A Heuristic Load Balancing Algorithm for Cloud Computing in Heterogeneous Resource Environment

Bagus Jati Santoso¹, Ary Mazharuddin Shiddiqi², Raldo Kusuma³

Department of Informatics, Faculty of Electrical Technology and Intelligent Informatics, Institut Teknologi Sepuluh Nopember, Surabaya, East Java, Indonesia

E-mail: ¹bagus@if.its.ac.id, ²ary.shiddiqi@if.its.ac.id, ³raldo16@mhs.if.its.ac.id

Abstract. Cloud computing is an alternative technology to develop web-based services. The continuous increase of internet users causes high web service traffic that leads to the requirement of good infrastructure in managing servers in cloud computing. Load balancing between servers is one of the main challenges in cloud computing across multiple nodes. The objective is to ensure that no single resource is overwhelmed nor underused. Load balancing is optimal when maximizing throughput, minimizing response times, and avoiding overloading on one of the connection lines. To solve problems in cloud computing infrastructure, we propose the utilization of two well-known heuristic algorithms to distribute loads on cloud computing. Experimental results indicated that the heuristic algorithm could distribute task loads proportionally to cloud computing resources. Our heuristic algorithm can also distribute tasks to resources with different computing resource specifications and different tasks weights.

1. Introduction

The Cloud computing application is multiplying each year that attracts many large organizations such as Amazon, Yahoo, and Google offer cloud services and have many users. The next generation of cloud computing will be measured on how effective the infrastructure and the availability of the offered services are. Load Balancing is one of the main challenges in cloud computing in distributing the workload across multiple nodes to ensure that no single resource is overwhelmed nor underused.

The characters of data transferred over cloud computing services are on-demand self-service, open network-access, resource pooling, rapid elasticity, and measured services. The on-demand self-service aims to provide services on user requests, the open network-access requires the cloud service to support heterogeneous clients, resource pooling provide a set of resources, resource allocation and practical physically supplied to the consumer, rapid elasticity enables the current resources in the cloud can be quickly provided to users, and measured services is used to monitor, control and report computing resources for planning.

A load balancing technique allocates workloads through multi-computers, clusters, links and nodes, central processing units (CPU), storage, and memory, to get optimal use of the resources, high throughput, less response time, and avoid overloading [8, 10]. This type of environment is not constrained by limited power (battery) as studied in [6, 2, 12]. Two well-known optimization algorithms can be used to meet these requirements: Genetic Algorithm (GA) and Particle Swarm Optimization (PSO). Both algorithms have similar characteristics, i.e. to randomly generate initial possible solutions and select solutions that meet a fitness function. GA’s mechanism (combines the use of discrete and continuous parameters in one problem statement) is suitable for generating a large number of possible
solutions. In contrast, PSO’s mechanism is suited to handle discrete problem optimization [7]. GA finds the optimal solution by three steps: selection (chromosome selection of two parents from the population), crossover (crossover probability of parents to generate new offspring), and mutation (mutate parents to generate new offspring) [4, 15]. Each generation will be examined by a fitness function to select only genes with the most balanced loads between the computing resources to generate the next offspring. PSO optimizes computational resources by iteratively improving candidate solutions based on a given quality measure [3]. An improved PSO in cloud environment was proposed in [11] by considering complex networks when establishing a resource-task allocation model. PSO populates candidate solutions (particles) and moves these particles in the search space according to a simple mathematical. Each particle’s movement is not only influenced by its best local position but also the best position in the search space. The best position is updated gradually which is found by other particles. Finding which algorithm is the best in performing load balancing is challenging. A challenging factor to determine a fair comparison between the two algorithms is designing the fitness function for each algorithm. The studies in [13, 7] compared GA and PSO algorithms for load balancing in cloud computing environment, however, it did not observe the performance and resource consumption of both algorithms.

We propose a heuristic algorithm to manage cloud computing loads to optimally distribute loads between cloud computing resources based on the GA and PSO algorithms. We design a fitness function for each algorithm to evaluate its performance. Then, we present the benefits and drawbacks of the two algorithms due to the fact that each algorithm does not dominate the other algorithm. The rest of this paper is as follows: we present our technique in Section II, experiments in Section III and conclusion in Section IV.

