Elasticity Based Scheduling Heuristic Algorithm for Cloud Environments

Ali Al Buhussain, Robson E. De Grande, Azzedine Boukerche
PARADISE Research Lab
University of Ottawa
Ottawa, ON. Canada
E-mails: aalbu052@uottawa.ca, {rdgrande, boukerch}@site.uottawa.ca

***Abstract*—Cloud computing environments mainly focus on the delivery of resources, platforms, and applications as services to users over the Internet. Cloud promises users access to as many resources as they need, making use of an elastic provisioning of resources. The cloud technology has gained popularity in recent years as the new paradigm in the IT industry. The number of users of Cloud services has been increasing steadily, so the need for efficient task scheduling is crucial for maintaining performance. In this particular case, a scheduler is responsible for assigning tasks to virtual machines efficiently; it is expected to adapt to changes along with defined demand. In this paper, we suggest an elastic scheduler that is able to alter its focus based on the current requirements demanded by the cloud service provider and the user of those services. The Elasticity Based Scheduling Heuristic (EBSH) suggested is measured against the bio-inspired optimization algorithms such as Ant Colony Optimization (ACO) and Honey Bee Optimization (HBO). Also, a networking algorithm is used in this study, namely Random Biased Sampling (RBS). The presented EBSH shows superior performance because of its ability to adapt to changes.**

***Keywords*-Scheduling; Elastic Scheduling ;Bio-inspired Algorithms; Swarm Optimization.**

I. Introduction

Cloud computing services aim to provide resources with high availability. Those resources are scalable depending on the need of the users. Flexibility and scalability are the most important properties of cloud environments, making it a highly complex and large distributed environment. These characteristics impose massive challenges on allowing centralized governance. Hence, there is an increasing need to identify distributed solutions that are able to govern the cloud environment through local knowledge. A distributed governance system is expected to be self-organized and able to manage itself [19].

Cloud Environments provide services that can be accessed through the Internet [13]. Those services reduce the need for infrastructure and all the management of it by individual IT companies. Cloud also offers software as a service to users and companies. The cloud computing environment is seen as the next step in the IT industry. As a result, cloud computing has been given a lot of attention and effort by many IT initiatives, such as Amazon EC2, Microsoft Azure, and Google Cloud Platform.

In the cloud, the demands for resources change dynamically, and cloud providers are expected to be able to accommodate and react to these changes. Those changes can be set on deadlines for hard real-time applications, cost, which corresponds mostly to re-sizing resources according to dynamic application demands, the load of the cloud environment, and SLA agreements. Virtualization of physical resources provides the necessary dynamicity to manage the resources in the cloud platform [7]. Consequently, scheduling the assignment or task to virtual machines directly impacts on the performance of the Cloud and aids in balancing the distribution of load among the different physical servers. Thus, finding an effective scheduling scheme is crucial to enable Cloud services.

Several scheduling algorithms have been proposed for allocating resources in distributed systems. However, the context of resource management systems for cloud computing restricts the use of such algorithms to a narrower class of solutions. Bio-inspired algorithms, more specifically swarm or gang scheduling algorithms, have shown to be very suitable for the cloud environments as per results presented in some works. However, all of the previous bio-inspired algorithms lack the elasticity and adaptability to requirements changes in the cloud environment.

In this work, we conducted a series of extensive analyses in order to observe the behavior of EBSH and selected bio-inspired algorithms against extreme load and variety of environment conditions. The results of such analyses provide conditions to better understand the scenarios that influence the performance of each algorithm, giving an indication of better parametric configuration and allowing us to propose enhanced solutions that are able to cover such extreme cases, enabling enhanced overall scheduling efficiency.

The remaining of the paper is structured as follows. Section II provides a brief description of previous works. Section III provides details of the architecture and functioning of the Honey Bee Optimization algorithm. Section IV thoroughly describes AntColony algorithm. Section V presents Random Biased Sampling algorithm. Section VI describes the proposed Elasticity Based Scheduling Heuristic (EBSH). Section VII introduces the experimental scenario and discusses the results obtained. Finally, Section VIII summarizes the paper and provides future work directions.

II. Related Work

A cloud is seen as a dynamic and heterogeneous system, and scheduling heterogeneous resources in the cloud is considered an NP-hard problem [26], [6] and used to be approached simplistically, such as the use of instance replication based on the evaluation of a rule set to distributed load in Amazon EC2 [19]. However, this approach is not practical for large-scale, complex systems, possibly triggering a cascading process on policy execution [19].

The self-organization for balancing load on entities is possible with the implementation of distributed algorithms that rely on local knowledge. In recent years, nature has influenced many works on seeking out solutions to the increasing scale and complexity of the Cloud system [1]. There are two general directions that are typically followed to solve this issue: population-based or evolutionary optimization. The latter heuristics are the best in tackling scheduling in the Cloud [20], [23], including genetic algorithms (GA), Particle Swarm optimization (PSO), Ant Colony optimization (ACO), and Honey Bee optimization (HBO).

