presented at the Energy Efficient Systems Workshop at ICT4S, Stockholm, Aug. 2014

Load Balancing to Save Energy in Cloud Computing

Theodore Pertsas
University of Manchester
United Kingdom
tpertsas@gmail.com

Usman Wajid
University of Manchester
United Kingdom
usman.wajid@manchester.ac.uk

***Abstract*— While load balancing techniques have been designed and evaluated for efficient resource utilization in cloud computing, achieving energy efficiency as a consequence of load balancing often does not get direct attention. In this paper we describe two load balancing algorithms that focus on balancing workload distribution among physical hosts in the cloud infrastructure. The test results presented in this paper reveal the strength and weaknesses of the algorithms. In future work we aim to analyze the impact of our load balancing on energy consumption in the cloud infrastructure.**

***Index Terms*—energy efficiency, load balancing, cloud**

I. Introduction

Cloud computing offers a scalable and economical solution for addressing rapidly increasing computational needs of ICT applications. Increasing popularity and widespread adaptation of cloud computing is resulting in continual growth in terms of numbers and size of cloud facilities, raising questions about effective management of workloads as well as long term environmental implication of cloud computing model. In fact, management of cloud applications with the view to achieve energy efficiency is now being considered a critical challenges to be dealt with in cloud computing.

Existing techniques for achieving energy efficiency in cloud computing focus on strategic power management operations e.g. by suggesting the use of low powered machines [1] and deploying applications on fewer machines or spreading them evenly across available resources/machines [2]. Typically, such techniques are applied at the initial deployment stage i.e. while deploying new applications on cloud. In the work presented in this paper we address the load balancing and energy efficiency issues at post-deployment stage. Particularly, we focus on load balancing as a way to ensure effective and efficient utilization of cloud resources and consequently to find alternative deployment configurations that can contribute towards saving energy in the cloud infrastructure.

The two load balancing algorithms we describe in this paper aim to minimize wastage of cloud resources as a result of under-utilization of some resources; and minimize lengthy response times as a result of over-utilization, where both cases contribute towards excess energy consumption. We adopt an agent-based system development approach for the implementation and testing of the load balancing algorithms. In the proposed system, cloud applications are decomposed into independent but interrelated tasks, each of which can be deployed and executed on *Host Agents* that are tightly coupled with a physical hosts or servers in the cloud infrastructure. Host agents implement the load balancing algorithms and interact with each other to perform load balancing operations.

The definition and testing of load balancing algorithms, reported in this paper, will lead to further investigation about their potential impact on energy consumption within the cloud infrastructure. The comparative evaluation of the algorithms reveals their strengths and weaknesses and more importantly provide motivation for further work in the area of energy efficient and environmentally aware cloud computing.

The structure of the paper is as follows, Section II presents the preliminary details about the load balancing algorithms. Section III presents the design details of the system, where the load balancing algorithms were implemented and tested. Section IV describes the two load balancing algorithms. Section V presents the results of testing the algorithms in a cloud environment. The paper ends with a summary in Section IV.

II. PRELIMINARIES

This section describes the preliminaries concerning the cloud infrastructure and load balancing algorithms.

Task is an application component which is either executed or stored in a virtual machine (VM) with specific CPU, memory and disk space. A cloud application can have several tasks each running on a VM.

Host is a physical server capable of hosting several VMs.

Load can be considered to be a number of tasks which currently exist at a host

As tasks arrive at a host, the load of the host is increasing. Due to the heterogeneity of the system tasks can be completed at different time intervals and therefore some hosts will be more loaded than the others at any given time. Thus *load balancing*, as the name implies, is a method to distribute the load to the hosts in the system.

A load balancing algorithm needs to consider various factors in order to decide if the system needs balancing and how to perform it. Depending on how these decisions are taken, where they are taken, how they are performed and what additional benefit is being sought (e.g. energy efficiency in our

© Copying permitted only for private and academic purposes.
This volume is published and copyrighted by its editors

perspective), we can identify different types of heuristics and policies for load balancing.

Before we delve into more technical details about our load balancing algorithms it is worth noting our reasons for its importance:

- Reduces overloading of certain resources e.g. hosts
- Energy efficiency by virtue of minimizing the overall execution time of tasks in cloud infrastructure
- Maximises the amount of work done by the cloud infrastructure (throughput).

