A Meta-Heuristic Load Balancer for Cloud Computing Systems

Sliwko L. and Getov V.

A Meta-Heuristic Load Balancer for Cloud Computing Systems

Leszek Sliwko
School of Electronics and Computer Science
University of Westminster
London, United Kingdom
e-mail: lsliwko@gmail.com

Vladimir Getov
School of Electronics and Computer Science
University of Westminster
London, United Kingdom
e-mail: V.S.Getov@westminster.ac.uk

***Abstract*—This paper introduces a strategy to allocate services on a cloud system without overloading the nodes and maintaining the system stability with minimum cost. We specify an abstract model of cloud resources utilization, including multiple types of resources as well as considerations for the service migration costs. A prototype meta-heuristic load balancer is demonstrated and experimental results are presented and discussed. We also propose a novel genetic algorithm, where population is seeded with the outputs of other meta-heuristic algorithms.**

Keywords-cloud computing; load balancing; meta-heuristic

I. INTRODUCTION

Modern day applications are often designed in such a way that they can simultaneously use resources from different computer environments. System components are not just properties of individual machines and in many respects they can be viewed as though they are deployed in a single application environment. In recent years the most advanced technologies offer cloud solutions [9]. A cloud system connects multiple individual servers and maintains the communication between them in order to process related tasks in several environments at the same time. Clouds are typically more cost-effective than single computers of comparable speed and usually enable applications to have higher availability than on a single machine.

Software as a service, where functionality is delivered to end users directly from distributed data centers, is a typical paradigm of the use of cloud systems [20]. Companies no longer need to be concerned about maintaining a huge IT infrastructure. Instead they can simply rent thousands of servers for a required time [9]. A few well-known examples of services backed up by cloud computing are Dropbox, Gmail, Facebook, Youtube and Rapidshare.

This elasticity of resources, without paying a premium for a large-scale usage, is unprecedented in the history of IT [9]. However, it introduces a new set of challenges and problems, which need to be solved. The cloud systems are usually made up of machines with very different hardware configurations and different capabilities. These systems can be rapidly provisioned as per the user's requirements [4] thus resource sharing is a necessity.

This piece of research outlines the significance of resource management strategies in cloud systems – a class of systems that are characterized by dynamic changes in their environments. The conventional containers for applications in cloud systems are virtual machines (VM), which can be quickly booted up or shut down on demand [8] and therefore the strategy needs to be robust enough to accommodate rapid changes in available resource configurations. In this paper, we specify an abstract model of cloud resources utilization (Section II), including multiple types of resources and consideration of service migration costs. Using an abstract model, Section III describes the research problem formulation. We then present the design of a prototype meta-heuristic load balancer (Section IV), which can be used to manage medium-size cloud systems. Sections V and VI provide the details of the experiments setup and results. In Section VII we conclude by discussing various employed strategies and highlight their advantages and weaknesses.

II. MODEL OF CLOUD RESOURCES UTILIZATION

Our model consists of nodes and services where the load balancer task is to keep a good load balance through resource vector comparisons. In considering what is actually constituted as a 'service' in a cloud environment an example may be seen in a popular cloud environment such as Amazon's EC2, where applications are deployed within the full operating system VM. One might question the effectiveness of this approach; however this schema has many benefits such as the almost complete separation of execution contexts and a complete control over the local system environment parameters.

Services run constantly, which means they are not *tasks* which can be defined as a finite piece of work to be done [12] and do require resources, which are provided by the nodes. Every node has a certain amount of variable resources available, referred to in this paper as the *available resources* set. All resources on nodes are considered renewable and continuous. Assigning a service to a node only temporarily lowers available resource levels. Both the *resources needed* by the service and the *resources available* on the node are described by the vector of integer values. In this experiment we use four types of resources: CPU, allocated memory, network bandwidth and I/O operations speed.

A cloud system environment is characterized by very dynamic changes in resource availability. During its operations, some nodes might become idle or overloaded,

additional nodes might become available, demand for particular service changes or part of a cloud network could go offline. Therefore it is critical to provide a mechanism to automatically migrate services to alternative nodes.