The genetic algorithms are multi-objective, and some works are focused on the cost of running tasks on the Cloud; the work described in [8] classifies tasks depending on time and budgetary cost constraints into priorities. Other works focus on the energy consumption, such as the work in [24]. Other works cluster objectives, such as the one in [12]. Reputation has also been used to introduce memory in the scheduling [17]. However, GA scheduling algorithms are slow for Cloud due the time to converge [15]. Clustering has been used based on memory consumption and computation requirements of tasks as in [27]. A GA scheduler was proposed in [5] to minimize the makespan of the tasks only. Thus, distributed, self-organized, and self-managed algorithms are needed to solve the Cloud scheduling problem.

Secondly, particle swarm optimization (PSO) is simple and fast as it only uses two aspects: velocity and position. PSO presents the fastest convergence when compared to GA and ACO [26]. The PSO takes into account only the cost, both data transmission and computational, as optimization objective. This work is similar to [25] in terms of the optimization objective but they differ in form [16] as the PSO is discrete not continuous [26]. The optimization factors were then further extended in [3] to include the makespan, cost, reliability. The only problem with [3] is that there is no dependency between the optimization factors. The study done in [21] had the objective as to reduce cost and meet deadlines. The work in [10] proposed to redefine what the index of resources used in [21] by using a renumbering based on the price of the resource and the ability to reorder. Also, [9] extended the [10] to include other scheduling objectives.

Lastly, using the behavior of ants or bees inspired the creation of distributed and self-managed algorithms, and thus they are studied thoroughly in this work. [26] stated

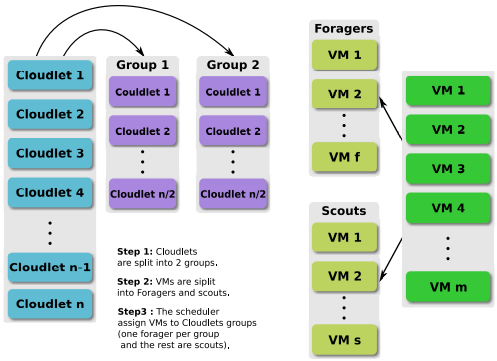


Figure 1: Honey Bee Structure.

that ACO is better than GA in terms of always finding a solution to the scheduling in the Cloud. The HBO was also chosen because of the speed in which it converges. The work conducted in [1] stated that schedulers present most bio-inspired behavior, like ant colonies, honeybees, and particle swarms, showing the viability of this type of algorithms in the Cloud environment.

The previously mentioned approaches application-oriented, so those solutions become not useful when the optimization objectives of the scheduler involve more general and adaptable conditions. Since the cloud is built to accommodate users with different needs and cloud providers with different characteristics, the scheduler must be able to easily alter its focus. Therefore, in this paper, we propose an adaptable scheduler called EBSH.

III. Honey Bee Optimization (HBO)

The basic elements in HBO scheduling algorithm are bees. They find the most profitable source and exploit it. Foragers are also considered by shifting in quality or profit in the nectar sources [22]. HBO provides a self-managed and self-organized solution, by essence decentralized, to the scheduling problem [14]. Such characteristics make HBO along with the speed of convergence a strong candidate for cloud scheduling.

HBO scheduling algorithm is basically divided into two parts, as delimited in Algorithm 1. The first part consists of the foraging behavior of the bees as they look for food sources. The second part is the scouting where the bees start their search for the best food source brought by the foragers. The HBO procedure is described in the Figure 1.

IV. Ant Colony Optimization (ACO)

In this approach, the behavior of ants in foraging for food is explored as a solution for the Cloud task scheduling. The ants search for food sources (VMs) in random. Then, they evaluate the quality of the food source and carry it back to the nest. The ants leave a chemical trail on the ground according to the quality of the food source [4]. In Cloud computing, ACO is used to conduct a search for the optimal scheduling of tasks to VMs [4], [11].

Algorithm 1 HBO VM Assignment Algorithm

Require: $Cloudlet_{list}, VM_{list}, Datacenter_{list}, fac_{LB}$
$Groups(q) \leftarrow divide(Cloudlet_{list})$
for $i = 1$ to q **do**
  $length_i \leftarrow lenghtOfGroup_K(Groups_i)$
end for
for $k = 1$ to q **do**
  $CloudLet_L \leftarrow max(Groups_k)$
  while $Groups_k \geq \{Groups_i | i = 1..q$ and $i \neq k\}$ **do**
    for $s = 1$ to n **do**
      $Datacenter_s \leftarrow select(Datacenter_{list})$
      if $fac_{LB} \leq VMsAssigned(Datacenter)$ **then**
        $assign(Cloudlet_L, Datacenter_s(VM_{leastLoad}))$
      else
        $assign(Cloudlet_L, Datacenter_{i \neq s}(VM_{leastLoad}))$
      end if
      $decrement(length_k)$
    end for
  end while
end for

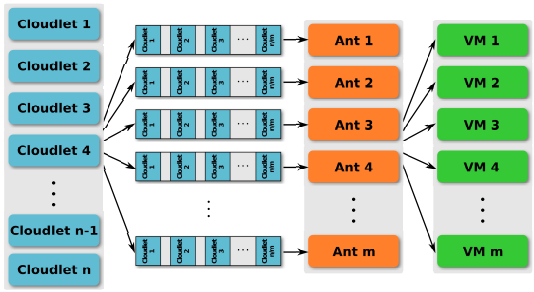


Figure 2: Ant Colony Architecture.