In a cloud infrastructure, we assume that tasks can be migrated between hosts and new tasks can arrive at any host at any time. In a cloud infrastructure, there can be N number of hosts, which form the set *H*. Each host $i \in H$, has a load l_i at any time t. The load l_i depends on the used CPU, memory, disk capacity, etc. of the host. Denoting $X_1, X_2, \dots X_k$ the set of values for CPU, memory, capacity, any other quantity that affects the load of a host, we define the load function

$l_i: X_1 \times X_2 \times \dots \times X_k \rightarrow \mathbb{R}$, for each host *i*.

Following the utility-based optimization approach [1] the utility of each task is defined as the inverse of its load. Hence, we define the utility function for a host *i* as

$$u_i = \frac{1}{l_i}$$

The utility function is a metric for the load of a host. Depending on the load balancing heuristic, we need to minimise or maximise u_i. Below we denote the *system* utility function, which represents the overall optimisation problem at the system level. We are trying to find solutions that will end up minimizing U over time.

$$U = \sum_{i=1}^{N} u_i$$

III. SYSTEM DESIGN

Here we describe the design features of the different modules which contribute towards the design of a system that allow us to implement and test our load balancing algorithms and give a solution that is applicable in cloud computing.

A. Tasks

A task is executed at the host and has certain requirements. In our system each new task has certain characteristics or structure, as shown in Figure 1.

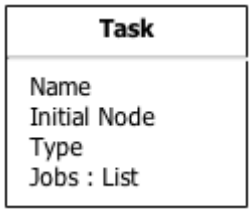


Figure 1: Task agent structure

Each task has a certain type and can only be executed if the host supports this type. Otherwise, it remains in the waiting list of the host.

B. Host

A host is a central entity in the cloud infrastructure and is the place where the load balancing happens in our system.

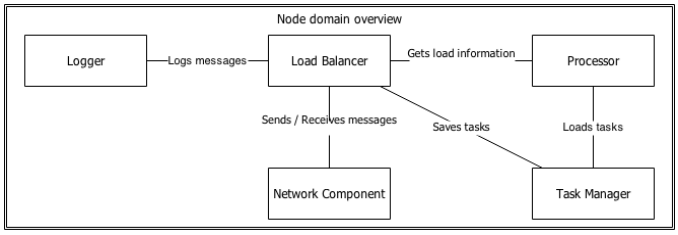


Figure 2: Internal structure of host Agent

Each host is represented by an agent that runs individually from other host agents in the cloud infrastructure. The internal structure of a host agent is composed of five components, the *network component*, the *processor*, the *task manager*, the *logger* and the *load balancer* as shown Figure 2.

Network Component: each host agent is able to interact with other host agents by sending and receiving messages in a reliable manner. In this respect, a host agent acts as both a server and a client. As a server, it runs continuously in the background, in order to accept messages and tasks.

Processor: The component that represents the running tasks in a host is the processor component. In this respect, the processor is responsible for creating or setting up the VMs where tasks can be deployed and executed. The processor starts the creation of VMs and subsequently execution of tasks by picking them from a waiting list, which is basically a temporary storage space in the host agent where new task (request) arrives. New tasks wait in the waiting list until their required resources are allocated (e.g. a VM is setup).

Task Manager: The task manager is the component of host agent that is responsible for receiving and analysing new tasks, and loading tasks from waiting list to the processor.

Logger: The logger component (as shown in Figure 3) receives messages from the load balancer about the state of the host and any requests that are received from other hosts. The logger can log messages, as list of events, either locally or receive messages from other hosts in the cloud infrastructure.

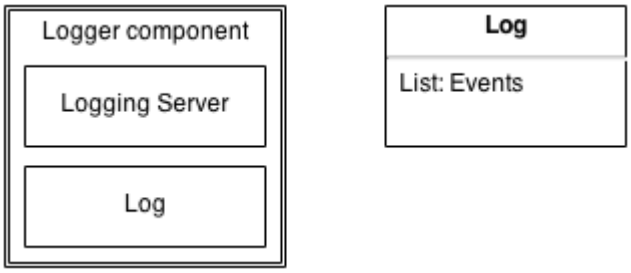
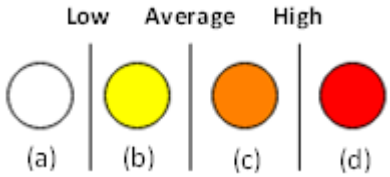