Distributed systems often store or process large amounts of '*states*' – a *state* consists of data such as a database, files, relations, session data and identifiers, which are frequently updated [20]. Service migration is similar to jobs check-pointing [5]. During service migration the service VM is stopped and its *state* saved to a state snapshot file. This file then gets copied over the network to an alternative node, where the same VM is then restored. Therefore, a service will always carry some of the system *state*. Our tests (VMware) show that saved state size (collapsed snapshot file) is proportional to application disk usage combined with memory usage (see 'general formula' in [25]).

When we move a service to an alternative node, the *state* also has to be transferred. In this model, every service has its integer cost value assigned which is an abstract representation of the impact the *service migration* will.

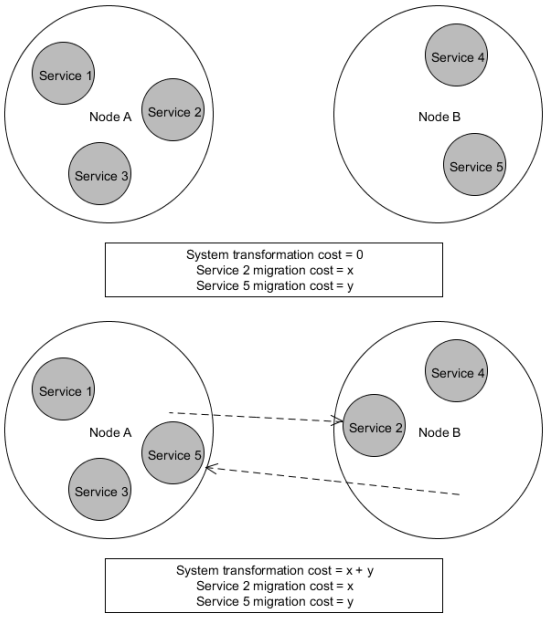

Figure 1. System transformation and migration cost

III. PROBLEM FORMULATION

Let us define $\Lambda=(\tau,\eta,\psi,a,r,c)$ as a problem space and system as a pair (Λ,μ). In the ***d-resource system optimization problem***, we receive a set τ of l mobile services $\tau=\{t_1,t_2,...,t_l\}$ and a set η of m fixed nodes $\eta=\{n_1,n_2,...,n_m\}$. We call $\mu:\tau\to\eta$ as a *service assignment* function, where each service has to be assigned to the node.

We also consider:

- $\psi=\{i_1,i_2,...,i_d\}$ as a set of all different kinds of resources. To illustrate, for $d=3$ we could define $\psi=\{CPU,memory,network\}$.
- $a:\psi\times\eta\to \mathrm{N}\cup\{0\}$ as a fixed *available resources* on the nodes. $a_i(n)$ is the available level (integer value) of a resource i on the node n.
- $r:\psi\times\tau\to \mathrm{N}\cup\{0\}$ as a fixed *required resources* for services. $r_i(t)$ is the required level (integer value) of a resource i of service t.
- $c:\tau\to \mathrm{N}\cup\{0\}$ as a *service migration cost* function. $c(t)$ means cost incurred migrating service executables and its *state* and preparing service environment.

For every node $n\in\eta$ we define a set $A_n=\{t\in\tau:\mu(t)=n\}$ of all services assigned to the node n. We also define $f:\psi\times\eta\to \mathrm{N}\cup\{0\}$ as *remaining resources* on the nodes:

$$f_i(n)=a_i(n)-\sum_{t\in A_n} r_i(t) \tag{1}$$

We consider system (Λ,μ) as *stable*, if $f_i(n)\geq 0$ i.e.:

$$\sum_{t\in A_n} r_i(t)\leq a_i(n) \text{, for every } n\in\eta,\ i\in\psi \tag{2}$$

Otherwise the system (Λ,μ) is *overloaded*.