ACO is delimited as by the following elements:

- Problem statement: ants search for the best VM from an initial random setup with pheromone trail $\tau(0)$;
- Heuristic desirability η: the inverse of the expected execution time is used;
- Constraint satisfaction method: each ant is only allowed to visit a VM once;
- Pheromone-updating rule: each ant deposits a pheromone that depends on the quality of the solution;
- Probabilistic transition rule: the ant related to $tabu_k$ is updated by adding the visited VM in the initial step. Afterwards, Ant_k selects the next VM_j to execute $Cloudlet_i$ based on a probability.

Figure 2 illustrates in general terms the whole process of combining Cloudlets to VMs, which is described in Algorithm 2. The scheduler firstly creates the ants and then distribute the Cloudlets to each ant. Then, the ants choose the best VM based on its defined parameters. Finally, the tabu of each ant is updated with the chosen VM, and the algorithm is repeated in the following iteration.

V. Random Biased Sampling (RBS)

The RBS algorithm provides a way to construct a network of resources (VMs). These networked resources are then divided into groups, which receive a degree. This degree on an average of all resource groups must be the same achieving balanced scheduling. The created network of resources is

Algorithm 2 AntColony VM Assignment Algorithm

Require: $\alpha, \beta, max_{iterations}, Cloudlet_{list}, VM_{list}$
for i in $Cloudlet_{list}$ **and** k in VM_{list} **do**
  $pair_{Cloudleti,VM_k} \leftarrow \tau_{i,j}(0) = C$ // pheromone(C)
end for
$VM_k \leftarrow Ant_j \leftarrow randomPick(Ant_{pool})$
$Ant_j^{tabu} \leftarrow add(VM_k)$
while NOT done **do**
  for $k = 1$ to m **do**
    $VM_s \leftarrow select(Ant_k, VM_{list}, Cloudlet_{list})$
    $Ant_j^{tabu} \leftarrow add(VM_s)$
  end for
  for $k = 1$ to m **do**
    $L_k \leftarrow calculate()$
  end for
  $\tau_{i,j} \leftarrow update()$
  $pheromone_{global} \leftarrow update()$
  $increment(iterations)$
end while

self-organized and able to distribute the tasks based on only local knowledge [18].

RBS builds a graph of resources with each node containing a degree (NID) and a walk length threshold (υ). The tasks coming into the servers have an associated walk in length (ω) that is used to schedule those tasks to the appropriate resources. The NID value varies depending on the number of free resources on a given node. When a $task_i$ comes into the servers, $node_k$ only executes a task if the execution test is fulfilled $(\omega \geq \upsilon)$. If the previous condition is not satisfied, ω is incremented by one and sent to the other nodes and the execution test is reapplied. Algorithm 3 represents the general procedure of RBS in a cloud environment. Figure 3 shows the step of RBS in assigning cloudlets to VMs.

Algorithm 3 Random Biased Sampling VM Assignment Algorithm

Require: $Cloudlet_{list}, VM_{list}$, and $r \leftarrow numberofCloudlets$
for $k = 1$ to n **do**
  $Groups \leftarrow divide(VM_{list}, groupSize(number(r)))$
end for
for $k = 1$ to q **do**
  $Group_k \leftarrow ascending(WIL)$
end for
for all $Cloudlet_i$ in $Cloudlet_{list}$ **do**
  $Cloudlet_i \leftarrow random(WIL)$
end for
for $k = 1$ to m **and** i to q **do**
  if $Cloudlet_k.WIL \geq Group_i.WIL$ **then**
    $Group_i \leftarrow Cloudlet_k$
  else
    $increment(Clouldet_k.WIL, 1)$
    $Group_{i+1} \leftarrow Cloudlet_k$
  end if
end for

VI. Elasticity Based Scheduling Heuristic (EBSH)

EBSH consists of creating a scheduler that is able to change preferences or importance depending on the change in the requirements of the cloud service subscriber and the expectations of the cloud service provider as seen in

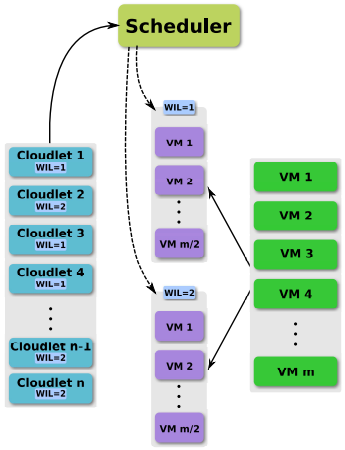


Figure 3: Random Biased Sampling Match-Making.

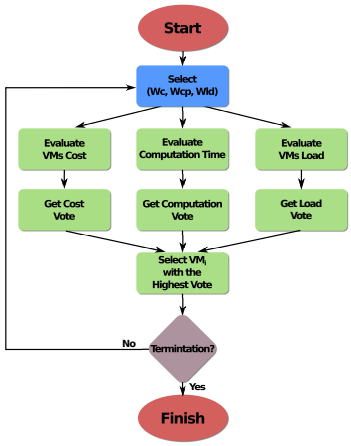


Figure 4: EBSH Execution Flow.