Figure 3: Logger component and format of a Log

Load Balancer: The most important and decision making entity in the host agent is the load balancer. Depending on the

algorithm that will be used, the decisions for the load balancing operation will be taken here. The load balancer is able to communicate with the processor, in order to get latest load information. It will use the network component to send messages to other hosts. The network component will pass any received messages to the load balancer for evaluation. If a task is received, then the load balancer uses the task manager to check the task details/structure. The load balancer has an address book that holds the addresses of other host agents in the system. The decision making about load balancing operation is based on the notion of state transitions. In this respect, each host agent, can be in one of the following states at any time:



$$\{Low \mid LowAverage \mid HighAverage \mid High\}$$

A host agent can determine the state by checking its current load (in the *Processor* component) against the thresholds M_{lower}, $M_{average}$, M_{upper} in a local statistical table. The following table shows the relation between the thresholds and the states.

High	$l_i > M_{upper}$
HighAverage	$l_i < M_{upper}$ && $l_i > M_{average}$
LowAverage	$l_i < M_{average}$ && $l_i > M_{lower}$
Low	$l_i < M_{lower}$

Each host agent determines its state independently from the others and takes the appropriate action depending on the load balancing algorithm.

IV. LOAD BALANCING

Here we describe two algorithms for load balancing and energy efficiency. The algorithms are based on heuristics and their main focus is to achieve efficient utilization of available resources and consequently lower the energy consumption, while having a minimal overhead in the system.

A. Secretaries

This heuristic is inspired by the swarm intelligence approaches and the reality of an office environment. For example, in a company, when a manager wants a task done, he delegates it to a secretary. The secretary in her part, makes a couple of phone calls, or goes around the office, and finds a suitable person to execute the task. If we imagine a situation, in which there is more than one secretary, then we can have the necessary background for a swarm intelligence approach. Each one of the secretaries will perform their quest, independently of the others and once they find a candidate they either pass-over their task, or move to the next candidate. The manager in our system is the *host* in which the task arrives, and the candidates are the other hosts in the network.

This algorithm is triggered when a host reaches the high threshold, which sets the host in high load state. Before we start the discussion on the load balancing algorithm, we need to set the following policies for host agents:

a) Transfer policy: We use a preset 'High' threshold value, above which the host agent triggers the load balancing operation. The host agent starts looking for candidate host agents to send tasks. If a host is below this value, then it may be able to receive more tasks.

b) Selection policy: Host agents follow a two-step process to choose tasks to send. In the first step, tasks are selected based on type e.g. small, medium or large VM instances. On the second step, hosts randomly choose from the set of tasks of the same type. We receive tasks of the type that we can execute in the respective hosts.

c) Location policy: Host agents can send requests to all the hosts in their address book and then wait for an answer. If more than two answers are received at the same time, then the hosts choose randomly from the two. Host agents wait for a period of time before they resend any requests, to avoid flooding the network with unnecessary messages.

d) Information policy: Host agents follow a demand-driven approach, since the requests are sent only when there is change in state of the host agent.

Algorithm 1: Secretaries (Stage 1)

```
Initialise a statistics table
while cancellation has not been requested do
  get host waiting list load
    store the values in the statistics table
    if the statistics table has adequate size then
        calculate average waiting list size
        determine the state of the host
        if the state is High then
          Decide number and type of tasks to send out
          Send load balancing request to hosts in the
      address book
```

If the algorithm decides that the host has high load, then it proceeds to ask for help. The "ask for help" operation will be referred as Load Balancing Request (LBR). As mentioned before, a host is in high state when it is over the high threshold. In this algorithm, host agents follow the strategy that they need to send as many tasks as they can to other hosts, in order to fall under the high threshold. In this case, this will happen when the hosts send the excessive items of the waiting list.

Once a host agent sends the LBR, the next stage happens in the load balancing algorithm of the host agent that receives it.

Algorithm 2: Secretaries (Stage 2)

```
Check host state
if host state is not High or High Average then
    Check if the types of tasks (within LBR) can be executed on this host
    if they can be executed then
        Decide number of tasks to ask for
        Accept the LBR and send the number of tasks back to the sender
else
    Forward the LBR to host agents in the address book
```

The receiver host agent, checks its status looking at the local statistics table. If the state is Low or Low Average, then it can accept the LBR. The host agent asks for as many tasks as it is needed to reach the High threshold, or just over it, thus setting its state at High Average or High. The main criterion for this to happen is the local queue size, because this determines the size of the batch. After receiving the tasks, the host agent can still accept more tasks until an acceptable state is reached.