Each service t is initially assigned by *service assignment* function μ_0 to some node η. During the *system transformation* $(\mu_0\to\mu_1)$ service $t\in\tau$ can be reassigned to any different node $n\in\eta$. The process of moving the service to a different node is referred to as *service migration* and this feature generates a *service reassigning cost*:

$$c_{(\mu_0\to\mu_1)}(t)=\begin{cases}0, & \mu_0(t)=\mu_1(t)\\ c(t), & \mu_0(t)\neq\mu_1(t)\end{cases}$$

Every *system transformation* process $(\mu_0\to\mu_1)$ has its *system transformation cost*:

$$c_{(\mu_0\to\mu_1)}=\sum_{t\in\tau} c_{(\mu_0\to\mu_1)}(t) \tag{3}$$

Consider initial *service assignment* μ_0; *service assignment* μ^* is optimal for μ_0, if μ^* renders system (Λ,μ^*) *stable* and:

$c_{(\mu_0\to\mu^*)}\leq c_{(\mu_0\to\mu)}$, for every *stable* system (Λ,μ).

N.b.: when (Λ,μ_0) is *stable* for initial *service assignment* μ_0, the *system transformation cost* equals zero as it is considered optimal.

We also consider two *service assignment* functions μ_0 and μ_1 to be *neighbors* if:

$$\left|\left\{t \in \tau : \mu_0(t) \neq \mu_1(t)\right\}\right| = 1 \quad (4)$$

The ***d-resource system optimization problem*** (D-RSOP) is a variant of classical Resource-Constrained Project Scheduling Problem (RCPSP), thus D-RSOP also belongs to the NP-hard (Nondeterministic Polynomial-time hard) problems class. RCPSP has been examined numerous times by researchers and numerous solutions have been proposed, implemented and tested [1][2][3][6][15][18]. RCPSP is solvable by simple heuristics such as the H1m (heuristic procedure where each job is assigned a fixed continuous resource amount equal to 1/m) and HCRA (heuristic procedure for continuous resource allocation) algorithms [17], however the result quality is low. Exact methods have been explored, but either they have a limitation of problem size or focus only on deriving new lower bounds as an optimal solution can be found and verified only in small problem instances [15][21]..

IV. LOAD BALANCER DESIGN

Our load balancer prototype was implemented in a functional programming language Scala (version 2.11.4). The source code of this load balancer is available at: https://github.com/lsliwko/MASB. All computations were performed on a MacBook Pro with 2.4GHz dual-core Intel Core i5 and Java 1.6.

The core of the load balancer is a decision-making module based on meta-heuristic algorithms, which assigns services to nodes. The load balancer sequence was designed as shown in Figure 2.

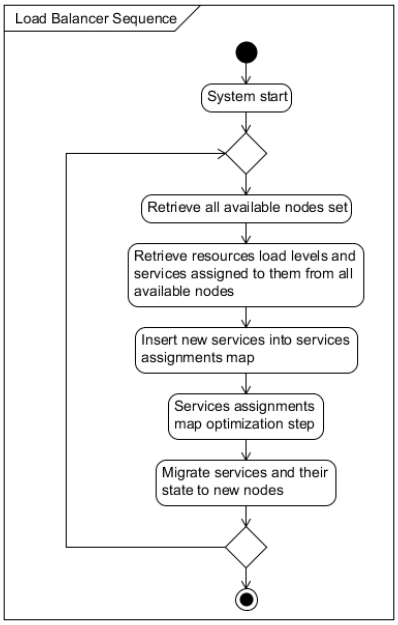

Figure 2. Load balancer sequence

Our load balancer has to maintain a difficult balance between the speed and quality of its decisions as badly assigned services can cause global system instability. The selection of the most efficient algorithm is critical. Based on previous research [16], as well as our existing work not every algorithm will perform well with this problem. For the purpose of the experiment, we have selected several of the most promising strategies as outlined below.