Figure 4. The algorithm asks the users of the cloud services to assign weights, typically between zero and one, on the certain optimization objectives. Those objectives are cost, computation. Moreover, the cloud service provider has a similar the computation and load. The meaning of those objectives are as follows:

- Cost: The cost of using VM_i to execute $Cloudlet_j$ in terms of RAM, MIPS, Storage, Bandwidth.
- Computation: The time VM_i will take to execute $Cloudlet_j$.
- Load: The number of Cloudlets assigned to VM_i.

The EBSH works as a voting system in which each optimization objective is adjusted to the weight desired. Then the virtual machine with the most or the highest vote at time T_i will be assigned the current cloudlet. This property makes EBSH stand-alone scheduler because of the ability to choose the best virtual machine for the current task. Moreover, EBSH can be used as a fixable meta-heuristic since it can be extended to arrange the virtual machines in order of highest votes and then use any PBSH to assign the highest voted virtual machines to a set of tasks. The Elasticity of EBSH to be able to work either as a heuristic

Algorithm 4 EBSH VM Assignment Algorithm

Require: $Cloudlet_{list}, VM_{list}, Datacenter_{list}$
Require: $W_{cost}, W_{computation}, W_{load}$
1: $Size(m) \leftarrow getSize(CLOUDLET_{list})$
2: $Size(n) \leftarrow getSize(VM_{list})$
3: **for** $j = 1$ to m **do**
4: **for** $i = 1$ to n **do**
5: $CtVote \leftarrow getCtVote(VM_i, VM_i, Cloudlet_j)$ // as in 1
6: $AdjCtVote \leftarrow adjustVote(W_c, CtVote)$
7: $CompVote \leftarrow getCompVote(VM_i, Cloudlet_j)$ // as in 5
8: $AdjCompVote \leftarrow adjustVote(W_{cp}, CompVote)$
9: $LdVote \leftarrow getLdVote(VM_i)$ // as in 6
10: $AdjLoadVote \leftarrow adjustVote(W_{ld}, LoadVote)$
11: $SumAdjVotesVMi \leftarrow sumOfVotes(AdjCtVote, AdjCompVote, AdjLdVote)$
12: **end for**
13: $VMToBeAssigned \leftarrow getMaximumVoteVM(SumAdjVotesVMi)$
14: $Assign(Cloudlet_j, VMToBeAssigned)$
15: **end for**

and meta-heuristic enable it to be both centralized and partially distributed. The steps followed by EBSH are:

1) The estimation of the cost for executing $Cloudlet_j$ on VMi where i(1..n) is done. The cost is estimated based on the RAM, Storage, Bandwidth, and MIPS required by $Cloudlet_j$ from VMi.
2) The execution time of $Cloudlet_j$ on VMi where i (1..n) is estimated. This estimation is based on the worst case to ensure that we have the virtual machines with the least execution time.
3) The load on each virtual machine is calculated by the number of cloudlets assigned to it. this is also based on the worst case to ensure we get the least loaded virtual machine.
4) If stand-alone scheduler
   - Convert all the optimization objectives from VM_i i (0..1).
   - Add all the votes for every virtual machine and then find the one with the maximum vote. Finally, assign that virtual machine to the cloudlet to execute.
5) if meta-heuristic
   - Convert all the optimization objectives from VM_i i (0..1).
   - Make a subset of virtual machines according to a threshold of the minimum number of votes. Then give the subset of virtual machines to any PBSH to assign tasks to that subset.

$$Cost_{Vote}^{i,j} = (CPSVM_i + CPRVM_i + CPBVM_i) \times (T_{CL_j}) \quad (1)$$

, i = 1...N, j = 1...M

Where,

$$CPSVM_i = DC_{CPS} \times size_{VM_i}, i = 1...N \quad (2)$$

$$CPRVM_i = DC_{CPR} \times RAM_{VM_i}, i = 1...N \quad (3)$$

$$CPBVM_i = DC_{CPB} \times Bw_{VM_i}, i = 1...N \quad (4)$$

The cost estimation functions are similar to the cost estimation followed by HBO.

$$Computation^{i,j}_{Vote} = \frac{(T_{CL_j} \times NOCVM_i)}{MIPSVM_i}, i = 1...N, j = 1...M \quad (5)$$

$$Load^{i,j}_{Vote} = NOCVM_i, i = 1...N, j = 1...M \quad (6)$$

All of the above equations are then used to get a vote for their optimization objective. This can be done by finding the minimum value calculated. Then this value will get a vote of 1. The others are divided by the minimum value given us the vote that is between (0 and 1). The importance of converting the optimization objectives to votes allow us to obtain different values. Thus, converting them to votes ensure fairness. Moreover, the voting system is necessary for the meta-heuristic mode.

