Future Generation Computer Systems 113 (2020) 106–112

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Energy-aware workflow scheduling and optimization in clouds using bat algorithm

Yi Gu *, Chandu Budati

Department of Computer Science, Middle Tennessee State University, United States of America

ARTICLE INFO

Article history:
Received 2 July 2019
Received in revised form 24 May 2020
Accepted 20 June 2020
Available online 26 June 2020

Keywords:
Workflow scheduling
Energy efficiency
Throughput
Latency
Clouds

ABSTRACT

With the ever-increasing deployment of data centers and computer networks around the world, cloud computing has emerged as one of the most important paradigms for large-scale data-intensive applications. However, these cloud environments face many challenges including energy consumption, execution time, heat and CO_2 emission, as well as operational cost. Due to the extremely large scale of these applications and a huge amount of resource consumption, even a small portion of the improvements in any of the above fields can yield huge ecological and financial rewards. Efficient and effective workflow scheduling in cloud environments is one of the most significant ways to confront the above problems and achieve optimal resource utilization. We propose an Energy Aware, Time, and Throughput Optimization heuristic (EATTO) based on the bat algorithm. Our goal is to minimize energy consumption and execution time of computation-intensive workflows while maximizing throughput, without imposing any significant loss on the Quality of Service (QoS) guarantee.

1. Introduction

Next generation e-science is producing colossal amounts of data, now frequently termed as "big data", on the order of terabyte at present, and petabyte or even exabyte and zettabyte in the predictable future. These scientific applications typically feature data- and network-intensive workflows comprised of computing tasks with intricate inter-task dependencies. The overall performance of executing such large-scale scientific workflows in network environments depends on how those tasks are assigned and executed on the selected computer nodes or virtual machines (VMs) based on different scheduling schemes.

It is estimated by Gartner that the worldwide public cloud services market will grow 18% in 2017 to $246.8 billion, up from $209.2 billion in 2016. Typically, a cloud environment is a collection of data centers that are installed globally with thousands of computers or servers using VMs and network storage technologies. Each cloud environment requires massive cooling systems to maintain the required operational temperatures. Servers and cooling systems combined use up to 80% of all the electricity within a data center [1]. Thus, an entire cloud with more than one data center consumes a huge amount of energy, typically in the range of 10 to 20 Megawatts during normal operation [2]. More recently, the estimated global data center electricity consumption in 2018 was 198 TWh, or almost 1% of total global electricity consumption [3,4]. The average cost of operation of servers has increased by a factor of four in just a decade [5] and following the current trend, it is expected to double in the next five years. Apart from the operational cost, the global information and communication technology (ICT) industry accounts for approximately 2% of the global carbon dioxide (CO_2) emissions due to PCs, servers, supercomputers, local area networks (LANs), office telecommunications and printers, cooling solutions, fixed and mobile telephony [6]. Therefore, optimizing energy consumption and reducing carbon footprint of data centers have been considered an increasingly important challenge. Any optimization in operational cost, environment footprint, and system reliability will indisputably lead to not only economic but also environmental benefits.

A lot of work has already been done to optimize the execution of large-scale scientific workflow applications in the recent years. Some applications have a demand for high reliability, large throughput, and low completion time; while others run under a certain monetary budget or energy consumption constraint, for which the highest possible level of QoS may not always be desirable. In this paper, we tackle the problem of developing a scheduling algorithm in a cloud environment to optimize energy consumption, execution time, and throughput of computation-intensive workflows, more specifically, to minimize energy consumption and execution time while maximizing throughput using a meta-heuristic genetic algorithm, named Energy Aware, Time, and Throughput Optimization (EATTO), based on the Bat algorithm [7]. The key contributions of this work lie in: (i) modeling

* Corresponding author.
E-mail addresses: Yi.Gu@mtsu.edu (Y. Gu), cb7c@mtmail.mtsu.edu (C. Budati).

https://doi.org/10.1016/j.future.2020.06.031
0167-739X/

and scheduling of a complex DAG-structured scientific workflow in the cloud; (ii) proposing a task allocation scheme for a multi-objective optimization problem (MOP) which takes latency, throughput, and energy into consideration, and finds the best possible compromise for all three of them; (iii) conducting an extensive set of performance evaluations and comparisons to illustrate effectiveness and superiority of the proposed algorithm.

2. Related work

The operation of a large-scale cloud environment requires a considerable amount of energy that accounts for a substantial portion of the total cost to run any application. Recently, many research efforts have been devoted to addressing multi-objective workflow optimization problems, such as execution time, energy consumption, reliability, operational cost, and bandwidth usage, etc., for either grid networks or virtualized servers. It has already been well understood that these problems are NP-hard when we consider networks with more than one task to be mapped on to a node. In [8], Wu et al. conducted a taxonomy and a review of different workflow scheduling algorithms. They highlighted some research directions for future investigation by surveying various scheduling algorithms in a problem–solution manner, including DTL [9], DBL [10], and DET [11], for implementing the deadline distribution strategy.

