JOB SCHEDULING FOR TAAS PLATFORM: A CASE STUDY

Paweł LAMPE, Jarosław RUDY

Abstract: In this paper, we consider a software testing platform employing cloud computing services with homogeneous-treated machines based on real-life system. We describe the system and its characteristics. We treat the testing process as a job scheduling problem and develop a mathematical model for this problem. Then we assess the quality of the model by comparing with the data collected from the actual system. We also apply Simulated Annealing metaheuristic for the purpose of scheduling jobs and compare it with several constructive algorithms against several goal functions.

Keywords: cloud computing, testing as a service, discrete optimization, job scheduling

1. Introduction

Software testing is one of the most important steps in the software development life cycle and in certain cases may require 40–70% of the total development costs [1]. The importance of software testing has increased over years, resulting in more advanced testing techniques. Automatic software testing and providing stable and uniform testing environment are examples of areas of interest in this topic. One of possible solutions for achieving above goals is the use of cloud computing. Existing cloud models like Software as a Service (SaaS) can be extended to obtain Testing as a Service (TaaS) model [2]. Existence of such system naturally raises some issues with managing the workflow of the testing process, leading to several discrete optimization problems.

In this paper, we consider a private TaaS system used by Nokia corporation with the main purpose of improving the workflow of the testing process. To this end we describe the system and its most important properties before constructing a mathematical model. The developed model could be then used to simulate the system and test various optimization algorithms. We model the optimization of workflow as a specific job scheduling problem and we consider several possible goal functions.

The remainder of this paper is organized as follows. In section 2 we present a brief literature overview, focusing on scheduling problems in cloud computing. In sections 3 and 4 we describe the system and formally model the problem of managing the workflow of the system as a job scheduling problem. In section 5 we describe the solving methods used on the developed model. Section 6 contains research concerning the quality of the developed model and performance of the solving methods. Finally, section 7 contains conclusions.

2. Literature overview

For the past two decades cloud computing has been a quickly developing concept, attracting a great deal of scientific and business attention. It can serve variety of roles, but generally allows companies easy access to vast computing resources (ranging from hardware to virtual environments and software) regardless of private resources of the company. Its ability to provide flexible, reliable and scalable solutions has been proven by

IT giants such as Amazon, Google, Microsoft, IBM and HP among others. In result, a great number of implementations of the idea of cloud computing were created. More information about new developments and challenges in creating and managing clouds can be found in review by Zhang et al. [3].

Testing as a Service (TaaS) is a relatively new concept, aimed as tailoring cloud resources for use with software testing. TaaS can be used for wide range of software, from tests for telecommunication companies (as shown in this paper) and mobile devices [4] to the test of clouds themselves [5]. Detailed overview, and study of possibilities of TaaS was shown by Gao et al. [6].

One of the most important aspects in cloud computing is workflow management, which can be understood as assignment of tasks to cloud resources. Since clouds are usually distributed systems, this workflow management problem can be understood as a specific online job scheduling problem for sophisticated computer networks (like grids). Due to widespread use of clouds, job scheduling in cloud and grid environments have been the target of considerable attention of researchers around the world. Here, we will name just a few examples.

Pooranian et al. [7] developed GLOA evaluation algorithm inspired by behavior of leaders in social group to minimize makespan in a geographically spread grid. Similarly, Mateos et al. [8] developed Ant Colony Optimization-inspired algorithm to minimize weighted flowtime in cloud environment for a Parameter Sweep Experiments application. The research performed on real PSE data show decreased completion time compared to other methods. Taheri et al. [9] employed Artificial Bee Colony method for both scheduling of jobs and replication of data in grid computing. Computer experiments against three benchmarks proved the superiority of the developed method.

Most of approaches consider only-criterion goal functions, however multi-criteria approaches also appear. For example, Rudy and Żelazny [10] developed an online hybrid memetic algorithm based on NSGA-II method enhanced by local search procedure for network scheduling problem. They considered several criteria (mean penalty of tasks, makespan, maximum penalty among others) and used to define four different goal functions, each composed of two or three criteria before testing the performance of the developed algorithms against commonly used constructive algorithms.