VII. Experimental Analysis

A series of experiments have been conducted in order to evaluate the performance of AntColony, HBO, and Random Biased scheduling algorithms on a set of varied setup scenarios. The scenarios were configured with parameters that stretched over extreme situations in order to stress the algorithms and identify their limits on providing the proper allocation of virtual resources. Simulations have been conducted in order for this analysis to be conducted on extremely large scenarios; particularly, CloudSim was used as the simulator and worked as the experimental testbed.

CloudSim emerged from the need to have a cloud simulation environment that would extend the traditional distributed systems simulators (Grid and Network) [2]. Moreover, testing on real cloud environment is costly and imposes several challenges, which considerably increase the complexity in conducting large-scale experiments; several factors drive such challenges, such as (i) variation in the clouds demand, supply pattern, system size, and resources; (ii) the heterogeneous characteristics of user and QoS requirements in these dynamic environments; and (iii) variations in applications performance, dynamic, and scaling requirements. Moreover, a simulator is crucial in providing controlled scenarios in which results are reproducible over the most diverse combination of setup parameters.

A. Baseline Scheduling

The algorithm of this Base Test is a simple scheduler that assigns Cloudlets to VMs in a cyclic matter. For instance, in an attempt to allocate a set of virtual machines (vm1, vm2) for a set of Cloudlets (c1,c2,...,cn), the Base Test assigns vm1 to c1, vm2 to c2, vm1 to c3 and so forth until all Cloudlets are assigned, showing an equally distributed of load of the virtual resources.

B. Simulation Scenarios

Two scenarios have been used to analyze the EBSH and the bio-inspired schedulers fully and thoroughly. The first scenario comprises a realistic cloud environment, in which heterogeneous resources and load are most likely to exist. This scenario tests the schedulers under a range of different virtual machines with a variety of capabilities and Cloudlets with different requirements. The scenario is specifically presented to demonstrates the adaptability and the ease of control of the suggested EBSH.

The heterogeneous environment scenario was implemented to mimic real cloud environment where task and virtual machines are not similar. In other words, different workloads were submitted to virtual machines that also have different capabilities. For the setup on this scenario, a different range of parameters was used, restricting to smaller dimensions; however, such restrictions did not impact on the results obtained. Thus, the number of virtual machines was reduced to 50 and the number of Cloudlets was reduced to 5000. Tables I, III, and II list the characteristics of the VMs, Datacenters, and the tasks, respectively.

Table I: VM Characteristics in the Heterogeneous Setup

Heterogeneous VM characteristics	Values
vmMips	500-4000
vmSize	5000
vmRam	512
vmBw	500
vmPesNumber	1

Table II: Cloudlet Parameters in the Heterogeneous Setup

Heterogeneous Cloudlet characteristics	Values
cLength	1000-20000
cFileSize	300
cOutputSize	300
cPesNumber	1

Table III: Datacenter Values in the Heterogeneous Setup

Datacenter characteristics	Values
CostPerMemeory	0.05-0.01
CostPerStorage	0.004-0.001
CostPerBandwidth	0.05-0.01
CostPerPrcessing	3

C. Metrics Used in the Performance Analysis

Three metrics were used to provide an overview of the performance of the algorithms on the experiments. Each of the metric is detailed in the following subsections.

1) Scheduling Time: The scheduling time of the scheduling algorithms is obtained by retrieving the start-up time in which the algorithm has been triggered and the final time when the algorithm returned with the solution of a specific combination setup. Due to the large scale of the setup environments used in our experiments, the unit of the scheduling time is hour even though the precision of the

Table IV: EBSH Factors

EBSH factors	Weight %
W_c	0
W_{cp}	100
W_{ld}	50

retrieved metrics are on the scale of milliseconds. The times for the ACO, HBO, and RBS are roughly the same. They are larger that the base test as they require more computation to schedule the Cloudlets to the VMs.

2) Simulation Time: The simulation time corresponds to the maximum overall time that the tasks took to complete the execution. As described in Equation 7, this metric consists in the difference between earliest start execution time of a Cloudlet and finish time of the latest Cloudlet to end its execution. Please note that this metric unit corresponds to milliseconds of simulation wall clock time.

$$T_{sim} = T_{maxFinishTime} - T_{minStartTime} \tag{7}$$

Where:

- T_{sim} : simulation time of the Cloudlets.
- $T_{maxFinishTime}$: maximum finish time of the Cloudlets.
- $T_{minStartTime}$: minimum start time of the Cloudlets.

Through this simulation time metric, we can determine the effectiveness of the scheduling algorithms.

3) Time Imbalance: The time imbalance provides an overview on the fairness of the Cloudlets execution on VMs. In summary, this metric represent the variation in the execution time of the tasks. Equation 8 defines time imbalance metric used in this evaluation.

$$T_{im} = \frac{[T_{max}] - [T_{min}]}{T_{avg}} \tag{8}$$

Where:

- T_{im}: time imbalance.
- T_{max}: maximum execution time of Cloudlets.
- T_{min}: minimum execution time of Cloudlets.
- T_{avg}: average execution time of Cloudlets.

4) Processing Cost: The processing cost of the Cloudlets is defined in terms of the assigned virtual machine Vm_i by the data center on which it executes. Those resources are defined as the following characteristics:

- Cost of Memory: the memory used by Vm_i in its datacenter;
- Cost of Bandwidth: the amount of bandwidth consumed;
- Cost of MIPS: the cost of computation (MIPS) to execute.