Techniques such as variable voltage Earliest-Deadline-First scheduling [12] and Dynamic Power Management [13] were developed and studied for achieving optimal energy consumption. Wang et al. proposed a look ahead genetic algorithm [14], which uses the reputation to evaluate the resources' reliability. Tan et al. proposed a Trust services-oriented Multi-Objectives Workflow Scheduling (TMOWS) model [15] for cloud services. Two algorithms were proposed in [16] based on SPSS, where one is SPSS-ED, focusing on meeting energy and deadline constraints, and the other is SPSS-EB, focusing on meeting energy and budget constraints. These two algorithms address the problem of resource provisioning under energy constraints with budget or deadline requirements. Most of the approaches stated are single-objective optimization problems and do not effectively deal with the trade-off between different priorities. Our goal here is to achieve the best possible compromise among energy consumption, execution time, and throughput optimization, without any significant adverse effect on other quality of service (QoS) parameters.

Many researchers have been working on multi-objective optimization problems recently, in which they aim to optimize more than one aspect of the system while balancing the others. In [17], Durillo et al. presented a new Pareto-based multi-objective workflow scheduling algorithm based on empirical models which capture the behaviors of energy consumption in heterogeneous parallel grids. However, this approach is based on some existing heuristics while we aim to develop a better algorithm which takes energy consumption, execution time, and throughput all into consideration. A particle swarm optimization (PSO) based approach was proposed in [18] which aims to minimize execution time of a computing workflow while balancing task load on all nodes. Like in many PSO approaches, a penalty function was used in their algorithm which marks the particles that violate the constraints. However, this method can lead to a premature convergence at a local minimum and the parameters used for optimization are highly empirical, making it hard to optimize a particular network. Sawant et al. used a genetic algorithm for virtual machine configuration [19], where they converted the constraints into a fitness function. But this approach also has drawbacks very similar to [18]. The ant colony optimization (ACO) approach was proposed for QoS optimization in grids [20], in which they designed seven heuristics based on ACO approach and proposed an adaptive scheme that allows artificial ants to select heuristics based on pheromone values of the ants.

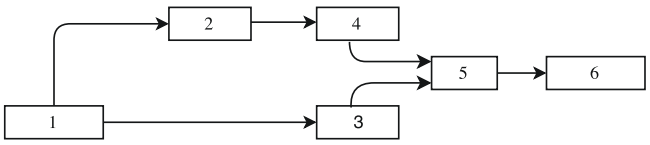


Fig. 1. A workflow application with 6 tasks.

A lot of research effort has been devoted to address MOP problems, such as minimizing completion time given a fixed budget [21] and optimizing a four-objective case study comprising of make span, economic cost, energy consumption, and reliability [22]. However, they convert one or more of the objectives into constrains and optimize remaining objectives, which may not always be practical. We have implemented some of the above ideas while evaluating the performance of the proposed EATTO algorithm in comparison to others.

We focus on computation-intensive applications which have very tight energy and time constraints but not too much on throughput due to high-speed networks. We propose a scheduling scheme that yields the minimum energy consumption, minimum execution time, and maximum throughput without imposing any significant losses on other QoS requirements by employing an evolutionary meta-heuristic to converge to an optimum solution. We hope to optimize three conflicting objectives by finding the best possible compromise, which is desirable, especially in the current cloud environments.

3. Mathematical models and problem formulation

We first describe the characteristics of workflow applications and cloud environments. Then we provide a formal definition of a workflow scheduling problem with three objectives, i.e., energy consumption, execution time and throughput.

3.1. Workflow and cloud models

The workflow application can be viewed as a collection of several tasks/modules with intricate inter-task dependencies, which can be modeled as a directed acyclic graph (DAG) $G(T, E)$, where $T = \{T_1, \ldots, T_n\}$ represents the task/module of the application, and $E \in T \times T$ denotes the dependencies between the tasks. A weight w_i is associated with each task T_i, which indicates the amount of work (the number of CPU cycles) to be done in order to complete the task, i.e., the amount of CPU cycles needed to perform that task. The streaming datasets enter the workflow at the start task and traverse the DAG from one task to another until the final output is collected at the end task.

For an illustration purpose, a workflow example with $n = 6$ tasks is plotted in Fig. 1. If task T_6 is the output/end task, we need to finish processing T_2 on one dataset before proceeding to T_4. The joint to task T_5 corresponds to the merge of two output datasets from tasks T_3 and T_4, respectively. Typically, each task has to wait until it receives the output from each of its predecessors in the workflow before executing its designated task.

The targeted platform is a cloud environment, which can be modeled as a set of m interconnected machines $M = \{M_1, \ldots, M_m\}$. The speed of a machine M_u is represented by frequency fr_u, denoted by CPU frequency GHz, the number of 10^9 cycles per second. Each machine in the network is assumed to have an input and output communication buffer. One machine performs one or more task, depending on the calculated scheduling algorithm.

To simplify the problem, we neglect transfer time as we focus on computation-intensive workflows and fully-connected high-bandwidth heterogeneous networks, where transfer time is much smaller than execution time.