A. Greedy is an algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum. In many problems, a greedy strategy is effective; however, it usually does not produce an optimal solution in this research. Nevertheless, a greedy heuristic will yield locally optimal solutions in a very quick time.

B. Tabu Search (TS) was introduced by Fred W. Glover in 1986 [10] and further formalized in 1989 [11]. This algorithm has been suggested by previous research on a similar problem [14]. TS searches for an improved solution in immediate neighbors (solutions that are similar except for one or two minor details). TS enhances its performance by maintaining a list of visited solutions so that the algorithm does not consider that possibility repeatedly.

C. Simulated Annealing (SA) is a general method for finding the global optimum by a process inspired from annealing in metallurgy heating and controlled cooling of a material to increase the size of its crystals and reduce their defects [26]. This effect is implemented in the SA algorithm as a slow decrease in the probability of accepting worse solutions as it explores the solution space. Previous research over use of this strategy in load balancing can be found [16].

D. Genetic Algorithm (GA) — is a search heuristic that mimics the process of natural selection. GA belong to the larger class of evolutionary algorithms, which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. Unmodified GA has been previously examined with good results [15]. In this research we have deployed a variant with Genetic Drift step — detailed in [23].

E. Seeded Genetic Algorithm (SDA) — the generation of random solutions is the most costly step in GA strategy, sometimes taking up to 60-70% of a total computation time. Therefore, we have implemented a novel approach, where Genetic Drift step has been replaced with locally optimal solutions (i.e.: solutions seeding) found by Greedy, Tabu Search (TS) and Simulated Annealing (SA) algorithms. This approach should allow us to dramatically lower the total size of population (as individual genotypes are of better quality). To test this approach, respective strategy variations were created: SGA-Greedy, SGA-TS and SGA-SA.

F. Full Scan (FS) – this strategy performs a full search over all available configurations. FS strategy is convergent meaning it is able to find the globally optimal solution in finite time, under appropriate modeling assumptions. Multiple optimization techniques have been implemented in this algorithm, including shaving and path cut [7] and largest-migration-cost-first.

V. EXPERIMENTS SETUP

The characteristics of a cloud workload in a data center significantly differ from traditional grid computing [8]. There exists only a limited number of publicly available cloud system workload traces and those are stripped of useful details [21]. The research community is mostly relying on simulations and models to conduct their experiments. The quality of input data and its realistic nature is a very important factor as it has a direct impact on the accuracy of results. In this experiment we have generated system configuration based on the previous research [8][13][21][22] and also on our professional experience while working with Amazon EC2 cloud instances (see Table 1 and Table 2).

TABLE I. EXPERIMENTAL DATA – NODES CONFIGURATION

Node	Resource I (CPU)	Resource II (Memory)	Resource III (Network)	Resource IV (I/O speed)
A	100	50	100	70
B	70	40	70	50
C	50	80	70	50
D	60	60	50	80
E	50	90	80	40
F	60	100	50	60
G	80	50	50	40
H	80	80	80	90