Less common, but not less important are hybrid approaches, combining more than one method. For example, Shojafar et al. [11] performed job scheduling for cloud environment using genetic algorithm method, enhanced by fuzzy theory. Fuzzy rules are used to transform linguistic terms (like "low length of job") into fitness value ("medium", "adequate" or "inappropriate"). More information about various methods of solving job scheduling and workflow management problems in grid and cloud computing can be found in works [12,13].

3. System description

The system under consideration is a private TaaS platform in Nokia corporation. Its purpose is to provide automatic testing capabilities for testers and developers of the management plane (M-Plane) software which is crucial part of the Nokia’s base station product. The system can be divided into three parts: (1) cloud resources, (2) TaaS toolkit and (3) end users. Elements (2) and (3) are in-house (provided by Nokia) although they are distinct departments in the structure of the company. Element (1) is provided by an external vendor.

3.1. Cloud resources and TaaS toolkit

Cloud service is provided by a few data centers clustered around a single geographic region. Such configuration is beneficial for the platform, ensuring low and reliable network latencies. The cloud is heterogeneous by default, allowing for a broad variety of obtainable virtual machine configurations. It is possible to launch instances ranging from very small (1 core, 2 GB of RAM, 25 GB of disk capacity) to very large (40 cores, 200 GB of RAM, 1280 GB of disk capacity). However, for the software testing process uniform configurations with 2 cores, 8 GB of RAM and 25 GB of disk capacity are used. All configurations use the same operating system. Distributed storage is provided by the vendor's distributed object storage solution compatible with Amazon's simple storage service (S3).

The TaaS toolkit serves as overlay between the cloud resources and the end users. It is a set of software tools dedicated to organizing and processing the whole platform workflow across distinct layers of the TaaS architecture. Currently all the tools are command line interface (CLI) applications developed within the company. They have various purposes such as test suite gathering, handling and monitoring of users' requests, job scheduling and dispatching, monitoring and management of the cloud resources and providing feedback among others. The toolkit also includes the second, in-house distributed storage solution based on SFTP service.

3.2. System workflow

The input of the system are test suites, each composed of at least one test case. As the platform is dedicated for integration testing, every test case is assumed to be coarse-grained. It means that the single test case needs exclusive access to a single virtual machine for the time of its execution. This strong assumption cannot be weakened since M-Plane software integration testing involves many executables and use of low level system calls.

Test suites are provided by the testers, developers and continuous integration robots, which are generally referred to as clients. To be processed by the TaaS platform, test suites must be submitted to the system. Such submissions can be performed by using a TaaS toolkit. The most important tool is local job dispatcher, which resides on each client machine. The purpose of the dispatcher is to map the test cases to the cloud resources. Such solution is insufficient as the local dispatchers invoked by the clients are competing with each other for access to the virtual machines. Furthermore, the dispatchers are not aware of each other existence. Hence, such architecture makes it impossible to implement more sophisticated forms of coordination and optimize the workflow. The overview of the architecture of the system and testing process is shown in Fig. 1.

3.3. System properties

The described system is real-life platform and have been working for some time, making it possible to perform case study and obtain some statistical data, which could prove useful in developing solving methods. In this subsection, we will discuss several such properties. This data (henceforth called history) was collected over the span of 30 days. This history contained 2 021 104 test cases grouped into 24 718 test suites (yielding 55 test cases per test suite).