The task execution time and energy cost depend on the calculated scheduling algorithm which assigns each of tasks to one of the machines. We use an allocation function $a(\cdot)$ to represent task assignment. For example, $a(i) = u$ indicates that task T_i is assigned to machine M_u.

3.2. Problem definition

We consider a tri-objective optimization problem of energy consumption, execution time and throughput. The formal problem definition is given as below.

Definition 1. Given a large-scale scientific workflow application consisting of tasks with intricate inter-task dependencies, and a heterogeneous cloud environment, we want to find a mapping/scheduling scheme that assigns each task in the workflow application to a strategically selected machine such that total energy consumption EC, workflow execution time ET, and throughput TP of the workflow application are all optimized.

We assume that all computer machines are running on predetermined configurations. Although configuration changes can be made once a mapping scheme is determined, changing operating frequency or other CPU metrics to achieve better results is not the focus of this work.

3.3. Performance metrics

If S denotes a calculated mapping scheme, energy consumption, execution time, and throughput of the scheme S are defined as follows.

The execution time of the scheme S is denoted by $ET(S)$, which is the execution time of entire workflow using the mapping scheme S, and calculated by adding all task execution times along the critical path (CP). If k is the number of tasks along the critical path of a DAG-structured workflow, the execution time of a computing workflow can be calculated as $ET(S) = \sum_{T_i \in CP} T(i, u)$, where $T(i, u)$ is the execution time of task T_i on machine M_u, and is computed as $T(i, u) = \frac{w_i}{fr_u}$, where w_i is the computational requirements of task T_i.

Throughput $TP(S)$ is the number of data outputs produced at the end task in the workflow per time unit given a mapping scheme S. Throughput is determined by the longest task or frequently named "bottleneck" in the workflow, which is calculated by finding the longest or slowest task in the workflow execution using the following equation $TP(S) = \frac{1}{\max_{i \in (1,\, n)} T(i,u)}$.

The energy consumption $EC(S)$ is the sum of energy consumed on each selected machine determined by the mapping scheme S, which can be computed as $EC(S) = \sum_{M_u \in S} E_u$. E_u is the total energy consumption on machine M_u, which is calculated as $E_u = Estat_u + Edyn_u$. $Estat_u$ is the static portion of energy consumption which depends on the total execution time $\sum_{i|a(i)=u} T(i, u)$ of all tasks that are assigned to machine M_u, and is given by $Estat_u = \epsilon_u \times \sum_{i|a(i)=u} T(i, u)$, where ϵ_u is the static energy consumption per time unit. $Edyn_u$ is the dynamic part of energy consumption and depends on machine speed. The dynamic energy consumption of task T_i executing on machine M_u at CPU frequency fr_u is proportional to ${fr_u}^{\beta_u} \times T(i, u)$ [23,24], where β_u is a constant and $\beta_u > 1$. Therefore, the total dynamic energy consumption of all assigned tasks on machine M_u is $Edyn_u = \sum_{i|a(i)=u} (C_{en} \times fr_u^{\beta_u} \times T(i, u))$, where C_{en} is a constant of proportionality.

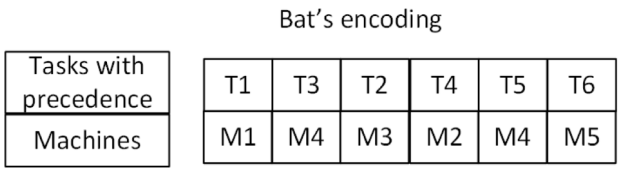


Fig. 2. An example of Bats' encoding.

3.4. Objective function

We formally define our objective function as

$$OF = \frac{ET \times EC}{TP} , \tag{1}$$

where we want to minimize both ET and EC while maximizing TP. It is worth noting that these objectives are inversely related. It may be impossible to obtain the best case scenario for each objective without adversely effecting the other one or two. Thus, we aim to find a possible compromise among all three objectives.

4. Technical solution

Our goal is to find a mapping/scheduling scheme such that the energy consumption and execution time are minimized while the throughput is maximized. We purpose a meta-heuristic genetic algorithm called Energy Aware, Time, and Throughput Optimization (EATTO) using BAT algorithm, and discuss how it can be adapted to find the best task scheduling scheme.

4.1. EATTO algorithm

The Energy Aware, Time, and Throughput Optimization (EATTO) algorithm is inspired by the Bat algorithm, which was initially proposed by Xin-She Yang in 2010 [7]. The Bat algorithm is similar to PSO and based on the echolocation behavior of microbats and uses a frequency tuning method. In our problem, we use an array to prioritize tasks according to their computational requirements and precedence constraints. We consider each sample in the solution space to be a bat i, a possible mapping scheme, taking all task dependencies into consideration. An example of bats' encoding of a workflow with six tasks is shown in Fig. 2, where the first row is an array of ordered tasks and the second row indicates the corresponding assigned machine for each task under this particular mapping scheme. The solution is evaluated at each generation t using the following metrics: a frequency or wavelength fb_i^t, a velocity v_i^t, a loudness A_i^t, a pulse emission rate r_i^t, and a location d_i^t. The loudness A_i^t and pulse emission rates r_i^t can be varied during the iterations. For a simplicity purpose, we use the definitions $A_{t+1}^i = \alpha A_t^i$ and $r_{t+1}^i = r_i^0[1 - exp(-\gamma t)]$, where $0 < \alpha < 1$ and $\gamma > 0$ are two constants. The location of the bat is described as an ordered set of values corresponding to each dimension, i.e., task.