TABLE II. EXPERIMENTAL DATA – SERVICES CONFIGURATION

Service	Initial node	Migration cost	Resource I (CPU)	Resource II (Memory)	Resource III (Network)	Resource IV (I/O speed)
01	A	4	1	10	4	2
02	C	5	1	6	5	2
03	G	4	5	2	5	6
04	A	17	10	17	1	2
05	D	10	14	10	1	1
06	C	3	3	12	3	8
07	C	6	15	2	18	3
08	F	6	1	4	8	4
09	D	4	4	3	17	10
10	D	4	8	19	19	8
11	B	8	5	9	18	4
12	I	6	16	14	3	2
13	G	4	6	5	17	11
14	E	5	18	11	13	4
15	F	1	10	9	12	8
16	A	9	12	17	14	1
17	D	5	3	6	8	6
18	B	5	8	12	3	11
19	C	7	15	12	8	9
20	G	1	4	8	6	12
21	F	7	12	10	5	1
22	G	2	3	16	16	2
23	H	5	6	19	1	4
24	D	3	16	11	2	3
25	F	4	14	8	15	9
26	G	10	4	15	7	8
27	B	2	20	19	5	2
28	B	8	16	2	3	5
29	G	6	16	10	3	1
30	F	5	1	1	3	10
31	J	4	19	18	1	8
32	H	10	6	14	3	7
33	I	2	3	10	3	2
34	E	4	2	8	1	8
35	E	9	8	9	9	5
36	F	4	8	15	13	1
37	G	2	12	8	5	3
38	E	2	16	11	1	2
39	D	8	13	8	6	4
40	G	3	6	9	10	1
41	I	6	14	1	11	8
42	H	7	8	3	10	3
43	F	9	9	9	10	9
44	A	8	11	8	12	11
45	C	2	5	5	7	18
46	G	6	2	7	3	2
47	J	5	4	3	10	16
48	H	8	5	2	14	8
49	B	2	6	7	1	1
50	I	1	1	9	6	13
51	G	4	4	11	9	6
52	L	3	7	2	7	5
53	E	12	6	6	10	12
54	J	10	3	9	8	10
55	K	8	5	5	4	8
56	H	7	6	3	5	7
57	A	3	8	12	2	6
58	F	1	12	17	1	9
59	F	6	10	8	6	14
60	C	5	9	2	3	8

VI. EXPERIMENTAL RESULTS

Three strategies (Greedy, Tabu Search and Simulated Annealing) were designed with the end state (i.e. no more steps were possible). If a strategy finished before given time it was continuously re-run and the best result selected. The number of runs significantly varied per strategy, especially in the lower sizes of the solutions space (see Figure 3).

The Greedy algorithm was the fastest strategy, testing the highest number of candidate solutions. It is especially visible in small-size problems (i.e.: the number of neighbor candidate solutions is small). However, the returned solutions have a low quality (see Figure 5).

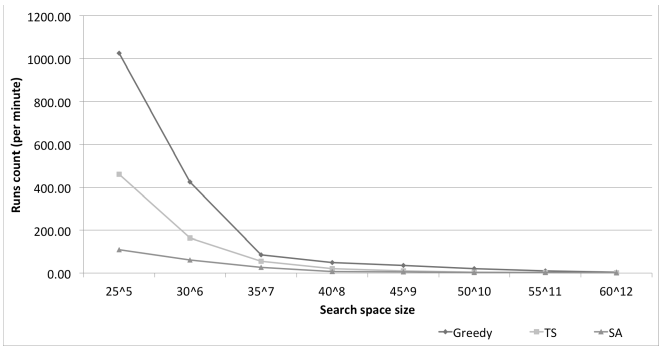

Figure 3. Runs count (per minute)

Each algorithm creates a number of candidate solutions during their run. Deciding if a candidate solution is stable (i.e.: no nodes are overloaded) tends to be the most expensive step in computations, with around 50-70% of CPU time (depending on tested strategy) spent on the validation of solution feasibility routines. As an optimization, implementations were caching newly created solutions, meaning the same tasks assignment setup is never tested twice for being stable as the result is retrieved from memory. In the chart shown in Figure 4 we have plotted the average number of unique candidate solutions created in each test scenario.

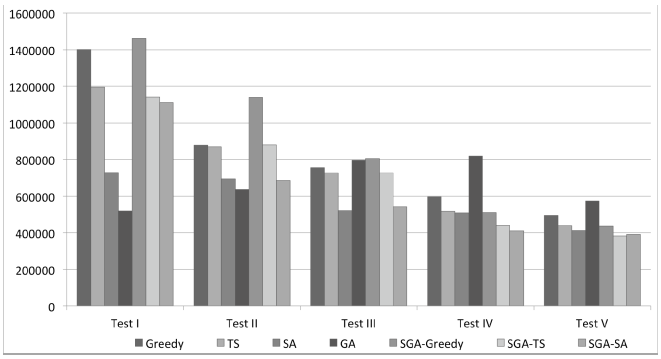

Figure 4. Unique candidate solutions created (per minute)

We designed five testing scenarios to see how each strategy copes with the increasing complexity of the problem. We have assumed that new nodes are added only when new services are deployed [21] and a demand for computing resources increases. We are simulating this scenario with enabling additional nodes (in each test two additional nodes and ten more services are added). We have assumed that the load balancer will be run periodically, thus we have selected an arbitrary computation time, after which the best-found solution was selected as output result (see Table 3).

