Virtual Machine Placement Optimization for Big Data Applications in Cloud Computing

SEYYED MOHSEN SEYYEDSALEHI AND MOHAMMAD KHANSARI
Faculty of New Sciences and Technologies, University of Tehran, Tehran 1417955840, Iran
Corresponding author: Mohammad Khansari (m.khansari@ut.ac.ir)

ABSTRACT  Big data and cloud computing are two advanced technologies that have overcome many computing and analytical challenges in recent years. With the rise in the applications of these technologies, the necessity of efficiency and optimization in the utilization of related resources has made sense. The procedure of locating virtual machines (VM) in physical machines (PM) affects the performance, speed, and costs of cloud computing services. VM placement in cloud computing is an NP-hard problem. Indeed, the problem is more complicated in big data tasks due to the need for transferring high volumes of traffic between VMs. This paper proposes a new approach for VM placement in a multi data center (DC) cloud environment. The aware genetic algorithm first fit (AGAFF) is a context-aware algorithm that distinguishes big data tasks with an input tag and uses a structure to minimize the traffic between MapReduce nodes. This multi-objective algorithm is based on the genetic algorithm, which is incorporated with the first fit methodology. The algorithm minimizes energy usage by minimizing the number of used servers, intra-DC traffic of big data tasks, and VMs’ live migration while maximizing relevant usage of CPU and RAM in every server. Furthermore, it improves job execution time, especially in big data processing, and reduces service level agreement (SLA) violations. A comparison between the results of AGAFF and four other algorithms shows by about 61% energy consumption reduction on average on different scales and approves a decrease in the number of needed PMs, intra-DC traffic of big data processing, and the number of live migrations.

INDEX TERMS  Big data, cloud computing, genetic algorithm, green computing, multi-objective VM placement.

I. INTRODUCTION
Cloud computing is one of the most significant shifts in modern ICT services [1] which is a key enabler for a number of recent developments, such as the big data analytics, Internet of things (IoT), edge computing, and smart cities [2]. Cloud computing models are majorly classified into IaaS, PaaS and SaaS [3] and it has changed computing from a purchasable product to a deliverable service over the Internet [4]. Low cost, flexibility, and scalability are the main benefits of using cloud computing. These advantages have been made feasible through virtualization technologies. Virtualization empowers cloud computing to manage resources in the multiuser environment [5] and provides computing resources like CPU, RAM, network, and platforms [6].

Big data is one of the technologies that have benefited from cloud computing capabilities. Many resource constraints in big data processing have now been removed through the cloud. In addition, analysts have not only more data to work with but also the processing power to handle large numbers of records with many attributes. This allows for increasing the predictability and knowledge extraction from big data [7].

Although processing big data based on the cloud has numerous advantages, service delivery is not an easy task for service providers. Based on our experience in launching and developing the XaaS Public Cloud,1 combining the two complicated technologies needs more effective resource management systems. For example, in MapReduce, an established framework for processing large-scale data-intensive applications [8], big data is partitioned and stored over several

1 XaaS Public Cloud was one of the public cloud operators in Iran.
data nodes in a cloud system [9], and the model undertakes efficient parallel computing through a large number of data nodes and computation nodes for data-intensive cloud application [10]. Hence, improper virtual machine (VM) placement can cause unbalanced resource utilization [11], transmission latency [5], [9], [10], increasing bandwidth usage [9], and energy consumption [12], [13]. All of these challenges increase service costs and violate service level agreements (SLA) from a business perspective. In this regard, any small optimization can save electricity and alleviate carbon dioxide emissions, providing green computing.

Many researchers have investigated different aspects of VM placement in general cases, with just a few cases considering big data requirements. However, non-operational and limitative assumptions in many cases make some solutions far from practical applications. For example, in [14], the algorithm can process just one request at each moment; [15] does not consider the overhead cost of live migration; in [5], some constraints are settled for the number of VMs for every requested job; and in [16], a fat-tree topology and a homogeneous data center are regarded.

It is very important for service providers that the proposed solutions improve the performance of the whole system such that other negative effects become minimal. For example, considering CPU without RAM limitations, not being aware of the type of the requested processes, or not paying attention to live migration overhead disturb all the placement processes in practical environments.

To tackle these issues, we propose an aware genetic algorithm first fit (AGAFF) for VM placement in big data tasks. AAGAFF is based on evolutionary algorithms and, more specifically, the genetic algorithm (GA). In the placement problem, we want to choose among different placement candidates. Every choice can be feasible or infeasible. The feasibility is determined by the limitations and constraints of devices. For each feasible solution, a cost is calculated. AAGAFF selects the lower cost.

GA characteristics match the placement problem. It is also proven to be a capable tool for NP-hard problems in computer science [17]. In addition, we incorporated the first fit methodology with GA to improve the AAGAFF efficiency. The AAGAFF targets significant service provisioning concerns; it minimizes the cloud energy consumption of a multi-DC, maximizes utilization of resources, and reduces scheduling time. The main contributions of our algorithm are as follows:

- Delivering an optimized solution considering all important aspects of requests, i.e., CPU, RAM, and intra-DC network traffic. (Moreover, we consider the context of the algorithm; herein, we pay specific attention to big data tasks and placement of high cohesion VMs.)
- Reducing big data traffic and placing interrelated MapReduce VMs in a structure with the minimum cost.
- Presenting a multi-objective solution that manages waste of resources and energy consumption by placing newly arrived VMs and migrating present ones in case of necessity.
- Defining a scalable solution to heterogeneous multi-DC cloud systems and using modern data center architectures.
- Proposing a comprehensive algorithm regarding real-world service provisioning limitations, circumstances, and priorities.

The rest of this article is organized as follows. Section two reviews the basic concepts. In section three, previous related work is investigated. Sections four and five are dedicated to problem formulation and AAGAFF algorithm, respectively. Eventually, section six discusses the results and analysis of AAGAFF.

II. BACKGROUND

Given that the perception of the problem and its solution requires knowledge of cloud computing and big data, this section gives a brief explanation of both technologies.

A. CLOUD COMPUTING

In cloud computing, the management of resource provisioning is centralized, and resources are allocated to requests on demand. This allocation is agile, flexible, and elastic. According to the definition given by the National Institute of Standards and Technology (NIST) of the United States [18], cloud computing is a model for enabling network access to a shared pool of configurable resources (e.g., networks, servers, storage, applications, and services). Access to resources is ubiquitous and convenient [18], [19]. On-demand, self-service, broad network access, rapid elasticity, and measured services are essential characteristics of cloud computing [18].

Cloud computing platforms are generally provided in three layers, including infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS). Moreover, four common deployment models are private cloud, public cloud, hybrid cloud, and community cloud.