We create a population B of p bats, $B = b_1, b_2, \ldots, b_p$, with random position and velocity. Among those p bats in the present population, the current best solution $b_\star$ is obtained using the objective function defined in Eq. (1). Once we have a current best solution, we generate a new population that is randomly distributed around the current best solution. We then repeat the above steps each time when calculating a new best solution. The mathematical equations for updating frequency fb_i^t, velocity v_i^t and location d_i^t can be defined as

$$fb_i^t = fb_{min}^{t-1} + (fb_{max}^{t-1} - fb_{min}^{t-1}) \times \delta,$$
$$v_i^t = v_i^{t-1} + (d_i^{t-1} - d_\star) \times fb_i^t, \text{ and,}$$
$$d_i^t = d_i^{t-1} + v_i^t,$$

respectively [7], where $\delta \in [0, 1]$ is a random vector drawn from a uniform distribution and $d_\star$ is the best solution so far. fb_{max}^{t-1} and fb_{min}^{t-1} are the maximum and minimum bat frequency for the $(t-1)$th generation, respectively. Similarly, all superscripts in the above equations represent the generation number.

We consider our problem a search in an n-dimensional search space, where n is the number of tasks in a computing workflow. The set of all possible mapping schemes is the search space. Every mapping scheme can be represented as a lattice point in this n-dimensional space, and each point is an ordered tuple of values corresponding to each task. We use the objective function defined in Eq. (1) to calculate the location d by minimizing the function. The pseudocode is presented in Alg. 1, in which the key ideas are: (1) **Initialization**: We generate a random population, initialize the parameters of the algorithm, such as frequency fb, velocity v, loudness A, pulse emission rate r, and location d, evaluate the initial population, and determine the best solution $b_\star$ in the current population. (2) **New Solution Generation**: A new random population is generated around the current best solution $b_\star$. (3) **Local Search**: We evaluate the new population and hopefully improve $b_\star$, which is called random walk. (4) **Tentative Best Solution**: Archive the tentative best solution after a terminations condition is met. (5) **Best Global Solution**: After each generation, variance in the population is reduced to concentrate the search on a specific location, which is referred to as frequency tuning. When the variance reduces to 0, the current best solution converges into the global best solution.

It is worth noting that frequency tuning acts as mutation as it varies the solutions locally. However, if a mutation is large enough (occurs when the random initialization is very far away from the desired result), it leads to a global search. Selection based on $b_\star$ is inspired from the echolocation behavior. The variations of loudness and frequency provide a zooming ability when the vicinity of a possible solution is detected and exploration becomes intensive in that area. This switches a global search to an exploratory local search automatically.

Algorithm 1 Energy Aware, Time and Throughput Optimization (EATTO) $((b_1, ..., b_p), F, G)$;
Input: Bat population $B = (b_1, ..., b_p)$ where p is the population, G is the number of generations and F is the objective function defined in Eq. (1).
Output: Best solution $b_\star$ and its corresponding fitness value $Fit_{min} = F(b_\star)$.

```
Initialize bat population randomly;
Evaluate B, calculate F(d_i) for all i ∈ (1, .., p);
Find the bat b_⋆ | F(d_⋆) is the minimum ;
t = 1;
while t ≤ G do
    i = 1;
    if random(0, 1) ≥ r_i then
        for i ≤ p do
            Update bat population using rules above;
            Calculate Fit_i = F(d_i);
            if Fit_i ≤ Fit_min and random(0, 1) < A_i then
                b_⋆ = b_i;
                Fit_min = Fit_i;
return b_⋆, Fit_min;
```

4.2. Complexity analysis

We compare EATTO algorithm with a random scheduling algorithm *RN* which assigns a task to a node randomly, a binary search heuristic *BSH* for maximum throughput by Benoit et al. [25], and an ant colony optimization based algorithm *ACO* described by Chen et al. [20]. We analyze and compare the complexities of above four algorithms and the optimization performance will be demonstrated in the later section. We evaluate the computational requirements of these algorithms and thus determine if they are actually worth the extra computational overhead to achieve the better performance. It should also be pointed out that all the above mentioned algorithms are general mapping strategies, i.e., there are no restrictions on which tasks can be executed on which node.

We first look at the random algorithm *RN*, which does not guarantee an optimal solution. All it does is to randomly assign each task to a node. Using *RN*, a mapping scheme can be generated in a linear time $O(n)$, where n is the number of tasks in the workflow application.