Once the host accepts the LBR message, it sends back to the initial host agent the number of tasks that it can receive, in order to help the initial sender/requester host agent. This leads us to the third and last stage of the algorithm.

Algorithm 3: Secretaries (Stage 3)

```
if suitable candidate was found after sending LBR then
    Stop load balancing
    Pick up the desired number of tasks from the waiting list
    Send tasks
```

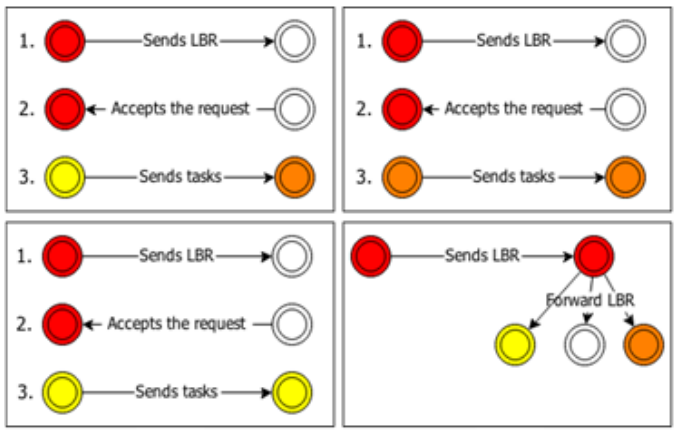


Figure 4: Some of the possible states of host agents. Load balancing starts when the host agent is in High load state. The numbers represent the stages of the algorithm.

Once a suitable candidate is found that is willing to accept excess load of the initial requester (host agent), the first step at this stage is to stop the load balancer. The final number of tasks to send out is the maximum number that the other host agent can accept. The next step is to pick up the tasks that will be sent out. To perform this step we choose the tasks randomly without adding further complexity to the algorithm.

B. Eager Worker

This algorithm is inspired by the epidemic protocols; a relatively new approach to load balancing by spreading information around the system in a similar way that a virus (or gossip) spreads. In this scenario, the central role is played by a worker who does not have any work to do. However, since he is eager to work, he goes to all the people that he knows and announces his availability ("infects"). Since he cannot do anything else, he waits for someone to call him. The people who know this fact now have the option to either use him, or spread the rumor around that there is an available worker ("spread the infection"). In our case, worker is a host agent who goes under the average threshold and is in either Low Average or Low state. The algorithm has following policies:

a) Transfer policy: Host agents use the Average threshold value in order to trigger the load balancing operation. Below this value the hosts will be able to receive tasks.

b) Selection policy: This policy is the same as in the "Secretaries" algorithm.

c) Location policy: Host agents are selected for sending a load balancing request. After sending the requests, the sender host agent waits for a reply. Requests are sent in an interval, to avoid flooding the system. If a request for help arrives from two or more host agents, then host agents operate in a "first-come, first-served" manner.

d) Information policy: Host agents follow a demand-driven policy. The requests are sent, when the host is in Low or Low Average state.

This algorithm follows a more opportunistic model: if there is help, the host agent will make use of it without considering the overall state of the system.

The algorithm has a two-stage execution, contrary to the three stages in "Secretaries" algorithm. At the first stage a host agent determines the state of the host and then sends the LBRs. Algorithm 4 includes the pseudo-code for this stage.

Algorithm 4: Eager Worker (Stage 1)

```
Initialise a statistics table
while cancellation has not been requested do
    get host waiting list load
    store the values in the statistics table
    if the statistics table has adequate size then
        calculate average waiting list size
```

determine the state of the host
if the state is Low or Low Average **then**
Decide number and type of tasks to send
Send load balancing request to host agents in the address book

Most of the steps in this algorithm are similar to the "Secretaries" algorithm. When deciding for the number of tasks and types to send, we base our calculations on the waiting list of hosts, since this is what determines the amount of tasks to delegate. The host agents in this algorithm ask for as many tasks as needed, in order to remain under the High threshold.

When the number and types of tasks to ask is decided, a LBR is sent to either a random number of host agents in the address book or to all of them. If the LBR is sent to all of them then this might lead to an overflow of messages in the system. Since this algorithm follows an epidemic approach, the LBRs will reach to all the host agents in the system..