B. BIG DATA

Big data is an evolving term that describes any huge amount of structured, semi-structured, or unstructured data that has the potential to be mined for useful information [7]. It refers to large datasets which require non-traditional scalable solutions for data acquisition, storage, management, analysis, and visualization [20]. Big data can be characterized by a set of characteristics such as volume, velocity, variety, variability, veracity, value, and validity.

The data are generated from multiple sources such as the Internet of things (IoT), social media applications, medical records, emails, documents, websites, science data, sensors, smart phones, and other resources [21]. This huge amount of data is so large that is estimated to account for 30 percent of data stored in data centers in 2021, up from 18 percent in 2016 [22]. Big data can use analytics such as artificial intelligence (AI), machine learning (ML) and deep learning (DL) to produce better analysis [21].
C. HADOOP FRAMEWORK

Hadoop is an open-source Java library that supports data-intensive distributed applications by implementing the MapReduce framework. It has two main subprojects, including Hadoop distributed file system (HDFS) and MapReduce programming paradigm [23]. The other subprojects, such as YARN, Common, Hbase, Hive, Ozone, and Zookeeper, provide complementary services [24]. Hadoop is suited to high-throughput and in-depth analysis where a larger portion or all of the data is harnessed [25].

The term “MapReduce” is based on two distinct tasks that are performed by a Hadoop program. The input data may be divided and then passed to the mapper functions. All the divisions are processed simultaneously [26] and are parsed into (key, value) pair records. Map function replicates these records and maps each of them to a set of intermediate (key, value) pairs [24]. At last, reducers combine them to get a consolidated output. Figure 1 displays the architecture of MapReduce.

![MapReduce architecture](image)

**FIGURE 1.** MapReduce architecture [27].

III. RELATED WORK

VM placement or consolidation is a challenging issue in cloud computing that has been extensively investigated in recent years [5], [8], [9], [10], [14], [15], [16], [28], [29], [30], [31], [33]. So far, different analyses have been performed, and various VM placement solutions have been proposed [28]. Since the placement problem is proven to be NP-hard in the general case [14], [15], [28], [29], [34], [35], alternative algorithms using different techniques are presented to solve it. Ant colony optimization (ACO) [14], [36], genetic algorithm (GA) [15], [16], [30], greedy algorithm [4], [28], [33], game theory [32], and biogeography-based optimization (BBO) [37] are some of the heuristic or meta-heuristic methods. Furthermore, some other models like graph [5], [8], [29], [33], approximation algorithms [38], and linear programming [9], [10], [33] have been widely used to solve the problem.

In the following, we categorize the available algorithms concerning their functions into two parts. In the energy-aware category, the main objective is to reduce cloud energy usage. On the other hand, the second category, SLA-aware algorithms, consists of solutions that improve the system performance regarding qualitative indices, such as execution time and access delay. In the following, we review just a few more related studies in both categories, and other investigated ones are summarized in Table 1.

A. ENERGY-AWARE ALGORITHMS

As mentioned before, energy accounts for a large part of cloud operation costs. TPJS [39] was an energy- and locality-efficient MapReduce multi-job scheduling algorithm. The main characteristic of this solution was that its basic unit of resource allocation was a rack. The multi-job scheduling process was divided into two phases: multi-job pre-mapping and parallel job execution. In the first phase, multiple jobs were merged into a job group. Each job in the group was centrally pre-mapped to multiple booked racks. In the second phase, each reduced task of one job was mapped to multiple map tasks to form a task group. The performance of the algorithm was evaluated by comparing its results with those of three other typical methods, w.r.t. job scheduling time, resource balance rate, rack-to-rack traffic, number of used racks, and energy usage.

MOGA [16] was another proposed solution based on genetic algorithms. Its objectives were minimizing energy consumption, resource wastage, and bandwidth usage. MOGAs’ main idea was to place traffic-dependent VMs on the same PM, and if it was not possible, preferably to be placed under a zone of the access layer, otherwise under the aggregation layer, and in the worst case, under the core layer in tree architecture of the DC [16]. The results were compared with an ACO-based algorithm, a heuristic-based FFD solution, and a random-based approach.

B. SLA-AWARE ALGORITHMS

Improving cloud efficiency is another subject that some researchers tried to integrate into placement problem in different ways. The Purlieus model [29] was one of the first studies focused on data placement in big data applications. Its goal was to reduce the network distance between storage and compute nodes for both “map and reduce” processing. To this end, the authors tried to improve the data locality for MapReduce phases. They divided MapReduce tasks into three categories according to the volume of input data in every phase and then adopted a different strategy for each one. They showed that the combination of their two proposed techniques for VM placement improved the tasks’ execution speed by 9.1% to 100%, compared with other techniques.

In [8], Li et al. provided the CAM platform using a min-cost flow approach. Their goal was to reconcile data and VM resources’ initial allocation and migration to avoid placement anomalies. They used MapReduce tasks’ classification under a procedure similar to that of [29] and finally showed that their proposed algorithm made the network traffic three times lower and the speed of MapReduce tasks 8.6 times higher.

2First Fit Decreasing.
In another study by Shabeera et al. [14], they reduced network traffic and bandwidth utilization by placing computing and data nodes near servers. The presented heuristic algorithm was based on the ant colony optimization (ACO) model. To optimize the solution, data was stored in selected PMs’ storage. In the next step, a set of VMs was placed on the PMs to process the data. VM placement was according to the processing capacity of each PM. Comparing the simulation results with those of the FFDD and distance-aware FFD algorithms showed that using ACO-based algorithms had an effective role in reducing the processing time, as well as the distance between VMs located on different PMs.

Wei et al. [10] used a bipartite graph to model the distribution and connections of data and computing nodes to minimize the total transmission latency and maximum data transmission delay. They first divided VMs into pre-map and other categories according to their latency to the data nodes. Then, they proposed two placement optimization algorithms to minimize total latency and maximum data transmission latency for the map phase. Finally, VMs of the reduce phase were placed based on the lag time of data transmission between the map and reduce phases. The simulation results indicated that the application of this method decreased the average latency of data transmission by up to 26.3% compared to other methods.

Malekimajd and Movaghar [5] assumed that in networks with tree topologies, defining a new property brings the time complexity of VM assignment into the P (polynomial) class. However, it had been proved that the VM placement problem (minimizing latency) followed a triangular inequality; so, the best answer was at least twice the optimal answer. Using this method, the allocation of VMs was an optimal solution with minimal delay. However, the quadrilateral inequality was not necessarily approximated well in the generated network. The simulation results indicated that the algorithm function linearly depended on the number of switches and their delay.

C. DISCUSSION

Table 1 summarizes various solutions to the VM placement problem in big data tasks. Several aspects of related studies, such as objectives, employed techniques, resource considerations, etc., are investigated in the table.