The *ACO* algorithm generates a population of size p for every generation in a greedy fashion, corresponding to the heuristic chosen by each ant, and finds the best solution for the current generation, which can be done in a linear time $O(p)$. We then repeat this process N_{gen} times and move all the populations towards the best solution each time until the best global solution is achieved. Thus, the complexity of the *ACO* algorithm is $O(p \times N_{gen})$.

The *BSH* scheduler assigns each machine a set of tasks by sorting the tasks in an ascending order. For example, assign a set of tasks T_i $(1 \leq i \leq n)$ for machine M_u. Then $rank_{i,u}$ represents the rank of T_i in the ordered set for M_u. To maximize the throughput, *BSH* heuristic performs a binary search on the period between the best case and the time required to perform sequentially all tasks on the slowest machine, where all tasks are assigned greedily. So the total complexity is $O(m \times n \times log(k))$, where m is the number of nodes in the network, n is the number of tasks in the workflow, and k is the number of all possible solutions assigned using greedy algorithm.

The proposed EATTO algorithm generates a population of size p for every generation in a random fashion, and finds the best solution for the current generation, which can be done in a linear time $O(p)$. We also repeat this process N_{gen} times and try to move all the populations towards the best solution each time until we find the best global solution. Thus, the complexity of the EATTO algorithm is $O(p \times N_{gen})$.

5. Performance evaluation and comparison

For the performance evaluation and comparison purpose, we implement the proposed EATTO algorithm, *ACO* heuristic [20], *BSH* heuristic [25], and random algorithm *RN* in Python on a Windows 7 Desktop PC with Intel Core i7, CPU at 2.66 GHz, and 8.0 GB RAM. We run all four algorithms on the same set of 21 test cases, which covers a wide range of application workflows and network types. The configuration of all nodes and tasks is randomly generated by our program within a predefined range to simulate an extensive set of different workflows ranging from small case with 4 tasks and 6 dependency edges to large one with 110 tasks and 710 dependency edges, and networks ranging from 6 nodes to 150 nodes. All networks are considered fully connected and heterogeneous with high bandwidth. For an illustration purpose, we provide an example of 10 test cases in Table 1. We also define the computational requirement of each task for a computing workflow as well as CPU processing speed for each machine in the cloud. The numbers of generations (N_{gen}) and populations (p) are set to be 1500 and 50, respectively.

We test all heuristics on each test case and obtain the corresponding mapping schemes. We then calculate and plot the values of energy consumption *EC* in Fig. 3, execution time *ET* in Fig. 4, throughput *TP* in Fig. 5, and the objective function *OF* in Fig. 6, respectively, for each algorithm on 21 test cases. We repeat this process four times and calculate the average value for each metric, where *EATTO* and *ACO* algorithms have a

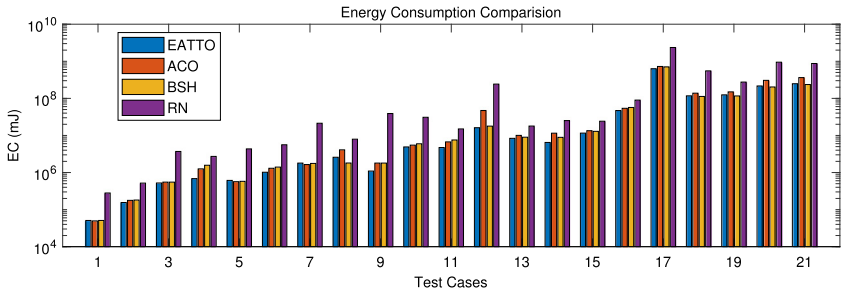


Fig. 3. Energy consumption comparison.

Table 1
An example of 10 workflow test cases.

Case #	1	2	3	4	5	6	7	8	9	10
Node #	47	6	10	15	200	20	250	25	28	31
Module #	4	6	10	13	15	19	22	26	30	35
Edge #	6	10	18	24	30	36	44	50	62	70

Table 2
Constant parameters of EC calculation.

Parameter	Meaning
ϵ_u	Static energy consumption per unit time
β_u	Scaling constant ($\beta_u > 1$)
C_{en}	Constant of proportionality
fr_u	CPU operational frequency of the node

similar low deviation of 2% between runs while *RN* has a much higher deviation of 61% due to its nature. *BSH* is a deterministic algorithm so it always gives the same output with multiple runs for each case. All the parameters that are required to calculate *EC* for a given mapping scheme are tabulated in Table 2, where all these parameters are constant and are defined in the cloud configuration. Since we consider a heterogeneous cloud environment, each machine has a different set of values for the parameters. So the program we developed does not only generate workflow and cloud structures, but also generates the parameters listed in Table 2 for each machine in each test case by specifying the range of the parameters according to real applications.

The horizontal axis in each figure represents the 21 workflow applications considered for the experiments, and the vertical axis gives the corresponding performance measurement obtained by each heuristic in comparison. We have observed that all three performance metrics, i.e., *EC*, *ET*, and *TP*, obtained by the EATTO algorithm have achieved the best performance in most of the test cases in comparison. Particularly, EATTO algorithm yields a much higher *TP* value while producing the minimum *EC* and *ET* values among the four heuristics.