2. Method
Our load balancing technique consists of two main parts: the pre-processing and the main program. The pre-processing calculates Time to Compute (TTC), and the main program performs a heuristic algorithm (Figure 1).

The Required Computation Time (RCT) is the time to complete a task by a virtual machine or a server. Required Computation Time (RCT) calculation is performed by involving Size of job, $S_j$, and Computer Power of Resource $i$, $C_P$, as seen in Formula 2.

$$ RCT_{ij} = \frac{S_j}{C_P} $$ (1)

2.1. Genetic Algorithm (GA)
We initiate the total number of jobs, resources, and forms of data randomly and then generate the next possible jobs, resources, and forms of data from the finest ones of the current generation [9]. We model the jobs as a series array (Figure 2). This array will later be used in the fitness function. When the RTC is obtained, the next part is to perform the heuristic algorithm. The $j$ indicates an initiated task index, while the * indicates the job to be executed by the next resource.

The GA starts its process by populating an initial generation. Then, a fitness function is applied to select the best chromosomes in the current generation.

$$ CT_{ij} = w + c $$ (2)

$$ \frac{CT}{n} = \frac{\sum CT}{n} $$ (3)

$$ F = \lambda \times maxCT + (1 - \lambda) \times \frac{CT}{n} $$ (4)

where $w$ is the waiting time, $c$ is the time to compute a job, $CT$ is the completion time, as depicted in Equation 2. totalCT is the total completion time, $n$ is the total job, while averageCT is the average completion time., as written in Equation 3. maxCT is the maximum completion time, and $\lambda$ is the minimizing factor ($0-1$). Chromosomes that pass the fitness function are selected to be used in populating the next generation, as written in Equation 4. The smaller the number of chromosomes that pass the
fitness function, the better. Chromosomes with the highest fitness scores provide a more optimum load balancing solution. For this, a high threshold value is set.

2.2. Particle Swarm Optimization (PSO)

Particles are represented in an indexed array indicating the task and the server index where the job is allocated to [1, 5]. For example, $j_1$ allocated to $r_1$ (Figure 3). The PSO starts its process by populating initial particles containing a set of load balancing solutions. These particles are evaluated using a fitness function in every population generation. We apply the same fitness function with the one used in GA.
for the sake of comparison fairness. There are two references to evaluate the particles, i.e. PBest and GBest. PBest is the current best score of a particle, while GBest is the global best score of the current population. PBest of particles is continuously updated every iteration. If a fitness score of the particle in a generation is higher than before, then the PBest of the particle is updated. However, when a fitness score of the particle in a generation is not higher than before, then the PBest of the particle is not updated. Updating the Gbest considers the PBest of all particles. If there is a particle with higher PBest than the current Gbest, then the current GBest is updated with the Pbest of the particle. However, if there is no higher PBest than the current GBest, then the current Gbest remains.

Next, we calculate the velocity of particles that update the position of the particles. Velocity calculations are performed on each dimension inside a particle. Changing the particle position aims to find a better solution. This is performed by adding up the result of each particle (Formula 5) with the current particle’s velocity. This procedure causes the velocity to keep changing every iteration if the best solution is not yet reached.

\[
V_{i,d} = V_{i-1,d} + c_1 \times r_1 \times (p_{best,d} - p_{i,d}) + c_2 \times r_2 \times (g_{best,d} - p_{i,d})
\] (5)

where \(v_{i,d}\) is the new velocity, \(v_{i-1,d}\) is the previous velocity, \(c_1\) is the learning rates for PBest, \(c_2\) is the learning rates for GBest, PBest is the best score of the particle, GBest is the global PBest score, \(p_{i,d}\) is the particle \(i\) in dimension \(d\), \(r_1\) is the random value 1, and \(r_2\) is the random value 2.

The procedures keep performing until the GBest value converges or has reached the maximum iteration. The latest GBest is regarded as the best solution to divide jobs into servers.