A review of previous studies indicates that despite the dramatic progress in proposed solutions over time, there is still much to do to coordinate theoretical solutions with cloud service providers’ expectations [40]. The main points from our view are as follows:

- Wide use of laboratory assumptions such as homogeneous cloud environment, VM configuration, and considering one request at each moment;
- Uncertainty about the scalability of the algorithms; For instance, very few researchers have considered multi-DC cloud environments and have evaluated algorithms on large scales;
- Not being comprehensive enough; e.g., ignoring live migration overhead, RAM constraints, and binding proposed solutions on specific DC architectures;
- No discussion about the negative effects of algorithm implementation on other performance indicators, e.g., not paying attention to SLA violations and customer dissatisfaction;
- Ignoring big data processing requirements on the cloud.

IV. PROBLEM FORMULATION

To approach the problem, we used Knapsack as a common method to model the VM placement problem in the first step and then try to make the model richer, step by step. Table 2 shows the used notations in the AGAFF.

The Knapsack is an NP-hard problem [14], [15], [28], [29], [34], [35], including a set of items, each with a weight and a value. Our proposed method is to maximize the total value regarding the weight limitation. We formulate the problem as follows. If \( x_i, v_i, \) and \( w_i \) represent the item, value, and weight, respectively, we have:

\[
Max \sum_{i=1}^{n} x_i v_i \text{ or } \min \sum_{i=1}^{n} w_i (1 - x_i) \tag{1}
\]

Subject to:

\[
\sum_{i=1}^{n} x_i w_i \leq W \\
x_i \in \{0, 1\} \tag{2}
\]

where \( W \) is the maximum sustainable weight. Since the Knapsack problem is proven to be NP-hard, we use a constrained optimization method to solve it. To this end, we suppose every answer causes a violation that is not fixed. To measure the violation penalty, we define the equation below:

\[
vio = \max\left(\frac{\sum_{i=1}^{n} x_i w_i}{W} - 1, 0\right) \tag{3}
\]

In the next step, we want to optimize the violation penalty. Therefore, we turn the constrained problem into an unconstrained one using the penalty function:

\[
z' = z + \alpha \nu \tag{4}
\]

\[
z' = \min \sum_{i=1}^{n} w_i (1 - x_i) + \alpha \left(\frac{\sum_{i=1}^{n} x_i w_i}{W} - 1, 0\right) \\
\alpha > 0 \tag{5}
\]

Now, we map the VM placement problem to Knapsack. To manage and control the placement process, we use a central control system, which is aware of the cloud system, available resources, servers’ specifications, and other necessary information in service provisioning. We consider a distributed cloud environment containing \( m \) physical machines (PM1, PM2, PM3, ..., PMm) to define the system. Every PM has \( k \) component resources. Our problem is to place \( n \) VMs on available PMs to minimize the total energy consumption, CPU and RAM energy usage, interrelated traffic between
| Research Name                   | Cloud Environment | Objective                                                                 | Used Technique   | Considered Resources | Compared Algorithms | Strengths                                                                 | Weaknesses                                                                 |
|--------------------------------|-------------------|---------------------------------------------------------------------------|------------------|----------------------|---------------------|---------------------------------------------------------------------------|----------------------------------------------------------------------------|
| Purlicus (2011) [29]           | In one DC, Homogenous | Improving runtime performance, Reducing traffic                         | Graph            | CPU, Bandwidth       | Random              | Implementation on a small number of PMs                                   | The static solution, Incompatibility with changes                          |
| CAM (2012) [8]                 | In one DC, Homogenous | Reducing network traffic and job execution time                          | Graph            | CPU, Bandwidth       | Vanilla Hadoop algorithms in [29]                                      | Being dynamic with changes                                                | Working under specified constraints                                        |
| VMPDN (2014) [9]               | In one DC          | Minimizing the maximum access latency among nodes                        | Linear programming | Bandwidth            | The lower bound results of their algorithm                             | Bounding the maximum access latency                                       | Not being comprehensive, Discarding VM and PM resource constraints          |
| Malekimajd and Movaghar (2017) [5] | In one DC, Homogenous | Minimizing communication latency                                          | Graph            | -                    | The algorithm proposed in [9]                                          |                                                                            | Processing only one request at a time, Not being comprehensive              |
| Wei et al. (2017) [10]         | In one DC, Homogenous | Reducing data transmission latency                                         | Linear programming | -                    | A placement approach with triangle inequality                          |                                                                            | VM placement is based on the data nodes' random location                  |
| Hall et al. (2017) [28]        | In one DC, Homogenous | Reducing data retrieval time of large datasets and energy consumption    | Linear programming | CPU, RAM             | Random- FF and Optimal                                                 | Considering I/O bottlenecks in infrastructure                             | High possibility of uneven load on PM, Negligible role of RAM and CPU in placement |
| Shabbera et al. (2017) [14]    | Multi DC cloud, Homogenous | Reducing cross-network traffic and bandwidth usage                     | Ant colony optimization | CPU, RAM             | FFD and distance-aware FFD                                              | An innovative solution to consider data and VM placement                   | Several simplifications to reach the goal                                   |
| Guerrero et al. (2018) [15]    | In one DC, Homogenous | Minimizing the power consumption, Physical resource waste, File unavailability | Genetic algorithm | CPU, Bandwidth       | Two other heuristic algorithms: NSGA-VM and AIA-block                  | Being multidimensional                                                     | A time-consuming algorithm in comparison to others                         |
| Han et al. (2019) [31]         | In one DC, Homogenous | Improving the energy usage efficiency of PMs                             | Knapsack         | CPU, RAM             | FFD                                                                | An innovative and simple solution to involve the estimated running time in the placement algorithm | Ignoring the additional energy usage in relocating VMs                      |
| Farzai et al. (2020) [16]      | In one DC, Homogenous | Energy-saving, Resource wastage reduction, and Traffic reduction          | Genetic algorithm | CPU, RAM, Bandwidth | An ACO-based FFD, and Random                                           | Regarding different resource components and multi objectives              | Resource heterogeneity and migration process are not considered             |
| Rawas and Zekri (2020) [30]    | In one DC, Homogenous | Maximizing the utilization of resources, Minimizing energy consumption, Reducing traffic | Genetic algorithm | CPU, RAM, Bandwidth   | Best Fit, Worst Fit, Combined Least Full First, and Combined Most Full First |                                                                            | Not being applicable in new network topologies                              |
| Tenghert et al. (2021) [32]    | In one DC, Homogenous | The trade-off between performance & energy consumption                   | Game theory      | CPU                  | -                                                                   | The trade-off between response time (RT) and the power consumption (PC)  | Only implemented for two VMs                                                |
big data VMs, and VMs migration overhead. In addition, performance indices should be in the best possible situation.

We suppose every PM is a bin, and $x_i$ represents the VMs that should be located. It includes running and newly arrived VMs at each moment, and $w_i$ denotes the required processing resources for each VM. The problem is to place as many as VMs in every PM, whereas the sum of processing resources will not be more than PMs'.