Algorithm 5: Eager Worker (Stage 2)

if load balancing request was received **then**
Check host state
If host state is High or High Average **then**
Check if the sender host agent can execute the tasks of this host
if they can be executed **then**
Stop load balancing
Decide number of tasks to send and choose tasks
Accept the LBR and send the tasks
Start load balancing
else
Forward LBR to host agents in the address book

The next stage of the algorithm is executed by the host agent that receives the request.

This time the host agent accepts the LBR if its state is in High Average or High. The High Average state is included as well, because the host agent at this stage needs tasks to finish execution. Since the algorithm follows an opportunistic model, if there is an idle worker, then it sends tasks across. The number of tasks depends on the total size of the waiting list. It sends either the maximum tasks that can be executed in the other hosts, or in the case of High Average state, enough tasks so as to get close to the Average threshold.

The load balancing operation needs to stop for the same reasons as before and it is restarted after the host agent finishes the transfer of tasks.

The choice of tasks to send follows a similar pattern to the "Secretaries" algorithm. In this initial design, we decided to choose the tasks in a random way.

If there is no need for load balancing, then the LBR is forwarded to other hosts of the network. If the LBR arrives back to the original host agent, then it is rejected automatically, thus, avoiding duplicate requests going through the network.

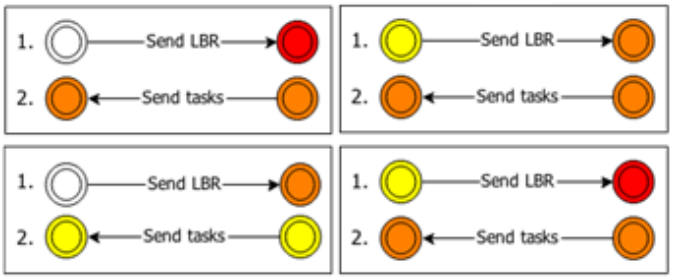


Figure 5: Possible states of host agents. Load balancing starts when the host agent is in Low or Low Average state. The numbers represent the stages of the Eager Worker algorithm

V. Results

The algorithms were tested in a single site cloud environment. The message exchanges between host agents happen in a round-robin format. A host agent "knows" only one other host agent. The reasoning behind this scenario, compared to broadcasting or *n*-to-*n* interactions, was to minimize the complexity of managing multi-agent interactions (which was not the focus of the work presented here) and more importantly to see if the algorithms develop any behavior. We used ZeroMQ (http://zeromq.org/) technology to realize message exchange between host agents. Command messages travel in one direction starting from one host agent to the last but task transfer can happen between any two host agents.

We ran our experiments for each algorithm five times and we get averages of the number of tasks at each host after an interval. We create a set of 75 tasks in individual VMs and deployed them in the first host. Our aim was to observe how the algorithms handle spikes of load and energy consumption at a host. Furthermore, we monitored the number of messages exchanged between the host agents every minute.

The results of testing both algorithms are shown in Figure 76 and Figure 7.

For Eager Worker load balancer, Table 1 shows the average messages per minute circulating in the network.

Table 1: Eager Worker: Number of messages (avg. five runs)

Total transferred tasks	**60**
LBRs per minute	**32,8**
Messages per minute	**234**

On average there were around 33 LBR messages every minute since a number of host agents with low or low-average states were asking for tasks.

Table 2: Secretaries: Number of messages (avg. five runs)

Total transferred tasks	49
LBRs per minute	0,5
Messages per minute	191

For Secretaries load balancer, the first host agent sends a LBR every time it needed load balancing – as shown in Table *12*, hence the very low number of LBRs in the network.

As shown in Figure 6, it is obvious that Eager Worker achieves a better overall performance at the expense of increased network traffic. The load is distributed uniformly among the nodes. In particular, Host B did not get as many tasks due to the preset MaxRetransmits value that restricts host agents to transmit only 5 messages during the testing period.