D. Experimental Results

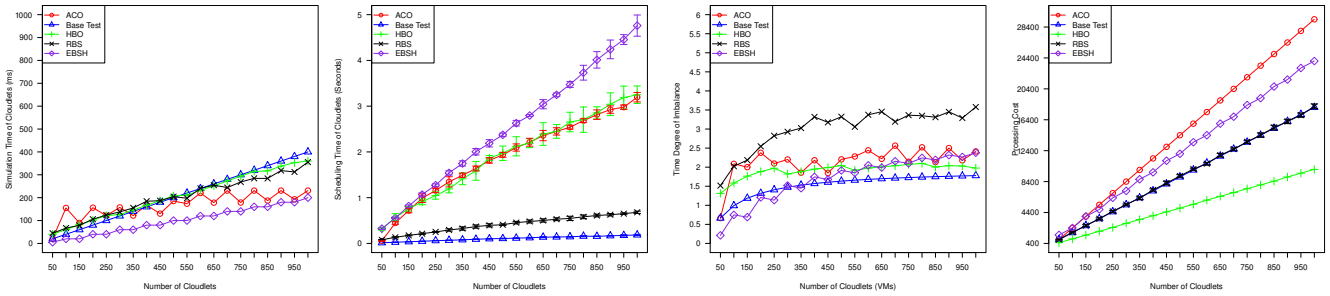
The results obtained in the experiments are presented and discussed separately in two sections: heterogeneous scenario and EBSH variations. In the heterogeneous scenario, the number of virtual machines is 50 virtual machines and up to 1000 Cloudlets. The scenario was used to evaluated the algorithms according to the same metrics described previously. Table IV demonstrates the configurations of the EBSH. This configuration is used in this part of the simulations, but other EBSH variations are analyzed in Section VII-E.

In Figure 5a, the obtained execution times are shown for an increasing number of cloudlets and keeping the number of virtual machines the same. In this part of the simulation that tests the execution time of the algorithm, it can be clearly seen that the EBSH has the best performance of all the algorithms. EBSH factors are set up to focus on computation (i.e. finish fast) and fairness is also considered as seen in IV. Thus, EBSH has the fastest execution time. ACO performs better than the other algorithms, namely. The factors of the ACO were chosen to give the best possible performance of the ACO in terms of the execution time of cloudlets. The EBSH scheduler has the edge on ACO in execution time because EBSH does not have a restriction of reusing virtual machines as the ACO does. However, fairness is considered in this set up of EBSH, so each factor has a fair importance in the scheduling decision. However, the current setup of EBSH focuses more on the computation aspect.

Figure 5b shows the time each scheduler takes to finish assigning virtual machines to Cloudlets. The time was recorded in seconds. The Figure describes that the base test presented the shortest scheduling time because of its simplicity and no computation needed to make the scheduling choices. The second best scheduling times is the RBS due to its simple assignment decisions as it only checks for the WIL (walk in length). The third best scheduling is the HBO as it only identifies the least cost of the virtual machines to execute the Cloudlets. ACO and EBSH are taking the longest time to provide a scheduling solution due to the complexity involved in the search process.

As shown in Figure 5c, it measures the time degree imbalance to find the best load distribution of Cloudlets among virtual machines. In other words, it measures the fairness of the execution time between the cloudlets. EBSH in the current setup IV of its factors has the fairness factor considered and its performance matches the Base Test that has the edge in load distribution due to its nature of scheduling as it gives each of the virtual machines an equal number of Cloudlets; hence, the execution times of those Cloudlets are close to each other. However, in the case of the Base Test, it does not mean that the Cloudlets finish fast as proven in Figure 5a. HBO and ACO have similar fairness performance. HBO has the load balancing factor it used to the allocating resources and thus it has a good performance when it comes to fairness. ACO has a restriction factor that prevents ants from choosing a virtual machine twice in a single ACO iteration. This restriction leads to fairness between cloudlets execution.

To calculate the cost of cloudlet processing, we took into account the bandwidth, memory, and MIPS needed. As depicted in Figure 5d, HBO presents the best price value as it does consider the lowest processing cost when scheduling even though the load balancing factor shows an effect on the decision making but it is minimal. The Base Test and RBS have a good cost because they have no restrictions



(a) Execution Time (b) Scheduling Time (c) Degree of Time Imbalance (d) Processing Costs

Figure 5: Performance Analysis of EBSH against ACO, HBO, and RBS

on choosing any virtual machine and they distribute the cloudlets. EBSH in the current setup IV does not consider the vote from the cost factor and hence, it has a high cost. ACO has a high cost because it focuses on using the strongest virtual machines when it schedules cloudlets.

E. EBSH Variations

EBSH has the edge on the existing algorithms because of its ability to change and adapt to different optimization objectives that are decided by the Cloud service provider and subscribers. EBSH is able to change focus by a simple change to the weights of each factor. The following graphs represent EBSH ability to change by shifting focus based on the change of the controlling weights.