### A. CLOUD ENERGY MODEL

As stated before, minimizing energy usage is the objective of AGAFF since it has a great impact on cloud providers’ OPEX. It is also an important step toward green computing. The main factors of energy consumption are running PMs, transferred traffic in the data centers, and VMs live migration. Energy consumption is calculated by (6):

$$E_{Total} = E_{Server} + E_{Bigdata} + E_{Livemig} \quad (6)$$

Total energy ($E_{Total}$) is composed of three components, i.e., server, big data, and live migration, which are elaborated below:

1) SERVER
This item has the highest impact on data center energy usage [28], [35], [36], [41], [42], [43]. Therefore, we have to use all the physical servers’ capacity while the disaster threshold is regarded in every PM. To calculate consumed energy in servers, we consider two states for PMs:

- The energy consumption of an idle server that is in hibernate state is regarded to be zero.
- A server containing running VMs. This item is a function of PMs’ components, especially CPU and RAM, which play significant roles in servers’ processing power [45].

Therefore, we calculate the energy usage of PMs as follows:

$$E_{Servers} = \sum_{i=1}^{m} E_{CPU}(i) + \sum_{i=1}^{m} E_{mem}(i) + m \cdot E_{baseline} \quad (7)$$

where $m$ is the number of PMs, $E_{CPU}$ and $E_{mem}$ represent the energy consumption of CPU and memory, respectively, and $E_{baseline}$ is the base power usage that is empirically determined. $E_{baseline}$ represents the energy consumption of a server when no user-level process is active [44]. The formula is the same for heterogeneous and homogenous servers.

Energy waste in PMs are inversely related to servers’ processing power usage. In optimum solutions, the energy waste must be minimum by using the maximum processing capacity of servers. So, we try to decrease energy losses in AGAFF as much as possible. To calculate the energy waste ratio, we define (8) when server $i$ is turned on.

$$EW_i = E_{CPU} \left(\frac{CPUCap (i) - \sum_{j=1}^{n} U_{VMj}^{CPU}}{CPUCap (i)}\right) + E_{RAM} \left(\frac{RAMCap (i) - \sum_{j=1}^{n} U_{VMj}^{RAM}}{RAMCap (i)}\right) \quad (8)$$

Equation (9) calculates the servers’ total energy waste ratio.

$$EW = Ave(EW_i) \quad (9)$$

### TABLE 1. (Continued.) Related work summary.

| Sadeq et al. (2021) [33] | In one DC, Heterogeneous | Reducing intra-DC network traffic | Graph, Linear programming | CPU, Bandwidth | Best Fit, Round Robin, Random, and Greedy algorithms | A two-phase VM placement strategy | Ignoring RAM and some server thresholds |
|--------------------------|--------------------------|---------------------------------|--------------------------|---------------|---------------------------------------------------|---------------------------------|-------------------------------------|
| Peake et al. (2022) [2]  | In one DC, Homogeneous and Heterogeneous | Improving runtime performance | Parallel Ant Colony Optimization (PACO) | CPU, RAM | Genetic Algorithm and ACO | Parallel implementation of algorithm | Assumptions for simplifying the problem |
| Balaji et al. (2022) [3] | In one DC, Homogenous | Scheduling the maximum VMs in every server as much as possible and minimizing the number of active servers | Discrete firefly algorithm | CPU, RAM | Genetic Algorithm and Particle Swarm Optimization | Experimental results for Memory-intensive and CPU-intensive VMs | Ignoring servers’ resources thresholds |
| Our proposed solution (AGAFF) | Multi DC cloud, Heterogeneous and Homogeneous | Reducing energy usage in big data processing and Concerning SLA obligations, simultaneously | Configurable genetic algorithm, First fit | CPU, RAM, Bandwidth | GARand, Random, First Fit, Best Fit | Implementing cloud service providing priorities in solution, Improving energy usage level and SLA obligations at the same time | No awareness of other new cloud processing requirements such as IoT |
TABLE 2. Description of AGAFF notations.

| Symbol        | Definition                                                                 |
|---------------|-----------------------------------------------------------------------------|
| $E_{\text{Total}}$ | Total energy consumption of cloud data centers                          |
| $E_{\text{Server}}$ | Energy consumption by PMs                                                  |
| $E_{\text{CPU}}$ | Energy consumption of a central processing unit in PMs                     |
| $E_{\text{RAM}}$ | Energy consumption of a random-access memory unit in PMs                   |
| $E_{\text{switch}}$ | Energy consumption of a PM when no user-level process is active            |
| $E_{\text{inet}}$ | Consumed energy to transmit big data traffic between VMs                   |
| $E_{\text{CPUviol}}$ | Consumed energy to migrate PMs                                             |
| $\text{CPUCap}(i)$ | CPU capacity of PM$_i$                                                    |
| $\text{CPUNeed}(i,j)$ | Requested CPU for VM$_j$                                                  |
| $\text{RAMviol}$ | Sum of RAM violation of determined constraints                             |
| $\text{RAMCap}(i)$ | RAM capacity of PM$_i$                                                     |
| $\text{RAMNeed}(i,j)$ | Requested RAM for VM$_j$                                                  |
| $\text{Bigdataviol}$ | Total number of the hops in big data tasks’ traffic route                 |
| $\text{Migrationviol}$ | Total number of VMs’ migration                                             |
| $\text{Migrationnum}$ | Number of migrations                                                       |
| $\text{At}(VM,VM_j)$ | Interrelation between VM and VM$_j$                                       |
| $\text{PM}(VM_i)$ | The PM on which VM$_i$ is hosted                                           |
| $\text{Rack}(VM_i)$ | The rack on which VM$_i$ is hosted                                         |
| $\text{DC}(VM_i)$ | The data center on which VM$_i$ is hosted                                   |
| $U_{\text{cpu}}(VM_i,PM_i)$ | Used CPU by VM$_i$ on PM$_i$                                               |
| $U_{\text{ram}}(VM_i,PM_i)$ | Used RAM by VM$_i$ on PM$_i$                                               |
| $EW_{i}$ | Total energy wastage                                                      |

2) BIG DATA TRAFFIC

High cohesion VMs and transferring large amounts of data between VMs are two characteristics of big data tasks. The traffic of VMs can be supplanted through the Internet or the internal network of the data center. In the case of using the Internet, VM positions are not important, but when the internal network is used, the placement of VMs, especially correlated ones, becomes important. As the servers are farther, the energy usage of the network is higher. As stated in [45], network traffic is a determining factor in data center energy consumption.

To diminish energy usage, interrelated VMs are preferred to be placed on the same server. If it is not possible, sitting in the same rack is the second choice, and being in distinct racks in the same data center is the last option. Although we evaluate the performance of AGAFF in a multi-DC cloud environment, it avoids placing interrelated VMs of a big data task in different data centers.

