boost in performance

The first one is simply performing a number of experiments with decision trees enabled and see if everything works fine. The performance had to be at least the same as in [10].

The second one is to evaluate time it took the algorithm to make a decision and average performance of the system during day times. Since usage of trees can't directly improve migration time (or processing time) we instead aimed at decreasing slowdown determination time. While doing so at each day time we increase the overall performance of the system.

And finally we check the performance of the system while the tree is enabled and optimization is working. For this stage we only checked it for virtual machines migration and disabled application migration.

At first we evaluated the performance with trees and just jobs distribution enabled. This experiment consisted of simply running a number of jobs and checking the performance. In this case we used 1000 jobs and checked time it took to process each of them. The results are on Fig. 4

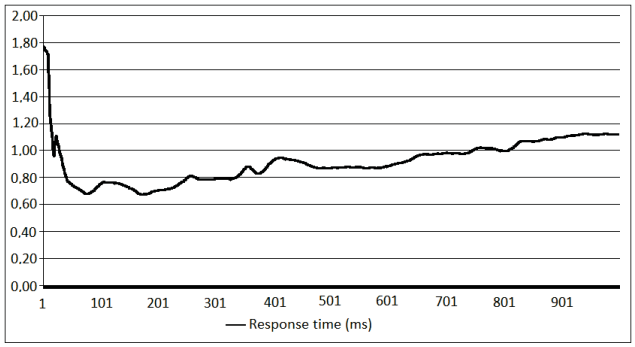

Fig. 4. Response time of the system with decision tree disabled

The response time is a little bit higher than without decision tree but the average is essentially the same.

Then we evaluated slowdown determination time without a decision tree. This gave us a value to compare. The value itself is dependent on two parameters:

- Agents' determination time
- Tree's determination time

Both values are pre-set, weren't changed during the experiments and represent amount of time the performance needs to be below threshold value to be marked as low. Both of them were set to 15 seconds. The threshold value for both agents and tree was 1 milliseconds. That means that if virtual machine's response time was higher than 1 millisecond for 15 seconds we decide that performance of this machine is low.

The third experiment was performed with a decision tree enabled. We were expecting at least some decrease in determination time due to the fact that the tree knows at which day time the performance is expected to be lower. According to that the system will determine current determination time and run optimization algorithm. Without the tree and with only agents determining the determination time it took 15.02 ms and with the tree it took only 14.18 ms. With the tree the determination time is lower which shows that there is improvement, however this needs to be researched further.

The last experiment was the same as the first one: 1000 jobs sent to the system, but this time decision tree was enabled. The results can be seen on Fig. 5.

The results are a lot less stable. First hundred iterations are the result of a "cold start": the system just started working and needs some time to stabilize. The response time at average is higher than without decision trees. This was expected as inevitably there would be some overheads. Then there are some peaks which is when recalculation of tree parameters

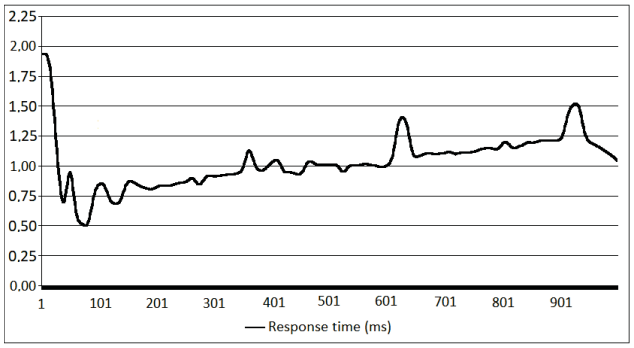

Fig. 5. Performance of the system with decision tree enabled

took place. However there is some drastic improvement near the end.

VI. Conclusion and future work

As the experiments show there are overheads when using decision trees. And although the response time is higher than without using them there are some improvements in performance. The application of decision algorithm definitely improves average performance on a long-run; however a longer experiment is needed to find out exactly how much.

In future the algorithm will be updated by integrating ontologies and not just using results of intellectual agents activity. In this work knowledge base was not taken into consideration and its effect on the algorithm will be researched later: the trees are planned to be generated based on current system's ontology which will provide a way to decrease time needed to generate decision trees. Some optimizations to the algorithm itself will be made to reach lower overheads.

Acknowledgment

This work was 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 led by ITMO University.

References

[1] *J*.H. Seo, Y.H. Kim, H.B. Ryou and S.J. Kang, "A genetic algorithm for sensor deployment based on two-dimensional operators", *in SAC '08 Proceedings of the 2008 ACM symposium on Applied computing*, March 2008, pp. 1812-1813.

[2] *A*. Cripps, "Using artificial neural nets to predict academic performance", *in SAC '96 Proceedings of the 1996 ACM symposium on Applied Computing*, Feb. 1996, pp. 33-37.

[3] *B*. Yong, Z. Xu, J. Shen, H. Chen, Y. Tian and Q. Zhou, "Neural network model with Monte Carlo algorithm for electricity demand forecasting in Queensland", *in ACSW '17 Proceedings of the Australasian Computer Science Week Multiconference*, Jan. 2017, Article No. 47.

[4] *Y*. Freund and L. Mason, "The Alternating Decision Tree Learning Algorithm", *in ICML '99 Proceedings of the Sixteenth International Conference on Machine Learning*, June 1999, pp. 124-133.

[5] *G*. Holmes, B. Pfahringer, R. Kirkby, E. Frank and M. Hall, "Multiclass Alternating Decision Trees", *in ECML 2002: Machine Learning: ECML 2002*, Sept. 2002, pp. 161-172.

[6] *B*. D. Ecole, H. P. Le, R. Makhmara, T. D. Nguyen, "Fast genetic algorithms", *in Proceedings of the Genetic and Evolutionary Computation Conference*, 2017, pp. 777-784.

[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 Proceedings of the 17th Conference of Open Innovations Association FRUCT*, 2015, pp. 259-266.

[8] *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", *in Knowledge Engineering and the Semantic Web*, 2015, pp. 273-282.

[9] *E*. Rogarda, A. Gelman, H. Luc, "Evaluation of multilevel decision trees", *in Journal of Statistical Planning and Inference*, April 2007, pp. 1151-1160

[10] *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 Proceedings of the 19th Conference of Open Innovations Association FRUCT*, 2016, pp. 279 - 284.

[11] *D*. A. Zubok, T. V. Kharchenko, A.V. Maiatin and M. V. Khegai, "Multi-Agent Approach to Monitoring of Cloud Computing System With Dynamically Changing Configuration", *in Proceedings of the 18th Conference of Open Innovations Association FRUCT*, 2016, pp. 410-416.

[12] *D*. A. Zubok, T. V. Kharchenko, A.V. Maiatin and M. V. Khegai, "Evaluation Of Optimization Control Parameters in Multi-Leveled Cloud Platform", *in Proceedings of the 21th Conference of Open Innovations Association FRUCT*, 2017, pp. 382.