Since our main focus is to find a mapping scheme that produces a good compromise among *EC*, *ET*, and *TP*, we further compare the objective function *OF* defined in Eq. (1) to illustrate this. We have plotted the *OF* measurements of four algorithms in Fig. 6, where we have observed that EATTO algorithm consistently outperforms all three other algorithms, i.e., *ACO*, *BSH* and *RN*, in terms of overall optimization goal.

Using EATTO algorithm, the average increase of the overall performance in Fig. 6 is around 20% when compared to that of *ACO*, and 13% when compared to *BSH*, respectively. The performance improvement between EATTO and *ACO* can be further viewed from each individual objective in Figs. 3, 4 and 5, where the average energy consumption, execution time, and throughput are around 36%, 31%, and 3 times better than that of the *ACO* algorithm, respectively. We have observed the similar trends while comparing EATTO and *BSH*, where the average decrease in energy consumption and execution time are respectively 10% and 15%, while the throughput increase is around 6 times.

We have also noticed that with the increase of the problem sizes, the overall optimization performance of the EATTO algorithm gets better, which is due to the increase of the problem search space, i.e., a larger number of possible solutions are available. This can be viewed by the first test case where the overall performance values of EATTO and *ACO* are very close, and the last test case with a much bigger workflow application of 110 modules, the overall performance value of EATTO is only 3% of the ACO's, which is roughly 33 times better. We can see that this pattern of asymptotical decrease persists when the scale of workflow scheduling problems increases. Note that all vertical axes are measured in a log scale.

For a better performance comparison among all four algorithms, we further adapt the scoring system introduced in [20] to generate a normalized score for each algorithm in comparison. The formulae used to calculate the score are defined as below. $Score_S = ScoreEC_S + ScoreET_S + ScoreTP_S$, where $Score_S$ is the final normalized score for a given scheduling scheme, and $ScoreEC_S$, $ScoreET_S$ and $ScoreTP_S$ are the normalized scores for energy consumption, execution time, and throughput, respectively. These values can be calculated as $ScoreEC_S = \frac{(EC_{max}-EC_S+1)}{(EC_{max}-EC_{min}+1)}$, where EC_{max} and $ECmin$ are minimum and maximum energy consumption values of all possible mapping schemes for a given application, and EC_S is the energy consumption produced by the selected mapping scheme. Similarly, $ScoreET_S$ and $ScoreTP_S$ are computed as $ScoreET_S = \frac{(ET_{max}-ET_S+1)}{(ET_{max}-ET_{min}+1)}$, and $ScoreTP_S = \frac{(TP_S-TP_{min}+1)}{(TP_{max}-TP_{min}+1)}$, where ET_{max} and ET_{min} are minimum and maximum execution times of all possible mapping schemes for a given application, and ET_S is the execution time produced by the selected mapping scheme. TP_{max} and $TPmin$ are minimum and maximum throughput and TP_S is the throughput of the selected mapping scheme.

The values of the scores discussed above for each objective are plotted in Fig. 7 to provide a better performance comparison for each algorithm, where a higher value indicates a better overall performance. We have seen that *EATTO* algorithm consistently achieves the highest values in all 21 test cases. These experimental results have clearly demonstrated that the proposed EATTO algorithm generates the best mapping scheme for the overall performance in comparison to other similar heuristics in the literature, and yields a significantly lower energy cost and execution time generally.

The overall performance of EATTO is roughly five times better than competing approaches when the network has a decent number of nodes. We also notice that this method favors minimizing throughput over other two metrics, which can be observed in later half of test cases. This is probably due to the fact that *TP* values have a larger range and minimizing that would result in overall better compromise. Therefore, we conclude that the proposed scheduling algorithm outperforms other similar methods by a performance improvement factor of at least five.

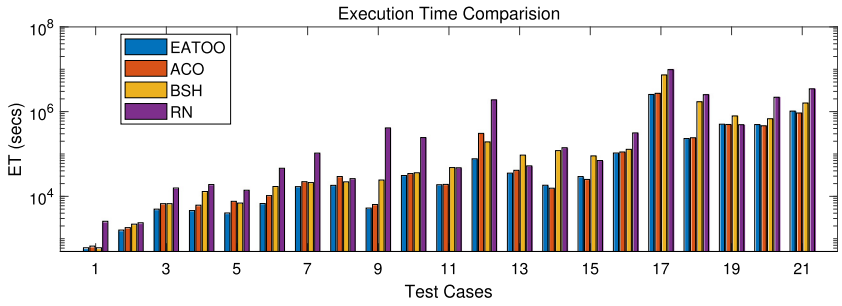


Fig. 4. Execution time comparison.