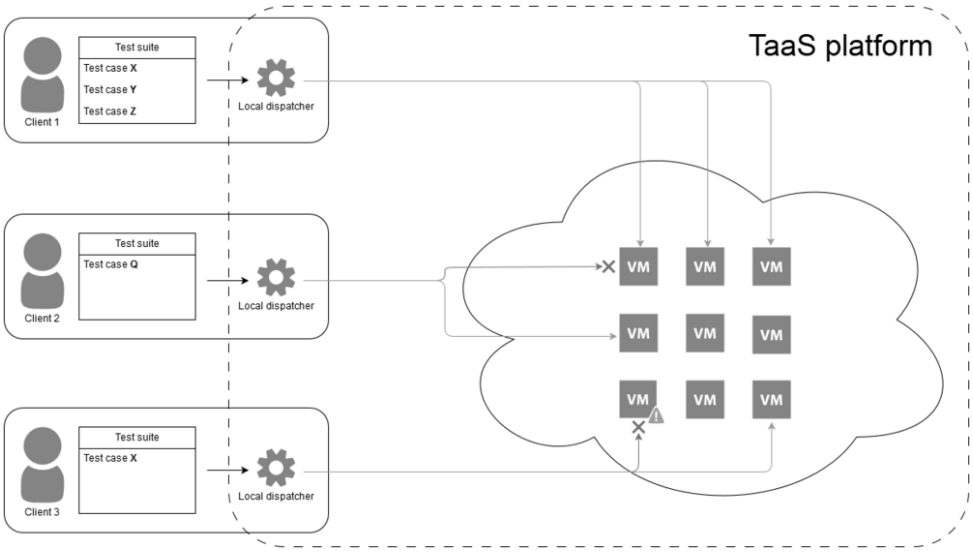

Fig. 1. Structure of considered TaaS system, showing clients, test suites, local dispatchers, cloud resources and submitting of test cases

Let us start with the analysis of the execution times (durations) of test cases. The histogram of execution times collected from the 30-day history of the running system is shown in Fig. 2. We see that a single test case may require anywhere between a few seconds to 10 minutes to complete execution. However, almost all test cases complete in under 3 minutes and majority of them takes no more than a minute. Moreover, duration of a test case is not fixed, instead it is dependent on the result of the test case (success, failure, error code) and the current state of the cloud (e.g. network transfer). Shown durations does not include the initial setup step where data must be transferred between client and the particular cloud resource. However, in practice such setup times are very small – ranging from 1 up to a few seconds. Thus, in this paper setup times are treated as constants and added to duration of test cases.

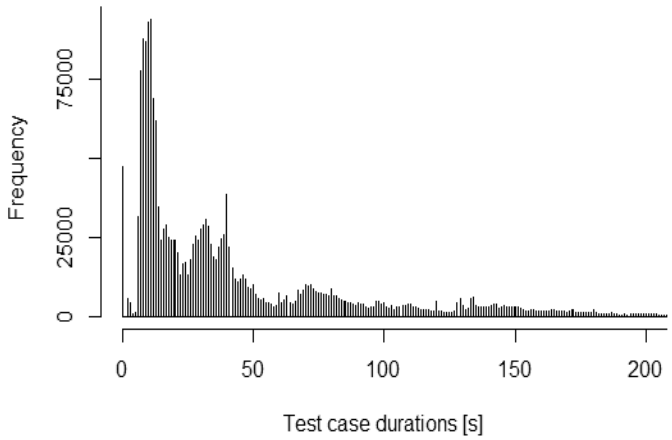

Fig. 2. Histogram of execution times of test cases

Next, and one of the most crucial, facts about the considered system is that a single test case will most likely be executed again in the future when the next version of software is tested. In result, test cases are repetitive, executed many times during system activity. This is extremely important as the test case durations are not known a priori. In fact, the longer system is working the easier it is to durations of test cases as they are more likely to appear in the history. Of course, history recording is required for this and such technique will not work for new, not previously encountered test cases.

In order to research this particular property, it is convenient to define several notions. First, let each define test cases types (or test types for short). For example, in Fig. 1. there are five test cases, but only 4 test types: *X*, *Y*, *Z* and *Q*. Test type *X* occurs twice, so second occurrence of test case of that type is a repetition. Next, let us define a parameter called repetition coefficient for each test suite. Let test suite *S* contain *m* test cases. Now if exactly *n* of those test cases are repetitions (i.e. test cases of their test type were executed at least once before) then repetition coefficient is given as:

$$R(S) = \frac{n}{m}. \tag{1}$$

Naturally, value of repetition coefficient ranges from 0 (0% of test cases in a suite occurred before) to 1 (100% of test cases occurred before). Now we can define a function where arguments are subsequent test suites (in history order) and the values are repetition coefficients for each test. Such function is presented in Fig. 3.