The Full Scan strategy was used only as a benchmark if a global optimal solution was found and such limit was not imposed. The Full Scan strategy was not able to finish scenarios Test IV and Test V in reasonable time (24 hours and 5 days respectively). All other strategies' results were plotted on the chart above (the lower system transformation costs are better).

TABLE III. EXPERIMENTAL DATA – NODES CONFIGURATION

Scenario	Deployed services	Enabled nodes	Computation time	Search space size
Test I	1-20	A-D	30 seconds	20^4
Test II	1-30	A-F	1 minute	30^6
Test III	1-40	A-H	2 minutes	40^8
Test IV	1-50	A-J	4 minutes	50^{10}
Test V	1-60	A-L	8 minutes	60^{12}

VII. RESULTS

As it was demonstrated in previous research [17][19][23], when solving classical Resource-Constrained Project Scheduling Problem and its variants, more complex meta-heuristics (e.g.: TS, SA, GA) perform significantly better than simple algorithms such as Greedy.

It was confirmed in our results (Figure 5), where more sophisticated algorithms had generally better results (i.e. lower *system transformation cost*). Below we present a discussion on outcome of each strategy.

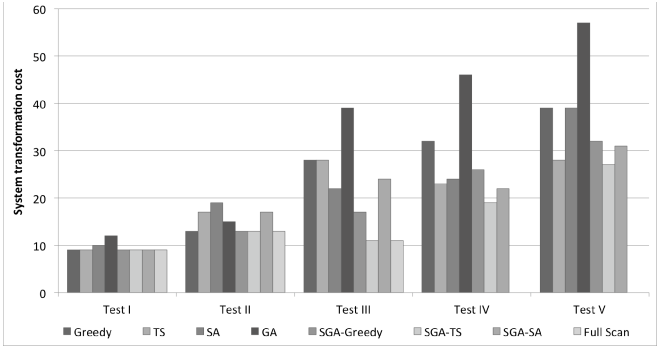

Figure 5. Results

A. Greedy – a very short execution time allowed strategy to be repeatedly run and therefore a few stable solutions were found in each test. Result solutions were of average quality; the most time consuming step was the generation of solution's neighbors (e.g.: during the Test V scenario, each step required 60 x 12 = 720 configurations to be examined).

B. Tabu Search (TS) – the main bottleneck in this approach was the last step where all of all same-value solutions had to be visited and marked as Tabu. Therefore, we have introduced a maximum limit of dull (without bettering solution) moves the strategy will perform, before the strategy gives up and returns the actual solution. Overall, the TS algorithm was working very well in small instances of a problem, which confirms results documented in [14].

C. Simulated Annealing (SA) strategy did require a much larger number of computations, often reaching only a fraction of runs in the same time as Greedy or TS. However, it did not require costly generation of all the solution neighbors, therefore re-runs count decreased at a much slower pace than above strategies. This strategy benefited the most from introducing the solution cache.

D. Genetic Algorithm (GA) variant was previously examined [23] and its main drawback is a costly generation of random solutions in the Genetic Drift step, especially when more types of resources are considered and a solution space grows in size. Performance was shown to be sufficient when examining two kinds of resources. However, due to the number of random generations required in order to create initial population the strategy performed quite poorly when four resources were introduced. As in [15], the larger the problem size, the lower the quality of the found solution was; while simpler algorithm's performance (i.a.: Greedy, TS and SA) was not impacted that much. Upon detailed examination, we have found out that randomized solutions pool often contained a significant number of solutions of low quality. They were often eliminated in the next step; however, this process had a computation cost. This became noticeably apparent in instances of a larger problem, where ten or more nodes are involved.