The cost of different variations of EBSH is shown in Figure 6a. It shows that the EBSH ($EBSH_{COMP}$) that focus on only computation only had the most expensive solution. The previous setup is suitable for use when the requirement is to finish the tasks as fast as possible such as real-time application tasks. Also, Figure 6a demonstrates that the cost of $EBSH_{LOAD}$ focused and $EBSH_{ALL}$ with all the optimization objectives has a lower cost than the $EBSH_{COMP}$. The best variation in terms of cost that is even better than HBO is $EBSH_{CC}$ and $EBSH_{CL}$ that are both focused on cost as a primary optimization objective and coupled with either computation or load to ensure fairness.

The EBSH variations are also evaluated for their execution time. Figure 6b shows the results of the simulation. $EBSH_{COMP}$ has the best (i.e. least Execution time) which is even better than all of the PBSH and the EBSH variation shown in Figure 5a. The second best EBSH variation is $EBSH_{ALL}$ because the focus is on all the scheduling objectives. The last three variations of EBSH, namely $EBSH_{LOAD}$, $EBSH_{CL}$, $EBSH_{CC}$ have a similar execution time to HBO, RBS, and BaseTest.

All the EBSH variations have the same scheduling time as seen in Figure 6c. For all the variations of EBSH, the algorithm has to evaluate all the deciding factors presented in IV to calculate the votes. Therefore, the variation of EBSH takes the same time to make a scheduling decision.

Moreover, EBSH variations are measures under time degree of imbalance 8. $EBSH_{COMP}$ has performs the best out of all the other variations because of the focus on finding the strongest virtual machines to perform the tasks which ensure that tasks have a similar execution time. $EBSH_{CL}$ $EBSH_{CC}$ have a good time degree of imbalance as seen in 6d that is close to the Base Test in Figure 5c because these two variations ensure that all the Cloudlets with the least price get their fair share of tasks. $EBSH_{LOAD}$ has a slightly high time degree of imbalance because it only ensures that all virtual machines get tasks, which is also applicable for $EBSH_{ALL}$.

VIII. Conclusion

In this paper, we conducted an extensive set of experimental analyses on an adaptive algorithm named EBSH. The performance of the EBSH is measured against the bio-inspired scheduling algorithms, HBO and Ant Colony, as well as Random Biased Sampling algorithm. In one experimental scenario, the schedulers were tested in heterogeneous setups with different tasks sizes. In this test the EBSH, focusing on only computation and load, showed the best simulation time and fairness. and HBO had the minimal processing cost. The other scenario evaluated variations of EBSH and demonstrated EBSH ability to adapt to different optimization objectives and the ease in adjusting its controlling factors.

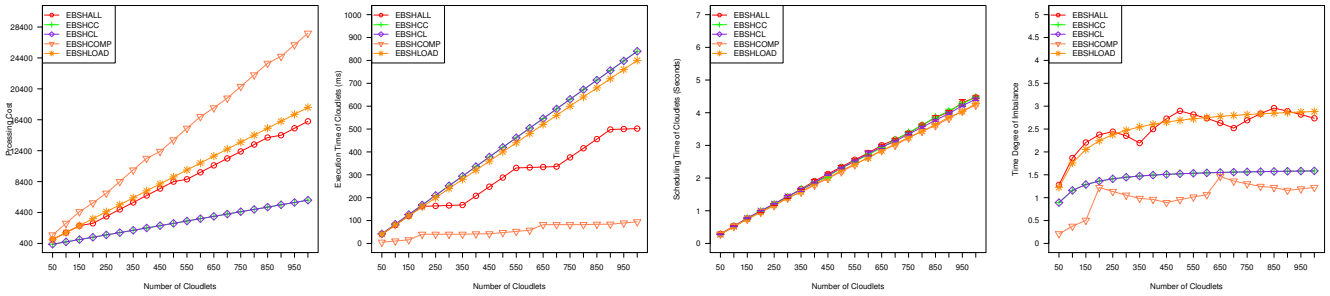
As future work, we intend to explore the drawbacks that have been shown in the results. Based on such issues, we will propose the use of machine learning techniques to automate the adjustment of the EBSH controlling factors which will further enhance EBSH.

References

[1] S. Bilgaiyan, S. Sagnika, and M. Das. An analysis of task scheduling in cloud computing using evolutionary and swarm-based algorithms. *Int. Journal of Computer Applications*, 89(2), 2014.

[2] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. De Rose, and R. Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[3] W.-N. Chen and J. Zhang. A set-based discrete pso for cloud workflow scheduling with user-defined qos constraints. In *Proc. of the IEEE Int. Conference on Systems, Man, and Cybernetics*, pages 773–778, 2012.



(a) Cost (b) Execution Time (c) Scheduling Time (d) Time Degree Imbalance

Figure 6: Performance Analysis of EBSH Variants.

[4] M. Dorigoa and C. Blumb. Ant colony optimization theory: A survey. *Theoretical Computer Science*, 344:243–278, 2005.

[5] Y. Ge and G. Wei. Ga-based task scheduler for the cloud computing systems. In *Proc. of the Int. Conference on Web Information Systems and Mining*, pages 181–186, 2010.