We can see that at the beginning test suites have *R(S)* near zero, but after several hundred test suites the values are approaching one. Let us also note that the function is not increasing or even non-decreasing – completely new test cases may appear, making it possible for subsequent test suites to have lower *R(S)* than the test suite immediately before it. However, such events are rare.

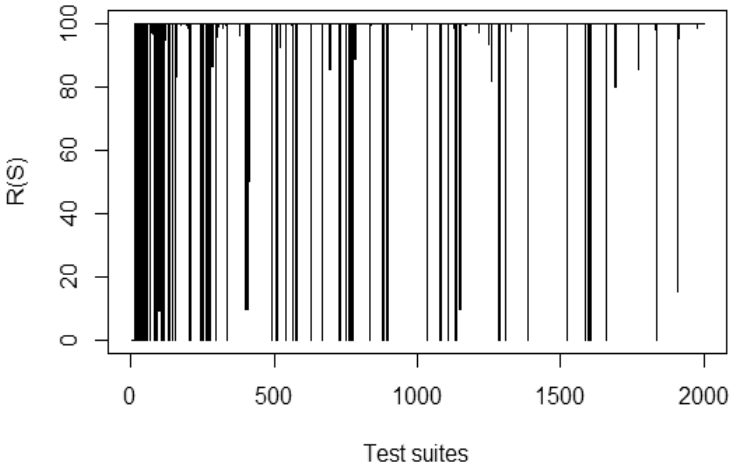

Fig. 3. Function of repetition coefficient for subsequent test suites

In addition to Fig. 3. let us present some more data concerning test case repetition. In aforementioned history of 2 021 104 test cases there were 6520 different test types in total. A single type had between 0 to 4566 repetitions (appeared in 1 to 4567 test suites) with 307 repetitions on average. Moreover, 588 test types appeared in only one test suite (this is 9% of all test types and only 0.03% of all executed test cases). A test suite had 81.5 repeated test cases on average, resulting in *R(S)* exceeding 99% of test cases of that suite. After 350 test suites, average *R(S)* exceeded 83%. We conclude that recorded history can be used to estimate test case durations for the most of system lifetime.

4. Problem description

The problem considered in this paper can be described as a specific type of online job scheduling problem. There is a set $M = \{ 1, 2, \dots , m \}$ of m homogeneous machines. Next, a set J of jobs (test suites) to be completed is given. In general, the set J is given online (i.e. jobs can be added to the set as it is processed). For the purpose of this paper we can assume time boundaries. Thus, set $J = \{ 1, 2, \dots , n \}$ has fixed size and consists of n jobs. Job j consists of t_j operations (test cases) numbered from 1 to t_j. Job j arrives at time a_j. Let us also define sequence $T = \{ 1, 2, \dots , t \}$ of all operations, ordered by the jobs i.e. operations from 1 to t_1 belong to job 1, operations from t_1+1 to $t_1 + t_2$ belong to job 2 and so on. The total number of operations is given by t:

$$t = \sum_{j=1}^{n} t_j \ . \tag{2}$$

We assume the processing time of each operation is fixed. Processing time of the i-th operation in set T is given by p_i. Alternatively, processing time of i-th operation of j-th job is given by p_i^j .

Each of the operations in T can be processed by any machine and the order of execution of operations in a job does not matter, similarly to the order of execution of jobs. However, execution of any job must be preceded by copying the necessary data files to the cloud. This is represented by a job setup time s (the same for all jobs). Also, let set Z_j designate all operations from job j.

The solution to the problem is given as a schedule which compromises of vector of operation start times S of size t. S_i indicates the start time of operation i, according to the ordering in set T. Similarly, we can define vector C of completion times of operations. Let us notice that $C_i = S_i + p_i$. The schedule must be feasible, meaning that (a) any operation at any time can be executed by at most one machine, (b) any machine at any time can execute at most one operation, (c) operations cannot be interrupted (d) any operation cannot start execution before the arrival (release) time of its job. The goal is to find a feasible schedule S^* which minimizes the given goal function K:

$$S^* = \min_{S \in \Pi} K(S), \tag{3}$$

where Π is the set of all schedules.