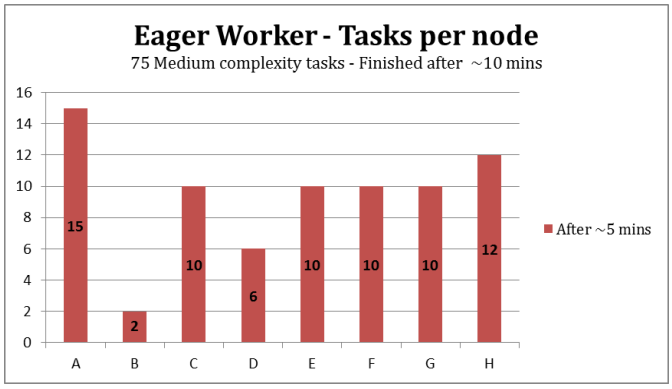


Figure 6: Distribution of tasks on each node after approximately 5 minutes of operation (averaged and rounded)

Whereas, in the case of secretaries algorithm, as shown in Figure 7, Host A was 7 hops away from Host H, hence its LBR messages never arrived that far. Due to the design of the algorithm, LBRs are always trying to find overloaded hosts.

Overall, Eager worker generates approximately 28% more traffic than Secretaries, but converges faster to an overall load balanced state. On the other hand, Secretaries tend to form "neighbourhoods" e.g. host agent A visits first its closest neighbor and gives them work to do and then slowly spreads its load to the rest. For this reason the further hosts seldom receive any tasks. Even if there were more tasks, Host H would have never been reached due to the preset MaxRetransmits value. Removing the MaxRestransmits constraint or increasing its preset value may allow spreading the tasks to further hosts.

However, the advantage of secretaries algorithm is lower network traffic: LBRs are sent only when it is needed. On the other hand, due to the formation of neighborhoods and slower distribution of load, the execution finished almost 2 minutes later than Eager Worker.

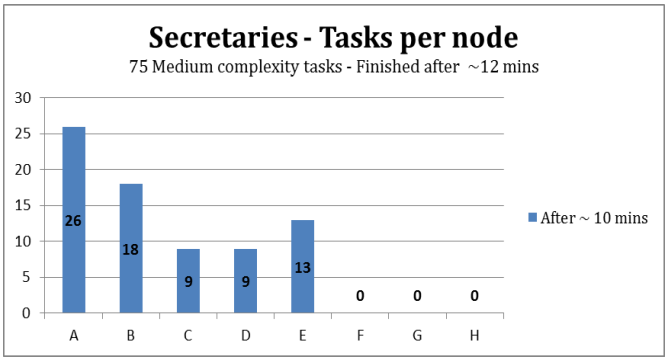


Figure 7: Distribution of tasks on each node after approximately 10 minutes of operation (averaged and rounded)

VI. SUMMARY AND FUTURE WORK

This paper presents two heuristics-based algorithms for load balancing. The results of testing both algorithms reveal their advantages and disadvantages as one might be performing better than the other in any given context. By tweaking the various parameters, we can achieve better performance, but there is always a trade-off. The work presented in this paper focuses on testing the load balancing aspect of the algorithm. In future work, we aim to analyze the impact of these algorithms on the energy consumption of cloud infrastructure within the context of ECO_2Clouds project (www.eco2clouds.eu).

ECO_2Clouds allows quantification of energy consumption and environmental impact (CO_2 emissions) at three different levels of cloud infrastructure. These include testbed, physical host and VM levels. The ability to quantify the energy consumption at testbed and physical host level will allow us to investigate the use of our load balancing heuristics as runtime adaptation mechanisms that can balance resource utilization with reduction in energy consumption.

REFERENCES

[1] G. Luigi Valentini, W. Lassonde, S. U. Khan, N Min-Allah, S. A. Madani, J. Li, L. Zhang, L. Wang, N. Ghani, J. Kolodziej. H. Li, A. Y Zomaya, C. Z. Xu, P. Balaji, A. Bishnu, F. Pinel, J. E. Pecero, D. Kliazovich. P. Bouvry. An overview of energy efficiency techniques in cluster computing systems. In *Cluster Computing*. March 2013, Volume 16, Issue 1, pp 3-15

[2] P. Lindberg, J. Leingang, D. Lysaker, S. U. Khan, J. Li. Comparison and analysis of eight scheduling heuristics for the optimization of energy consumption and makespan in large-scale distributed system. In *Journal of Supercomputing*. January 2012, Volu. 59. Issue 1, pp. 323-360.

[3] L. Skorin-Kapov, et al., "Approaches for Utility-Based QoE-Driven Optimization of Network Resource Allocation for Multimedia Services," in *Data Traffic Monitoring and Analysis. vol. 7754, E. Biersack, et al., Eds., ed: Springer Berlin Heidelberg*, 2013, pp. 337-358.