[6] M. Guzek, P. Bouvry, and E.-G. Talbi. A survey of evolutionary computation for resource management of processing in cloud computing [review article]. *Computational Intelligence Magazine, IEEE*, 10(2):53–67, 2015.

[7] J. Hu, J. Gu, G. Sun, and T. Zhao. A scheduling strategy on load balancing of virtual machine resources in cloud computing environment. In *Proc. of the Int. Symposium on Parallel Architectures, Algorithms and Programming*, pages 89–96, 2010.

[8] S. H. Jang, T. Y. Kim, J. K. Kim, and J. S. Lee. The study of genetic algorithm-based task scheduling for cloud computing. *Int. Journal of Control and Automation*, 5(4):157–162, 2012.

[9] H.-H. Li, Z.-G. Chen, Z.-H. Zhan, K.-J. Du, and J. Zhang. Renumber coevolutionary multiswarm particle swarm optimization for multi-objective workflow scheduling on cloud computing environment. In *Proc. of the Companion Publication on Genetic and Evolutionary Computation Conference*, pages 1419–1420, 2015.

[10] H.-H. Li, Y.-W. Fu, Z.-H. Zhan, and J.-J. Li. Renumber strategy enhanced particle swarm optimization for cloud computing resource scheduling. In *Proc. of the IEEE Congress on Evolutionary Computation*, pages 870–876, 2015.

[11] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang. Cloud task scheduling based on load balancing ant colony optimization. In *Proc. of the Annual Chinagrid Conf.*, pages 3–9, 2011.

[12] J. Liu, X.-G. Luo, X.-M. Zhang, F. Zhang, and B.-N. Li. Job scheduling model for cloud computing based on multi-objective genetic algorithm. *Int. Journal of Computer Science Issues*, 10(3):134–139, 2013.

[13] P. Mell and T. Grance. The nist definition of cloud computing. Technical Report Special Publication 800-145, National Institute of Standards and Technology, U.S. Department of Commerce, 2011.

[14] S. Nakrani and C. Tovey. On honey bees and dynamic server allocation in internet hosting centers. *Adaptive Behavior*, 12(3-4):223–240, 2004.

[15] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Proc. of the 24th IEEE Int. Conference on dvanced Information Networking and Applications*, pages 400–407, 2010.

[16] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Proc. of the 24th IEEE Int. Conference on Advanced Information Networking and Applications*, pages 400–407, 2010.

[17] F. Pop, V. Cristea, N. Bessis, and S. Sotiriadis. Reputation guided genetic scheduling algorithm for independent tasks in inter-clouds environments. In *Proc. of the 27th Int. Conference on Advanced Information Networking and Applications Workshops*, pages 772–776, 2013.

[18] O. A. Rahmeh, P. Johnson, and A. Taleb-Bendiab. A dynamic biased random sampling scheme for scalable and reliable grid networks. *INFOCOMP Journal of Computer Science*, 7(4):1–10, 2008.

[19] M. Randles, D. Lamb, and A. Taleb-Bendiab. A comparative study into distributed load balancing algorithms for cloud computing. In *Proc. of the IEEE 24th Int. Conference on Advanced Information Networking and Applications Workshops*, pages 551–556, 2010.

[20] V. Roberge, M. Tarbouchi, and G. Labonté. Comparison of parallel genetic algorithm and particle swarm optimization for real-time uav path planning. *IEEE Transactions on Industrial Informatics*, 9(1):132–141, 2013.

[21] M. A. Rodriguez and R. Buyya. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds. *IEEE Transactions on Cloud Computing*, 2(2):222–235, 2014.

[22] T. D. Seeley, S. Camazine, and J. Sneyd. Collective decision-making in honey bees: how colonies choose among nectar sources. *Behavioral Ecology and Sociobiology*, 28(4):277–290, 1991.

[23] M. Shen, Z.-H. Zhan, W.-N. Chen, Y.-J. Gong, J. Zhang, and Y. Li. Bi-velocity discrete particle swarm optimization and its application to multicast routing problem in communication networks. *IEEE Transactions on Industrial Electronics*, 61(12):7141–7151, 2014.

[24] X. Wang and Y. Wang. An energy and data locality aware bi-level multiobjective task scheduling model based on mapreduce for cloud computing. In *Proc. of the IEEE/WIC/ACM Int. Conferences on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 648–655, 2012.

[25] Z. Wu, Z. Ni, L. Gu, and X. Liu. A revised discrete particle swarm optimization for cloud workflow scheduling. In *Proc. of the Int. Conference on Computational Intelligence and Security*, pages 184–188, 2010.

[26] Z.-H. Zhan, X.-F. Liu, Y.-J. Gong, J. Zhang, H. S.-H. Chung, and Y. Li. Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Comput. Surv.*, 47(4):63:1–63:33, July 2015.

[27] C. Zhao, S. Zhang, Q. Liu, J. Xie, and J. Hu. Independent tasks scheduling based on genetic algorithm in cloud computing. In *Proc. of the Int. Conf. on Wireless Communications, Networking and Mobile Computing*, pages 1–4, 2009.