We consider four different goal functions. Let F_j be the flowtime of job j defined as follows:

$$F_j = C^j - S^j \,, \tag{4}$$

where C^j and S^j are start and completion times of job j computed as:

$$S^j = \min_{i \in Z_j} s_i^j \,, \tag{5}$$

$$C^j = \max_{i \in Z_j} c_i^j \,, \tag{6}$$

where s_i^j and c_i^j are start and completion times of the i-th operation in j-th job.

The four criteria are: total (average) job flowtime $\sum F$, minimum job flowtime $\min F$, maximum job flowtime $\max F$ and standard deviation of job flowtime σF, given respectively as:

$$\sum F = \sum_{j=1}^{n} F_j \,, \tag{7}$$

$$\min F = \min_{j \in J} F_j \,, \tag{8}$$

$$\max F = \max_{j \in J} F_j \,, \tag{9}$$

$$\sigma F = \sqrt{\frac{1}{n} \sum_{j=1}^{n} \left(F_j - \overline{F} \right)^2} \,. \tag{10}$$

5. Solving methods

In order to solve the problem, we have considered both simple constructive algorithms as well as more advanced metaheuristics. Thanks to repetition coefficient exceeding 99% for relatively short history, we assume that all algorithms are clairvoyant i.e. processing times of all operations all known in advance.

We start with constructive algorithms as they are simpler and one of them is already employed for use in the actual system (although performed by local dispatchers). All constructive algorithms share basic behavior: they collect incoming operations in a buffer and wait for available machines. Machine becomes available when it completes all operations currently assigned to it (machines are also available at the beginning of the algorithm). When both at least one machine is available and at least one operation in the buffer is awaiting assignment, then one operation from the buffer is chosen and assigned to

arbitrary chosen available machine. Thus, all constructive algorithms work as per Least Busy Machine strategy and differ by the method of choosing the operation to assign.

Perhaps the simplest of the constructive algorithms used in this paper is C_{rnd}. It has no criterion of assignment and hence it assigns a random operation from the whole buffer to the next available machine. This might remove some form of bias other constructive algorithms may possess, but also makes the C_{rnd} algorithm non-deterministic and highly unreliable and thus is rarely used in practice. However, when run enough times and using the best-found result C_{rnd} can perform well and might be used as a reference point.

Next constructive algorithm is C_{max} which is currently employed in company for dispatching of jobs on local machines. It maintains a queue of operations for each buffered job and operations in each queue are sorted in ascending order (hence the name of the algorithm). When a machine becomes available a queue is chosen at random and the operation from the end of that queue is assigned to that machine. This version of C_{max} is thus non-deterministic as well, but it was made that way to represent the algorithm currently employed in the company – local dispatchers compete for the cloud resources and might submit operations at random. Lastly, we consider C_{min} algorithm which works the same as C_{max}, but sorts the queues in descending order.

Next we consider metaheuristic methods, which are slower and usually probabilistic methods but are easier to control and often provide better results. As the target algorithm must be both resilient and simple we have decided to use S*imulated Annealing* (SA) method, which is commonly used for both discrete and continuous optimization problems. As reported in [14], SA is a mature optimization method and its utility both in theory and practice was proven over the years. It has been also very successful in the field of job scheduling. As the computer hardware gets more powerful over time it is also desired for algorithm to be susceptible to parallelization. An example of fine-grain parallelization using SA method can be found in [15] where authors present two effective methods of parallel objective function computation.

The SA metaheuristic used in this paper is a classical form of algorithm dedicated to job scheduling problem. The main assumption is that the operation execution times are known a priori. The solution is represented as a matrix where rows represent machines and columns represent operations assigned to the machines. Initial solution is created using constructive algorithm C_{max} described above. Every new solution is obtained from the previous one by either swap of random two operations or move of random operation from machine m_x to m_y into random column. The cooling scheme take the form of $\alpha(t) = t_0 - i/I$ where i is the current iteration number and I is the total number of iterations (fixed at 100 000). For an acceptance probability calculation, a classical exponential function was used.