Upon entering every big data task, AGAFF’s central control system generates an $n \times n$ top triangular matrix. Adjacency matrix shows which VMs are interrelated:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}$$

where $a_{ij}$ represents the relation between VM$_i$ and VM$_j$. If $a_{ij}$ is 1, it means VM$_i$ and VM$_j$ are interrelated. Otherwise, it means there is no special relation between VM$_i$ and VM$_j$.

By using matrix $A$ and based on the number of hops between every two VMs in leaf-spine data center topology, we define Bigdataviol. This index displays the imposed cost by big data VMs’ traffic (Bigdataviol calculation is described in the next section). The cost is calculated based on the number of hops between interrelated VMs in a task. Now, we can compute the energy consumption of big data traffics by (10):

$$E_{\text{Bigdata}} = \text{Bigdataviol} \times \text{Trafficenergy} \quad (10)$$

3) LIVE MIGRATION

In every run of AGAFF, we assume $n$ VMs have recently arrived and should be located. Moreover, there are some running VMs. To manage the VM placement, we categorize all VMs into two classes:

1. Running VMs (VM$_0$).
2. Newly arrived VMs (VM$_n$).

Sometimes, an optimum VM placement needs several VM migrations. VM migration is an effective capability in modern cloud architectures. This capability can be implemented in OpenStack$^3$ based on shared storage like SAN (storage area network) or Ceph.$^4$ Since VM live migration consumes extra energy due to the increased data center traffic and raises processing loads, it is better to minimize the number of live migrations.

To compute live migration energy consumption, we use the statefulness capability of AGAFF. Every time new VMs

---

$^3$ https://www.openstack.org.

OpenStack is a set of software components that provides common services for cloud infrastructure.

$^4$ https://ubuntu.com/ceph/

Ceph is a software-defined storage solution designed to address the object, block, and file storage needs of data centers adopting open source as the new norm for high-growth block storage, object stores, and data lakes.
arrive, a new placement process runs. Therefore, we have two matrices at the moment. One of them represents VMs’ location in the previous state, and the other one denotes new placement. A comparison between the number of the host PM of every VM in these two matrices determines the number of necessary migrations. To equalize matrix ranks, all the PM numbers of new VMs in the previous matrix are assumed to be zero. By having the number of migrations, energy consumption can be calculated using (11):

\[ E_{Livemig} = MigrationNum \times MigrationOverheadEnergy \]  

(11)

V. AGAFF ALGORITHM

AGAFF is the smart combination of evolutionary genetic algorithms and greedy first fit. Using the first fit algorithm optimizes some processes of the genetic algorithm (GA). This algorithm is context-aware, i.e., it is aware of cloud topology, type of input data, and temporal data. AGAFF tries to minimize multi-DC cloud energy usage such that the performance indices be in the optimum state. In the following, the structure of AGAFF is explained.

Over recent years, many researchers have used GA to solve VM placement/consolidation problems [46]. GA belongs to the class of evolutionary algorithms [47]. It works through the natural selection process and is well-known for solving multi-objective optimization problems [30]. GA is used due to its efficient, parallel, and global search characteristics [48]. The algorithm’s complexity depends on some factors, such as the number of generations, number of iterations, and size of the mentioned population of chromosomes.

As a basic component in GA, a chromosome is made up of genes. The length of every chromosome is based on the number of VMs. For example, Figure 2 shows a chromosome sample in which the number of VMs and PMs are 8 and 5, respectively.

| VM Number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----------|---|---|---|---|---|---|---|---|
| Number    |   |   |   |   |   |   |   |   |
| of PM     | 3 | 4 | 5 | 5 | 5 | 1 | 2 | 2 |

**FIGURE 2. A sample of chromosome.**

We are going to place n VMs on m PMs such that energy consumption and intra-DC traffic become minimized, and components’ usage of every server becomes balanced. In placing VMs, resource capacity is a vital factor, and the algorithm should consider every PM’s capacity while placing VMs. To observe resource constraints, we define the following equations:

\[ U_{VM1}^{CPU} + U_{VM2}^{CPU} + \ldots + U_{VMj}^{CPU} \leq \eta C^{CPU} \]  

(12)

\[ \sum_{i=1}^{j} U_{VMi}^{CPU} \leq \eta C^{CPU} \]  

(13)

Equation (13) expresses the used CPU by VMs co-hosted on PMi to be less than a definite ratio of CPU capacity of PMi. “\( \eta \)” is the CPU disaster threshold in the real world which means some capacity of CPU must always be free.

Similarly, we define (14) for RAM usage:

\[ \sum_{i=1}^{j+\alpha} U_{VMi}^{RAM} \leq \rho C^{RAM} \]  

(14)

A. INITIAL POPULATION

GA forms the initial population randomly by default. This procedure increases the number of solution iterations to reach the best results, which produces a longer delay. AGAFF is improved in setting the initial population. To conduct a smart initialization, we combine random and first fit algorithms’ results. One-third of the first generation is randomly generated. In another one-third, VM placement is done based on the requested CPU according to the first fit method. The first PM to start placement is chosen randomly. The other one-third of the population is generated by using the first fit method based on the requested RAM.

B. CROSSOVER AND MUTATION OPERATORS

After generating the initial population, a crossover operator is applied. The crossover operator defines how two parents are combined to obtain two offspring [15]. We use a one-point crossover, and PC% of parents are chosen randomly.

Mutation, another operator used in GA, aims to produce a random twitch of genes in the chromosome to introduce the diversity of the chromosomes [47]. In AGAFF, the mutation operator chooses the PM% of chromosomes. Then, one of its genes (for example, x) is selected randomly. To mutate the chromosome, [x/2] replaces x, lowering the VM placement dispersion. Accordingly, some mutated chromosomes are added to the initial population.

At this step, all the population is sorted based on cost value. Chromosomes with lower costs are preferred. The algorithm repeats the whole process until the termination condition is met. The termination condition can be determined based on operational conditions of the environment or the rate of tasks entering the cloud. We set AGAFF’s number of iterations as the termination condition. Figure 3 and figure 4 depict the process of the crossover operator and a sample of the mutation operator, respectively.