3. Experiments

We used dockers to simulate computing servers [14], an open-source application for consolidating files needed by software. Settings and file data supporters referred to as images. These images are collected in one location called the container. Docker is often used to develop applications due to its flexibility in various environments, e.g. different system configurations in a computer or a server. Details of the experimental environment are listed in Table 1.

We used artificial data containing a predetermined number of tasks, total computer power resources, and job sizes. We set the generation number of GA, chromosome number in a population, and parent numbers prior to running the algorithm. We also applied the same procedure in PSO by setting particles number and the maximum iteration.

| Table 1. Experimental environment |
|----------------------------------|
| **Environment** | **Description** |
| Operating system | Ubuntu 18.04.2 LTS |
| Programming | Python 3.6.9 |
| Text Editor | Visual Studio Code 1.43.2 |
| Virtualization | Docker 19.03.8 |

![Figure 4. Genetic Algorithm’s functionality experiment results](image)
3.1. Experimental Scenario
We divided the experiments into two types: functionality and performance experiments. The functionality scenario aims to examine whether the system has met the requirements. The performance experiment aims to compare GA and PSO performances to provide the best solution to balance loads between servers.

3.2. Functionality testing
The GA shared the jobs with proportionally both in homogeneous and heterogeneous computational resources (Figure 4). Jobs are distributed equally in a homogeneous environment (Figure 4(a)) and proportionally in a heterogeneous environment (Figure 4(b)). For example, three identical computers with the same computational resources received the same number of jobs. On the other hand, jobs are distributed based on computational resources of each computer in a homogeneous environment. For example, computers with higher computational resources receive more jobs than other computers.

The PSO divided the jobs in a balanced manner with the same job size (Figure 5). Jobs are distributed equally in a homogeneous environment (Figure 5(a)) and proportionally in a heterogeneous environment (Figure 5(b)). For example, there are jobs with the same sizes that are all the same; then, all computers will receive an equal number of jobs. When various job sizes are given, they are shared proportionally. For example, a computer was only given one job with size nine that other computers with more number of smaller jobs.

3.3. Performance experiments
The addition of tasks requires more execution time and memory to find the best solution for both algorithms. We increased the number tasks to process and observed the performance of both algorithms (Figure 6). The execution time of GA is more stable and faster than PSO because due to its quick nature in finding the best solution (Figure 6(a)). However, GA requires more memory than PSO to perform the same number of jobs (Figure 6(b)).
Next, we experimented with increasing chromosome or particle number and steady number of tasks to process (Figure 7). GA is still faster and more stable than Particle Swarm Optimization to converge (Figure 7(b)), but GA consumes more memory than PSO (Figure 7(a)). We conclude that crossover and mutation are the determining factors for the GA to outperform PSO. Lastly, we use the same number of jobs with an increasing generation or iteration numbers (Figure 8). Here, Genetic Algorithm still outperforms Particle Swarm Optimization in terms of execution time (Figure 8(a)), but with the same trade-off, i.e. GA requires more memory consumption than PSO (Figure 8(b)). We conclude that crossover and mutation are the determining factors for the GA to outperform PSO.

4. Conclusion

We proposed a load-balancing method based on GA and PSO. The GA based load balancing method starts by initiating populations containing solutions (chromosomes) of job distribution scenarios. Then, the chromosomes are evaluated using a fitness function. Chromosomes that pass the function are then crossoverted and mutated to produce chromosomes in the next generation.

This procedure is performed until we find convergent solutions or the maximum number of generations is reached. The PSO algorithm based load balancing method initializes particles with their movements to find the best solution for load sharing. Every particle will be updated in every iteration along with its velocity. The movements keep particles continue moving until the fitness values converge or iteration number is maximized. Experimental results indicate that the GA requires shorter execution time than PSO. This is due to the nature of GA that can produce convergent values faster than PSO.
However, the high performance of GA requires more memory than PSO to accommodate the generated chromosomes as well as the crossover and mutation processes. Based on the experiment results, we conclude that GA is more suitable for generating fast solutions, while the PSO is more suitable for lower resource environments.