6. Computer experiment

In this section, we present two computer experiments. First is aimed at presenting the quality of the developed model of the problem by comparing it with the historical data collected from the actual system. In the second experiment, we compare the performance of constructive algorithms and SA metaheuristic for a given queue of jobs/operations.

We start by the evaluation of the quality of the developed model. To this end we consider instance consisting of a fragment of the historical data collected from the actual system. This fragment covers the span of several hours during which the number of machines remained constant. This instance contained 26554 operations and 188 jobs. It is

important to note that on average 30% of operations of a given job were new i.e. had unknown processing times. The data contains not only the jobs, operations and their arrival times, but also recorded info about each job/operation start and completion times.

Using the aforementioned instance, we have computed the values of the four goal functions that were obtained during the real-life work of the system. Note that actual system uses local dispatchers on client machines and those dispatchers employ C_{max} constructive algorithm. Thus, that result was compared with the C_{max} algorithm applied to our developed model for the same instance. Comparison results are shown in Tab. 1.

Tab. 1. Comparison of the actual system performance with the performance of the developed model

	$\overline{F}$	min$F$	max$F$	σF
system model	395.377	73.000	1102.000	273.298
historical data	391.085	11.000	1430.000	295.020

The average flow time and its standard deviation are very similar for both the model and the actual system (1% and 8% difference respectively), while maximum flow time shows larger difference (approximately 30%). The biggest issue is with minimum flow time, which greatly differs between the model and the actual system (664%). This might be caused by the fact that some problem properties are not taken into account by the current model. Such properties include system failures, data transfers and operation processing times which are not fixed, but dependent on the test outcome. However, we conclude that our model is fairly close to the representation of the actual system.

Next research considers the performance of various algorithms for scheduling a queue of operations. We have prepared 100 random instances, divided into five groups of 40, 80, 120, 160 and 200 operations. We have prepared 7 algorithms. Three of them are constructive algorithms C_{min}, C_{max} and C_{rnd} described in the previous section. Moreover, let C_{Bo3} indicate the result consisting of the best results from those three algorithms. The remaining four algorithms are four versions of SA metaheuristic, each tailored for a different goal function.

Our research was divided into four groups, each for a different goal function. For each goal function, we have run all 100 instances for five different numbers of machines: 4, 8, 12, 16, 20. Each algorithm was run 10 times (important because of the probabilistic nature of C_{rnd} and SA). This means that for each goal function we have performed 5 000 runs. In each run, we receive four values, one for each algorithm. Those raw results for each run are then normalized to the best (minimum) result. For example, if the raw result is 8, 10, 12 and 14 then we normalize it into 1.0, 1.25, 1.5 and 1.75. Finally, normalized results for all runs are averaged and those are presented in Tab. 2.

We see that SA metaheuristic severely outperforms all constructive algorithms and even the C_{Bo3} results. This is true for both average and minimum flow time, but is even more visible for standard deviation of job flow time. SA metaheuristic produces results 14 to 19 times better than the constructive algorithms. The only goal function for which SA is not having a big advantage over the constructive algorithms is maximum flow time, although SA still produces the best results for this goal function. Moreover, SA is viable for all goal functions, while the constructive algorithms tend to favor specific goal functions (as seen from underlined values). Thus, running all three of the constructive algorithms seems

Tab. 2. Comparison of performance of constructive algorithms and Simulated Annealing metaheuristic for scheduling a queue of operations

	$\overline{F}$	min$F$	max$F$	σF
C_{min}	<u>1.470</u>	<u>2.477</u>	1.092	19.419
C_{max}	1.474	2.561	<u>1.064</u>	18.149
C_{rnd}	1.591	3.182	1.090	<u>14.019</u>
C_{Bo3}	1.470	2.477	1.064	14.019
SA	**1.000**	**1.023**	**1.000**	**1.002**

necessary to obtain better results. We conclude that the SA metaheuristic is better choice in presence of longer queues of operations and such queues will occur more often if we reduce the number of machines (desired as it lowers the maintenance costs and difficulty of managing the entire system).