C. COST FUNCTION

As stated in the previous section, AGAFF’s main objective is minimizing total cloud energy consumption. We should consider several constraints to reach an applied solution. To achieve the goal, we define a cost function by (15). AGAFF selects answers with the minimum cost.

\[ \text{Minimum Cost Function} = \text{Number of Used PMs} + \text{Minimum Total Violation} \]  

(15)

“Total Violation” is composed of four components: CPU violation, RAM violation, live migration violation, and big data violation.
1) CPU VIOLATION
According to (5), we can turn a constrained problem into an unconstrained one using the penalty function below:

\[
CPU\text{Viol}(i) = \max \left( \sum_{i=1}^{m} \sum_{j=1}^{n} CPU\text{Need}(j); if x(i) == 1 \right) - CPU\text{Cap}(i), \quad (16)
\]

Now, we can calculate the total CPU violation for every placement algorithm implementation using (17):

\[
CPU\text{Viol} = \frac{\sum_{i=1}^{m} CPU\text{Viol}(i)}{m}, \quad (17)
\]

2) RAM VIOLATION
Similar to CPU violation, we define RAM violation for PM, and the whole in (18) and (19), respectively:

\[
RAM\text{Viol}(i) = \max \left( \sum_{i=1}^{m} \sum_{j=1}^{n} RAM\text{Need}(j); if x(i) == 1 \right) - RAM\text{Cap}(i), \quad (18)
\]

\[
RAM\text{Viol} = \frac{\sum_{i=1}^{m} RAM\text{Viol}(i)}{m}, \quad (19)
\]

3) MIGRATION VIOLATION
Sometimes, the scheduling program should move some of the running VMs to reach the best answer. VMs’ live migration in cloud computing solutions imposes a processing load on the system and involves CPU cores to handle the relocations. More number of live migrations leads to greater processing overhead. Furthermore, these replacements may cause a slight interruption in customers’ services, worsening the user experience. Therefore, we try to minimize live migrations in AGAFF. To count this violation, two similar matrices are generated at every run of AGAFF. The number of migrations can be extracted by comparing these matrices:

\[
Migration\text{Viol}(j) = \begin{cases} 0 & \text{PreviousX}(j) \neq X(j) \text{ and PreviousX}(j) \neq 0 \\ 1 & \text{else} \end{cases}
\]

where previousX(j) and X(j) are linear matrices indicating every VM’s location in the last and current run of AGAFF, respectively.

Now, MigrationViol is calculated using (20):

\[
Migration\text{Viol} = \sum_{j=1}^{n} Migration\text{Viol}(j), \quad (20)
\]

4) BIG DATA VIOLATION
One of the costly challenges is the traffic produced by big data tasks within a data center. A good way to manage the traffic is by optimizing the placement of VMs on the cloud. AGAFF is a context-aware algorithm. In other words, when a big data task arrives, we assign the traffic matrix to it. We define highly cohesive VMs whose data transfer in \(\Delta t\) is more than a threshold.

\[
A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix}
\]

where \(a_{ij}\) is a binary variable:

\[
a_{ij} = \begin{cases} 1, & \text{if vm}_i \text{ and vm}_j \text{ have high cohesion} \\ 0, & \text{otherwise} \end{cases}
\]
To model the cloud data centers, we consider a multi-DC cloud with a leaf-spine topology [12]. Nevertheless, AGAFF can match any other DC architecture. The leaf-spine architecture helps cloud providers reduce costs and deliver high-quality services to customers. Unlike traditional three-tier architectures, it is not important which leaf switch is connected to which server in leaf-spine topology. In this structure, the traffic always has the same hop count to reach another server [49]. In a leaf-spine data center, if two interrelated VMs are located on the server, the energy consumed to transfer the traffic is assumed zero. If these VMs are placed on different servers but the same rack, we consider one-hop energy consumption. At last, if the highly cohesive VMs are located on different racks, two hops are regarded based on leaf-spine topology. A typical structure of the leaf-spine structure is shown in Figure 5.

Here, it must be noticed that we put some limitations on the placement of big data tasks. Since considerable traffic is transmitted between highly cohesive VMs, the algorithm is not allowed to place these VMs in different data centers. This helps save energy by reducing the costs of data transfer by lowering the running latency of big data tasks, as in (21), shown at the bottom of the page.

5) TOTAL VIOLATION
After normalizing the four aforementioned components, the Total Violation is calculated as follows:

\[
Total\ Violation = \alpha \times CPU\ Violation + \beta \times RAM\ Violation + \gamma \times Migration\ Violation + \delta \times Big\ data\ Violation \quad (22)
\]

\[
BigdataViol = \sum_{j=1}^{n} BigdataViol(j)
\]

\[
BigdataViol (i, j) = \left\{ \begin{array}{ll}
0 & A(VM_i, VM_j) = 0 \\
0 & A(VM_i, VM_j) = 1, PM_{VM_i} = PM_{VM_j} \\
1 & A(VM_i, VM_j) = 1, PM_{VM_i} \neq PM_{VM_j}, Rack_{VM_i} = Rack_{VM_j} \\
2 & A(VM_i, VM_j) = 1, PM_{VM_i} \neq PM_{VM_j}, & Rack_{VM_i} \neq Rack_{VM_j}, DC_{VM_i} = DC_{VM_j}
\end{array} \right. \quad (21)
\]

Observations based on our experience in XaaS Public Cloud show that in the real world, RAM violation has the main impact on system performance, and its violation can result in the failure of the computing process. Figure 6 and Figure 7 show the AGAFF flowchart and the pseudocode of the AGAFF, respectively.

VI. EXPERIMENTAL RESULTS AND DISCUSSION
To evaluate AGAFF performance, a multi-DC cloud environment and our placement algorithm are simulated using MATLAB language and Octave open-source software V4.4. Server specifications are selected based on available instances in XaaS Public Cloud. Most of the servers are
HP G9, and their CPU model is Xeon series. Moreover, the server storage and RAM types are SSD and RDD4, respectively.

Different experiments are performed to assess AGAFF efficiency. Some parts of conducted tests are dedicated to study the main objectives of AGAFF, which are energy consumption and execution time reduction. Other experiments are done to assess generated violations to reach the best result. Every test is performed ten times, on average. To provide a precise assessment of AGAFF performance, four algorithms with different methodologies are compared. The algorithms are as follows: GARand (Genetic Algorithm with Random initial population), which is the primitive version of AGAFF with a random fit algorithm in the population generation process, along with Random, First Fit (FF), and Best Fit (BF), which are common algorithms in computer science.

We have investigated AGAFF scalability by increasing the number of requested VMs from 50 to 250 at each request. The consumed energy is calculated based on the standard energy consumption of mentioned component resources, and we set 10w and 0.375w energy usage for a CPU core and 1GB RAM, respectively. In addition, 1GB is the average traffic transmitted between high cohesion VMs. Table 3 shows the configurations of the cloud, PMs, and AGAFF algorithm.

### Table 3. Configurations of the cloud, PMs, and AGAFF algorithm.

| Algorithm item                  | Value          |
|---------------------------------|----------------|
| Number of Population            | 100            |
| Percent of Crossover            | 80             |
| Percent of Mutation             | 60             |
| Number of PMs                   | 1000           |
| Number of CPUs in a PM          | 32 to 120 cores|
| RAM of a PM                     | 64 to 512 GB   |
| Number of CPUs in a VM          | 2 to 64 cores  |
| RAM of a VM                     | 4 to 256 GB    |

### A. RESULTS AND ANALYSIS

1) CLOUD ENERGY CONSUMPTION

Figure 8 depicts the performance of AGAFF in terms of energy consumption. To assess AGAFF behavior in different scales of VMs, figure 8(a) and figure 8(b) show simulation results on the scale of 50 VMs to 250 VMs and 1000 VMs to 2000 VMs, respectively. Results show that implementing AGAFF decreases energy consumption by 55% on average in figure 8(a). It also reduces the energy usage by 68% in the high scale of VMs on average. If we omit the Random algorithm in our calculation, AGAFF declines the usage by 62% on average. The improvement in energy usage on different scales of VMs shows that AGAFF is scalable and it can optimize energy consumption without any limitations in the highly loaded clouds.