E. Seeded Genetic Algorithm (SGA) was the most interesting strategy in our experiment. As mentioned in GA, the randomized solutions pool contains low quality solutions and elimination of those is costly, therefore we have introduced solutions seeding to replace the previously designed Genetic Drift step [23] in the Genetic Algorithm. This allowed us to downsize the available genetic pool to 25% of its original size, which greatly reduced the computation time (around 50-70%) required to find good solutions without a reduction in quality. SGA returned the best results within the set time frame. In each case (Greedy vs. SGA-Greedy, TS vs. SGA-TS, SA vs. SGA-SA), the found solution was improved and generally less candidate solutions were examined (in Test V ca.14% less candidates were visited). In this experiment the variant with TS strategy returned the best results.

F. Full Scan strategy guarantees a globally optimum solution is found. Over the course of a research, this strategy has been heavily optimized: currently only about 9% of a solutions tree is traversed, the strategy starts moving services with the highest migration costs first, algorithm cuts leaves as soon as partial solution is deemed unstable. However, this still cannot be considered an efficient strategy due to a large number of computations required. In this experiment, Full Scan strategy was used to produce a global optima solution only in minor instances of a problem.

VIII. CONCLUSIONS

In this paper after analyzing the algorithms performance, we came to the following conclusions, which might help us design new and/or enhance already existing algorithms:

1. The meta-heuristic algorithms rely on traversing a search space in small steps, meaning the next selected solution is usually similar to current one, but usually better. It might be beneficial to give higher priority to moving already-migrated services (as they already increased migration cost) and also prioritize moving services with smaller migration cost (due to reduced impact upon total migration cost). However, this step requires building problem-specific knowledge into algorithms.

2. The initial random generation of candidate solutions is expensive. This behavior is well visible in the upward trend in the number of candidate solutions created and tested (Figure 3) in Genetic Algorithms strategy. The number of tested solutions does not correlate with the quality of solutions and better results can be achieved if the solutions pool is initially created from already pre-computed set.

3. A few strategies succeed in reaching a certain solution level and they have difficulty to move out from this or recognize a last state (e.g.: only one neighbor solution is better). Especially the Tabu Search is prone to this issue. In this implementation we encountered a counter of steps without increasing quality of solution. When an arbitrary limit of steps is reached, strategy returns the current solution. However, we believe this can be handled in more intelligent way.

In our experiments, we tested the load balancer on a medium-sized networked system and found it capable of generating a huge (60^{12} possible combinations) solution search space. We have also shown that increasing the number of tested resources did not hinder the performance of examined meta-heuristic strategies. In [23] we tested two resources, and in [24] we tested three resources. Finally, in the current experiments a four-resource metric was used. Without doubt there is an opportunity to develop this area of research further and to focus on even more complex configurations.

REFERENCES

[1] Boctor, F. (1990) "Some efficient multi-heuristic procedures for resource-constrained project scheduling." *European Journal of Operational Research* 49, no. 1 : 3-13.

[2] Bouleimen, K. and Lecocq, H. (2003) "A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version." *European Journal of Operational Research* 149, no. 2 : 268-281.

[3] Brucker, P. Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. (1999) "Resource-constrained project scheduling: Notation, classification, models, and methods." *European journal of operational research* 112, no. 1 : 3-41.

[4] Buyya, R. Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. (2009) "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility." *Future Generation computer systems* 25, no. 6 : 599-616.

[5] Chtepen, M., Filip HA Claeys, Bart Dhoedt, Filip De Turck, Piet Demeester, and Peter A. Vanrolleghem. (2009) "Adaptive task checkpointing and replication: Toward efficient fault-tolerant grids." *Parallel and Distributed Systems*, IEEE Transactions on 20, no. 2 : 180-190.

[6] Demeulemeester, E. and Willy Herroelen. (1992) "A branch-and-bound procedure for the multiple resource-constrained project scheduling problem." *Management science* 38, no. 12 : 1803-1818.