7. Conclusions and future works

In this paper, we presented a case study of software testing platform employing cloud computing. The case study included the analysis of the characteristics of the platform and well as construction of the mathematical model, portraying the testing process as a job scheduling problem. The results can be used for simulations and development of algorithms for managing the cloud platform and optimizing testing workflow. Furthermore, we presented research indicating the quality of the model as well as the potential of metaheuristic methods for this particular platform.

Future research for this project include development of more advanced models, which will take into account aspects such as variable number of machines (due to system failures), uncertain processing times and use of more complex goal functions that will correspond to the needs of the company. Another research would include development of better online algorithms. The final research path assumes making changes to the platform architecture (include decentralization) and applying the research results to the actual system.

Bibliography

1. Yu, L., Tsai W., Chen X., Liu L., Zhao Y., Tang L., Zhao W.: Testing as a Service over Cloud, Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium on, 2010, 181–188.
2. Candea G., Bucur S., Zamfir C.: Automated software testing as a service, Proceedings of the 1st ACM symposium on Cloud computing, 2010, 155–160.
3. Zhang Q., Cheng L., Boutaba R.: Cloud computing: state-of-the-art and research challenges, Journal of internet services and applications, 1, 1, 2010, 7–18.
4. Gao J., Bai X., Tsai W., Uehara T.: Mobile application testing, Computer, 47, 2, 2014, 46–55.
5. Gao J., Bai X., Tsai W.: Cloud testing-issues, challenges, needs and practice, Software Engineering: An International Journal, 1, 1, 2011, 9–23.

6. Gao J., Bai X., Tsai W., Uehara T.: Testing as a service (taas) on clouds, Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on, 2013, 212–223.
7. Pooranian Z., Shojafar M., Abawajy J., Singhal M.: GLOA: a new job scheduling algorithm for grid computing, IJIMAI, 2, 1, 2013, 59–64.
8. Mateos C., Pacini E., Garino C. G.: An ACO-inspired algorithm for minimizing weighted flowtime in cloud-based parameter sweep experiments, Advances in Engineering Software, 56, 2013, 38–50.
9. Taheri J., Lee Y. C., Zomaya A., Siegel H. J.: A Bee Colony based optimization approach for simultaneous job scheduling and data replication in grid environments, Computers & Operations Research, 40, 6, 2013, 1564–1578.
10. Rudy J., Żelazny D.: Memetic algorithm approach for multi-criteria network scheduling, Proceeding of the International Conference on ICT Management for Global Competitiveness and Economic Growth in Emerging Economies, 2012, 247–261.
11. Shojafar M., Javanmardi S., Abolfazli S., Cordeschi N.: FUGE: A joint meta-heuristic approach to cloud job scheduling algorithm using fuzzy theory and a genetic method, Cluster Computing, 18, 2, 2015, 829–844.
12. Bardsiri A., K., Hashemi S., M.: A review of workflow scheduling in cloud computing environment, International Journal of Computer Science and Management Research, 1, 3, 2012, 348–351.
13. Ma T., Yan Q., Liu W., Guan D., Lee S.: Grid task scheduling: algorithm review, IETE Technical Review, 28, 2, 2011, 158–167.
14. Dowsland K., Thompson J.: Simulated annealing. Springer, 2012.
15. Bożejko W., Pempera J., Smutnicki C.: Parallel simulated annealing for the job shop scheduling problem, International Conference on Computational Science, 2009, 631–640.

Paweł LAMPE, M.Sc.
Jarosław RUDY, Ph.D.
Department of Computer Engineering
Wrocław University of Science and Technology
50-370 Wrocław, Wybrzeże Wyspiańskiego 27
tel./fax: 71 320 27 45
e-mail: pawel.lampe@pwr.edu.pl
jaroslaw.rudy@pwr.edu.pl