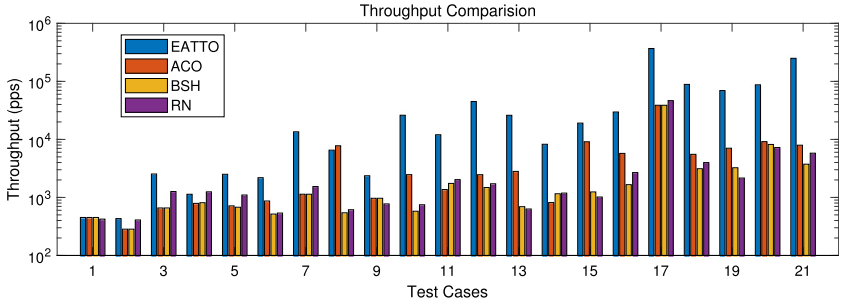


Fig. 5. Throughput comparison.

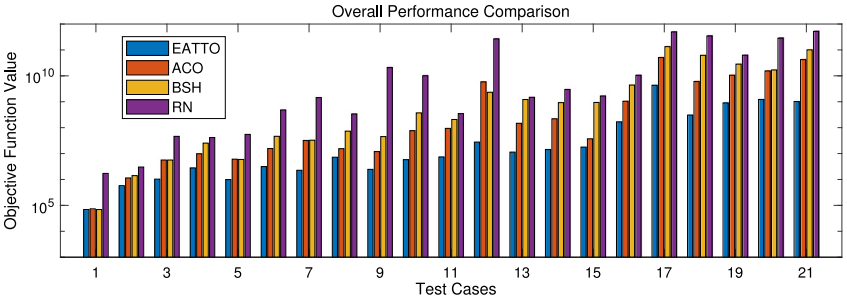


Fig. 6. Overall performance comparison.

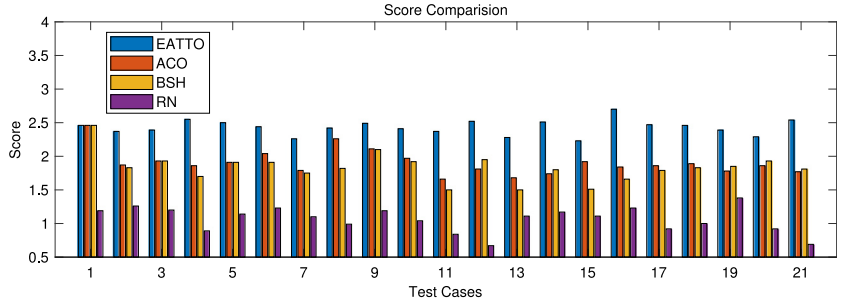


Fig. 7. Performance score comparison.

6. Conclusion and future work

We proposed a meta-heuristic workflow scheduling algorithm, namely Energy Aware, Time, and Throughput Optimization heuristic algorithm (EATTO), for cloud network environments to address the multi-objective optimization problem of energy consumption, execution time, and throughput. We considered our optimization problem as a search problem in an n-dimensional search space, where n is the number of tasks. We formulated a multi-objective function to optimize the overall performance and identified the best global solution by comparing all local optimal ones. We illustrated the superiority of the proposed EATTO algorithm over other existing mapping solutions by conducting an extensive set of performance comparisons. The results showed that EATTO not only consistently generates the best global solution, in terms of overall performance of three objectives, but also produces the best performance for each single objective in most cases.

It is of our future interest to investigate other mapping rules and restrictions on different platforms. We will apply Pareto principle while defining objective functions and consider communication costs, making it even better suited and applicable

to real-life scenarios. Performance evaluation on some real-life workflow applications and cloud environments would be also explored.

CRediT authorship contribution statement

Yi Gu: Conceptualization, Methodology, Validation, Formal analysis, Resources, Data curation, Writing - review & editing, Visualization, Supervision, Project administration, Funding acquisition. **Chandu Budati:** Software, Validation, Formal analysis, Investigation, Data curation, Writing - original draft, Visualization.

Declaration of competing interest

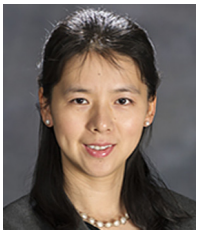
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is sponsored by U.S. National Science Foundation under Grant No. 1525537 with Middle Tennessee State University (MTSU) and by MTSU Faculty Research and Creative Activity Awards under Award No. 18-18-256.