Optimizing the number of used PMs, minimizing big data traffic through better placement, and decreasing the number of VM migrations by AGAFF are the main reasons for energy usage reduction. Furthermore, as mentioned before, AGAFF avoids placing interrelated VMs of a big data task in different data centers. This strategy plays a significant role in decreasing intra-DC big data traffic in a multi-DC cloud.

The other four algorithms have different reactions to increasing the scale of VMs. In GARand, converging to the best answer needs a high number of iterations and it requires a long time to accomplish. Random does not follow a particular method to solve the problem and its results fluctuate on different scales of VMs. FF does not have proper performance in the low number of VMs, but it shows more acceptable outcomes in the high number of VMs. However, it is shown in Figure 10 that FF’s total violation goes up considerably
in high scale VMs. BF shows inefficient results on all scales of VMs.

In Figure 9, the energy waste ratio of AGAFF and other algorithms are represented. As mentioned in section 4, the energy waste is calculated based on servers’ resources that are not used. In other words, the energy waste ratio gets higher if more CPU cores and RAM capacity in working servers are unused. Figure 9(b) shows that the AGAFF’s energy waste ratio is less in a highly loaded cloud. The other algorithms have the same approach roughly. Random is an exception. Since it uses the normal distribution, it scatters the VMs across a more number of PMs. This approach causes much more energy waste ratio in figure (b). AGAFF has the minimum energy wastage ratio in comparison to others in all the tested scales.

2) ALGORITHM EXECUTION TIME

Figure 10 shows two charts that display the number of function evaluations (NFE) required to reach the optimum answer. This experiment is only done for AGAFF and GARand, since converging to an optimum answer makes sense only in evolutionary algorithms. In Figure 10(a), the number of function evaluations is compared between AGAFF and GARand. As explained in previous sections, a substantial advantage of incorporating two heuristic algorithms (GA and FF) is the smarter generation of the initial population. This improvement causes the answer of primary iterations to be very near the optimum answer, lowering the number of iterations and, thus, the algorithm execution time.

3) OTHER RESULTS OF THE AGAFF ALGORITHM

Figure 11(a) shows the number of PMs used by different algorithms on the scale of 50 VMs to 250 VMs. As one can see, decreasing the number of used PMs by up to 80% by AGAFF improves resource utilization efficiency. AGAFF tries to decrease VMs’ scattering on PMs. This strategy in the VM placement is the main reason for the low number of PMs used in AGAFF. Results in figure 11(c) show that implemented strategy works in optimizing VM placement in the high number of VMs, too. AGAFF can lessen the number of used PMs by about 73% on a scale of more than 1000 VMs. An interesting point is that the same strategy is applied in GARand, but since the initial population is generated randomly, the number of PMs does not approach the optimum number in the determined time. So, by increasing the number of VMs and enhancing the necessity for more iterations, GARand’s results get worse concerning AGAFF results. Random presents the worst results in scat-
The number of servers turned on in Random’s results is more than in other algorithms.

Figure 11(b) demonstrates the total violation of algorithms’ execution, indicating the undesired effects of algorithms on the scale of 50 VMs to 250 VMs. As explained, the Total Violation contains four subordinates, each of which has different coefficients regarding its importance in the desired answer. Since big data tasks are important in our solution, the big data violation has the largest coefficient, and RAM violation, CPU violation, and the percent of live migration are in the next ranks. It can be seen that AGAFF’s total violation is lower than other algorithms in figure 11(b), except for GARand. This behavior of GARand can be justified by comparing its performance in Figures 11(a and b). Similar to AGAFF, GARand is an evolutionary algorithm that works based on genetics. However, GARand uses the random method to generate the initial population. Figure 11(a) shows the impact of using different methods in generating the initial
population. AGAFF optimizes the used PMs by about 80% on the scales of the low number of VMs. On the other side, using up to five times more PMs in GARand reduces the total violation by about 20% in comparison to AGAFF. Both numbers of the used PMs and total violation show that AGAFF produces the best results in comparison to other algorithms.

By increasing the number of input VMs in figure 11(d), AGAFF advantages become more transparent. Results in figure 10(d) show that in the high number of VMs (more than 1000), AGAFF’s total violation is the least among investigated algorithms. Time-consuming processes by GARand are the factor of rising total violation in determined time. Other algorithms’ behavior in the total violation indicator is the same in all the scales of VMs. To show the behavior of the components of total violation on the scale of 100 VMs, four charts are depicted in figure 12. Based on the results, both AGAFF and GARand can show the best performance in two components out of the four ones. Figure 12(a) shows the created big data violation by algorithms. AGAFF can decrease big data violation as a result of paying attention to the placement of high cohesion VMs. Figure 12(b) displays the migration percentage in the execution of algorithms. AGAFF is successful in decreasing the number of migrations and has better performance in comparison to others. If we did not consider the live migration overhead in AGAFF ($\gamma = 0$), the total violation would be decreased generally. However, the overhead is included to put real-world limitations in AGAFF. Figures 12(c and d) represent the violation of PMs’ resource components in the placement process. In these items, GARand shows better functionality as a result of using more PMs compared with AGAFF. It is worth noting that in BF and FF, VMs are located in PMs based on the required RAM. It results in zero RAM violation and notable CPU violation. We have chosen RAM as a more critical resource given our observations in the XaaS Public Cloud. However, this experiment can be done by giving priority to the CPU, which replaces the results of RAM violation with CPU violation. If the scheduling is executed based on the required RAM and CPU in BF and FF at the same time, the number of used PMs goes up significantly.

VII. CONCLUSION

The emergence of cloud computing has changed many aspects of computing. It provides an economical and efficient platform for high-level technologies, such as big data, IoT, artificial intelligence, and edge computing. However, its expansion may cause some challenges like high-scale energy consumption, CO$_2$ emissions, and global warming. Many researchers have focused on related subjects to decrease the side effects of cloud computing by optimizing the VM placement. Presenting various solutions to place requested VMs optimally is an effective way toward energy usage reduction and green computing.
Nevertheless, due to the extensive use of laboratory assumptions and ignoring many aspects of the problem or negative effects of implementing the presented solutions, most of the proposed solutions are not applicable in a practical environment. AGAFF is a novel multi-objective VM placement algorithm that smartly combines genetic algorithm and the first fit (FF) method. FF algorithm is used to generate the initial population intelligently. We have proposed AGAFF based on our experiences and observations in delivering service in the XaaS Public Cloud in the last seven years. AGAFF decreases the energy consumption of a multi-DC cloud environment compared to other algorithms considering the main customer challenges like execution time, SLA obligations, and VM migration overhead. To optimize the undesired effects of the implementation of the scheduling algorithms, we have defined a cost function, which is included multiple parameters, such as intra-DC traffic, possible usage of every PM’s resource components, relevant usage percent of resource components in every PM, and the number of VMs’ migration.