References
[1] Alguliyev R M, Imamverdiyev Y N and Abdullayeva F J 2019 PSO-based load balancing method in cloud computing Automatic Control Comp. Sci. 53 pp 45–55 DOI: 10.3103/S0146411619010024
[2] Cuzzocrea A, Gaber M M and Shiddiqi A M 2014 Adaptive data stream mining for wireless sensor networks Proc. of the 18th Int. Database Engineering & Applications Symp. (IDEAS ’14) (New York: Association for Computing Machinery) pp 284–7 DOI: 10.1145/2628194.2628213
[3] Dave A, Patel B, Bhatt G and Vora Y 2017 Load balancing in cloud computing using particle swarm optimization on Xen Server Nirma University Int. Conf. on Engineering (NUICONE) Ahmedabad pp. 1-6 DOI: 10.1109/NUICONE.2017.8325618.
[4] Dasgupta K, Mandal B, Dutta P, Mandal J and Dam S 2013 A genetic algorithm (GA) based load balancing strategy for cloud computing Procedia Technol. vol 10 pp 340–7 DOI: 10.1016/j.protcy.2013.12.369
[5] de Almeida B S G and Leite V C 2019 Particle swarm optimization: a powerful technique for solving engineering problems Swarm Intelligence - Recent Advances, New Perspectives and Applications ed J D Ser et al (London: IntechOpen) DOI: 10.5772/intechopen.89633.
[6] Gaber M M and Shiddiqi A M 2010 Distributed data stream classification for wireless sensor networks Proc. of the 2010 ACM Symp. on Applied Computing (SAC ’10) (New York: Association for Computing Machinery) pp 1629–30 DOI: 10.1145/1774088.1774439
[7] Hassan R, Cohanim B, de Weck O and Venter G 2012 A comparison of particle swarm optimization and the genetic algorithm 46th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conf. pp 804–7 DOI: 10.2514/6.2005-1897
[8] Lagwal M and Bhardwaj N 2017 Load balancing in cloud computing using genetic algorithm pp 560–5 DOI: 10.1109/ICCCONS.2017.8250524
[9] McCall J 2005 Genetic algorithms for modelling and optimisation J. of Computational and Applied Mathematics Vol 184 no 1 pp 205–22 ISSN 0377-0427 DOI: 10.1016/j.cam.2004.07.034.
[10] Nuaimi K A, Mohamed N, Nuaimi M A and Al-Jaroodi J 2012 A Survey of load balancing in cloud computing: challenges and algorithms 2012 Second Symp. on Network Cloud Computing and Applications, London pp 137–42 DOI: 10.1109/NCCA.2012.29
[11] Pan K and Chen J 2015 Load balancing in cloud computing environment based on an improved particle swarm optimization 2015 6th IEEE Int. Conf. on Software Engineering and Service Science (ICSESS), Beijing pp 595–8 DOI: 10.1109/ICSESS.2015.7339128
[12] Parenreng J M, Syarif M I, Djanali S and Shiddiqi A M 2011 Performance analysis of resource-aware framework classification, clustering and frequent items in wireless sensor networks Proc. of the Int. Conf. on e-Education, Entertainment and e-Management, Bali pp 117–20 DOI: 10.1109/IceEEM.2011.6137858
[13] Pathak K and Vahinde G 2015 Comparison of particle swarm optimization and genetic algorithm for load balancing in cloud computing environment Int. J. of Research in Computer & Information Vol I Issue I DOI: 10.5772/intechopen.89633
[14] Singh S and Singh N 2016 Containers & docker: emerging roles & future of cloud technology 2016 2nd Int. Conf. on Applied and Theoretical Computing and Communication Technol. (iCATcT), Bangalore pp 804–7 DOI: 10.1109/ICATCCT.2016.7912109
[15] Thirumoorthy K, Selvakumar B and Umakanth N 2018 Genetic algorithm based task deployment approach for load balancing in cloud environment Int. J. of Pure and Applied Mathematics,
vol 118 no 20 pp 371–7