[7] Demassey, S., Christian Artigues, and Philippe Michelon. (2005) "Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem." *INFORMS Journal on computing* 17, no. 1: 52-65.

[8] Di, Sheng, D. and Walfredo Cirne. (2012) "Characterization and comparison of cloud versus grid workloads." In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pp. 230-238.

[9] Fox, A. Rean Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica. (2009) "Above the clouds: A Berkeley view of cloud computing." *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS* 28: 13.

[10] Glover, F. (1986) "Future paths for integer programming and links to artificial intelligence." *Computers & Operations Research* 13, no. 5: 533-549.

[11] Glover, F. (1989) "Tabu search-part I." *ORSA Journal on computing* 1, no. 3: 190-206.

[12] Guo, L. Shuguang Zhao, Shigen Shen, and Changyuan Jiang. (2012) "Task scheduling optimization in cloud computing based on heuristic algorithm." *Journal of Networks* 7, no. 3: 547-553.

[13] Iosup, A. Simon Ostermann, M. Nezih Yigitbasi, Radu Prodan, Thomas Fahringer, and Dick HJ Epema. (2011) "Performance analysis of cloud computing services for many-tasks scientific computing." *Parallel and Distributed Systems, IEEE Transactions on* 22, no. 6: 931-945.

[14] Józefowska J. Grzegorz Waligóra, and Jan Węglarz. (1996) "A tabu search algorithm for discrete-continuous scheduling problems" *Modern Heuristic Search Methods*: 169-182.

[15] Józefowska, J. Marek Mika, Rafal Różycki, Grzegorz Waligóra, and Jan Węglarz. (1998) "Local search metaheuristics for discrete–continuous scheduling problems." *European Journal of Operational Research* 107, no. 2: 354-370.

[16] Józefowska, J. Marek Mika, Rafał Różycki, Grzegorz Waligóra, and Jan Węglarz. (2001) "Simulated annealing for multi-mode resource-constrained project scheduling." *Annals of Operations Research* 102, no. 1-4: 137-155.

[17] Józefowska, J. Marek Mika, Rafał Różycki, Grzegorz Waligóra, and Jan Węglarz. (2002) "A heuristic approach to allocating the continuous resource in discrete–continuous scheduling problems to minimize the makespan." *Journal of Scheduling* 5, no. 6: 487-499.

[18] Kolisch, R. and Sönke Hartmann. (1999) Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. Springer US

[19] Leung, J. ed. (2004) Handbook of scheduling: algorithms, models, and performance analysis. CRC Press

[20] Limoncelli, T. Strata R. Chalup, and Christina J. Hogan. *The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems*. Vol. 2. Addison-Wesley Professional, 2014.

[21] Mishra, A. Joseph L. Hellerstein, Walfredo Cirne, and Chita R. Das. (2010) "Towards characterizing cloud backend workloads: insights from google compute clusters." *ACM SIGMETRICS Performance Evaluation Review* 37, no. 4: 34-41.

[22] Moreno, I. Peter Garraghan, Paul Townend, and Jie Xu. (2013) "An approach for characterizing workloads in google cloud to derive realistic resource utilization models." In *Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on*, pp. 49-60.

[23] Sliwko, L. (2008) "A reinforced evolution-based approach to multi-resource load balancing." *Journal of Theoretical and Applied Information Technology*, Vol. 4, No. 8: 717-724.

[24] Sliwko, L. and Zgrzywa, A. (2007) "Multi-resource load optimization strategy in agent-based systems" *Lecture Notes in Computer Science* 4496: 348-357.

[25] VMware (2015) "Verifying sufficient free disk space for an ESX/ESXi virtual machine (1003755)" *VMware Knowledge Base*, Article id: 1003755. Retrieved 28.04.2015.

[26] Weinberger, E. (1990) "Correlated and uncorrelated fitness landscapes and how to tell the difference." *Biological cybernetics* 63, no. 5: 325-336.