References

[1] F. Cao, M. Zhu, Energy-aware workflow job scheduling for green clouds, in: Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (IThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, IEEE, 2013, pp. 232–239.
[2] S. Garg, C. Yeo, A. Anandasivam, R. Buyya, Energy-efficient scheduling of hpc applications in cloud computing environments, arXiv preprint arXiv: 0909.1146.
[3] Tracking Buildings, Tech. Rep., IEA, Paris, 2019, https://www.iea.org/reports/tracking-buildings.
[4] R. Huang, E. Masanet, Data center it efficiency measures, the uniform methods project: Methods for determining energy efficiency savings for specific measures, 2015, https://www.nrel.gov/docs/fy17osti/68576.pdf.
[5] R. Brown, R. Brown, E. Masanet, B. Nordman, B. Tschudi, A. Shehabi, J. Stanley, J. Koomey, D. Sartor, P. Chan, et al., Report To Congress on Server and Data Center Energy Efficiency: Public Law 109-431, Tech. rep., Ernest Orlando Lawrence Berkeley National Laboratory, Berkeley, CA (US), 2007.
[6] Gartner says data centres account for 23 per cent of global ict co2 emissions, 2007.
[7] X. Yang, A new metaheuristic bat-inspired algorithm, in: Nature Inspired Cooperative Strategies for Optimization, NICSO 2010, 2010, pp. 65–74.
[8] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, J. Supercomput. 71 (9) (2015) 3373–3418.
[9] J. Yu, R. Buyya, C. Tham, Cost-based scheduling of scientific workflow applications on utility grids, in: E-Science and Grid Computing, 2005. First International Conference on, IEEE, 2005, p. 8.
[10] Y. Yuan, X. Li, Q. Wang, Y. Zhang, Bottom level based heuristic for workflow scheduling in grids, Chin. J. Comput.-Chin. Ed. 31 (2) (2008) 282.
[11] Y. Yuan, X. Li, Q. Wang, X. Zhu, Deadline division-based heuristic for cost optimization in workflow scheduling, Inform. Sci. 179 (15) (2009) 2562–2575.
[12] H. Aydin, Q. Yang, Energy-aware partitioning for multiprocessor real-time systems, in: Proc. of the IEEE Int. Parallel & Distributed Processing Symposium, 2003.
[13] T. Horvath, T. Abdelzaher, K. Skadron, X. Liu, Dynamic voltage scaling in multitier web servers with end-to-end delay control, IEEE Trans. Comput. 56 (4) (2007) 444–458.
[14] L. Liu, M. Zhang, R. Buyya, Q. Fan, Deadline-constrained coevolutionary genetic algorithm for scientific workflow scheduling in cloud computing, Concurrency and Computation: Practice and Experience 29 (5) (2017).
[15] W. Tan, Y. Sun, G. Lu, A. Tang, L. Cui, Trust services-oriented multi-objects workflow scheduling model for cloud computing, in: Joint International Conference on Pervasive Computing and the Networked World, 2012, pp. 617–630.
[16] R. Tolosana-Calasanz, J. Bañares, C. Pham, O. Rana, Enforcing qos in scientific workflow systems enacted over cloud infrastructures, J. Comput. System Sci. 78 (5) (2012) 1300–1315.
[17] J. Durillo, V. Nae, R. Prodan, Multi-objective energy-efficient workflow scheduling using list-based heuristics, Future Gener. Comput. Syst. 36 (2014) 221–236.
[18] S. Pandey, L. Wu, S. Guru, 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, AINA, 2010, pp. 400–407.
[19] S. Sawant, A genetic algorithm scheduling approach for virtual machine resources in a cloud computing environment.
[20] W. Chen, J. Zhang, An ant colony optimization approach to a grid workflow scheduling problem with various qos requirements, IEEE Trans. Syst. Man Cybern. C 39 (1) (2009) 29–43.
[21] H. Kllapi, E. Sitaridi, M. Tsangaris, Y. Ioannidis, Schedule optimization for data processing flows on the cloud, in: Proc of the 2011 ACM SIGMOD International Conference on Management of data, 2011, pp. 289–300.
[22] H. Fard, R. Prodan, J. Barrionuevo, T. Fahringer, A multi-objective approach for workflow scheduling in heterogeneous computing environments, in: Proc. of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, 2012, pp. 300–309.
[23] N. Bansal, T. Kimbrel, K. Pruhs, Speed scaling to manage energy and temperature, J. ACM 54 (1) (2007) 3.
[24] A. Othman, J. Nicod, L. Philippe, V. Rehn-Sonigo, Optimal energy consumption and throughput for workflow applications on distributed architectures, Sustain. Comput.: Informatics and Systems 4 (1) (2014) 44–51.
[25] A. Benoit, A. Dobrila, J. Nicod, L. Philippe, Mapping workflow applications with types on heterogeneous specialized platforms, J. Parallel Comput. 37 (8) (2011) 410–427.



Dr. Yi Gu is an associate professor in the Department of Computer Science at Middle Tennessee State University (MTSU). She received her M.S. and Ph.D. degrees in Computer Science from University of Memphis in 2008 and 2011, respectively, and the B.S. degree in Computer Science from Jiangsu University, P.R. China in 2005. She worked as an assistant professor of Computer Science at University of Tennessee Martin from 2011 to 2013, and then joined MTSU in 2013. Dr. Gu received her tenure in 2018. Her research interests include parallel and distributed computing, workflow scheduling and optimization, Cloud/Green computing, wireless sensor networks, and cyber security.

Mr. Chandu Budati has been working as a Senior software engineer at Visa, in Austin, Texas, since January 2020. Prior to that, he had worked as a software engineer at Cerner Corporation since he graduated from Middle Tennessee State University as a master student in fall 2018. This research was partially done during that time under Dr. Yi Gu's direct supervision.