The AGAFF results are compared with four algorithms (GARand, Rand, FF, and BF) on different scales from 50 to 2000 VMs. Evaluations show that AGAFF reduces energy consumption by 61% on average such that its execution time and cost are lower than the compared algorithms. That is, AGAFF can minimize energy usage while managing other impacts on service provisioning.

In addition, AGAFF considers interleaved VMs in big data programming frameworks like MapReduce, and it decreases the traffic of big data tasks in a data center by a minimum of 50%. Furthermore, it avoids placing interleaved VMs of a big data task in different data centers and diminishes the traffic between data centers in a multi-DC cloud.

To improve the AGAFF performance, we plan to export the statistical distribution of big data tasks from public cloud service providers and present a more customized AGAFF for big data applications. Moreover, customizing AGAFF for IoT and edge computing are important challenges that could be targeted in the next revisions of the AGAFF.

REFERENCES

[1] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, “The rise of ‘big data’ on cloud computing: Review and open research issues,” Inf. Syst., vol. 47, pp. 98–115, Jan. 2015. doi: 10.1016/j.is.2014.07.006.
[2] J. Peake, M. Amos, N. Costen, G. Masala, and H. Lloyd, “PACO-VMP: Parallel ant colony optimization for virtual machine placement,” Future Gener. Comput. Syst., vol. 129, pp. 174–186, Apr. 2022. doi: 10.1016/j.future.2021.11.019.
[3] K. Balaji, P. S. Kiran, and M. S. Kumar, “Power aware virtual machine placement in iaas cloud using discrete firefly algorithm,” Appl. Nanosci., pp. 1–9, Feb 2022. doi: 10.1007/s13204-021-02357-0.
[4] A. R. Ilkhechi, I. Korpeoglu, and Ö. Ülusoy, “Network-aware virtual machine placement in cloud data centers with multiple traffic-intensive components,” Comput. Netw., vol. 91, pp. 508–527, Nov. 2015. doi: 10.1016/j.comnet.2015.08.042.
[5] M. Malekimajd and A. Movaghar, “Minimizing data access latencies for virtual machine assignment in cloud systems,” IEEE Trans. Services Comput., vol. 13, no. 5, pp. 857–870, Sep. 2020. doi: 10.1109/TSC.2017.2735972.
[6] G. Singh, M. Malhotra, and A. Sharma, “A comprehensive study on virtual machine migration techniques of cloud computing,” in Applications of Computing, Automation and Wireless Systems in Electrical Engineering. Singapore: Springer, 2019, pp. 591–603.
[7] B. Balachandran and S. Prasad, “Challenges and benefits of deploying big data analytics in the cloud for business intelligence,” in Proc. Int. Conf. Knowl. Based Intell. Inf. Eng. Syst., Marseille, France, 2017, pp. 1112–1122.
[8] M. Li, D. Subhraveti, A. R. Butt, A. Khasymski, and P. Sarkar, “CAM: A topology aware minimum cost flow based resource manager for MapReduce applications in the cloud,” in Proc. 21st Int. Symp. High-Performance Parallel Distrib. Comput., Delft, The Netherlands, 2012, pp. 211–222.
[9] I.-I. Kuo, H.-H. Yang, and M.-I. Tsai, “Optimal approximation algorithm of virtual machine placement for data latency minimization in cloud systems,” in Proc. IEEE Conf. Comput. Commun. (IEEE INFOCOM), Toronto, ON, Canada, Apr. 2014, pp. 1303–1311.
[10] J. Wei, S. Wang, L. Zhang, A. Zhou, Q. Sun, and F. Yang, “Virtual machine placement to minimize data transmission latency in MapReduce,” J. Internet Technol., vol. 18, no. 6, pp. 1359–1391, Nov. 2017.
[11] S. Jan, M. Da Li, N. Ji, and B. Li, “An online virtual machine placement algorithm in an over-committed cloud,” in Proc. IEEE Int. Conf. Cloud Eng. (IC2E), San Francisco, CA, USA, Apr. 2018, pp. 106–112.
[12] T. Duong-Ba, T. Nguyen, B. Bose, and T. Tran, “Joint virtual machine placement and migration scheme for datacenters,” in Proc. IEEE Global Commun. Conf., Austin, TX, USA, Dec. 2014, pp. 2320–2325.
[13] Z. A. Mann and M. Szabó, “Which is the best algorithm for virtual machine placement optimization?” in Proc. 16th Int. Conf. Comput. Commun. (Concurr. Comput. Pract. Exp.), vol. 29, no. 10, p. e4083, May 2017, doi: 10.1002/cpe.4083.
[14] T. P. Shabeera, S. D. M. Kumar, S. M. Salam, and K. M. Krishnan, “Optimizing VM allocation and data placement for data-intensive applications in cloud using ACO metaheuristic algorithm,” Eng. Sci. Technol. Int. J., vol. 20, no. 2, pp. 616–628, Apr. 2017, doi: 10.1016/j.estech.2016.11.006.
[15] C. Guerrero, I. Lera, B. Bermejo, and C. Juiz, “Multi-objective optimization for virtual machine allocation and replica placement in virtu-alized Hadoop,” IEEE Trans. Parallel Distrib. Syst., vol. 29, no. 11, pp. 2568–2581, Nov. 2018, doi: 10.1109/TPDS.2018.2837743.
[16] S. Farzai, M. H. Shirvani, and M. Rabbani, “Multi-objective communication-aware optimization for virtual machine placement in cloud datacenters,” Sustain. Comput., Informat. Syst., vol. 28, Dec. 2020, Art. no. 100374, doi: 10.1016/j.suscom.2020.100374.
[17] M. Kumar, S. Sharma, A. Goel, and S. Singh, “A comprehensive survey for scheduling techniques in cloud computing,” Inf. Syst. Comput. Appl., vol. 143, pp. 1–33, Oct. 2019, doi: 10.1016/j.isca.2019.06.006.
[18] Definition of Cloud Computing, Nat. Inst. Standards Technol., Gaithersburg, MD, USA, 2011.
[19] X. Chen and J.-H. Jiang, “A method of virtual machine placement for fault-tolerant cloud applications,” Infell. Autom. Soft Comput., vol. 22, no. 4, pp. 587–597, Mar. 2016, doi: 10.1016/j.tsc.2016.11.075.
[20] D. Gupta and R. Rani, “A study of big data evolution and research challenges,” J. Inf. Sci., vol. 45, no. 3, pp. 322–340, Jun. 2019.
[21] M. Al-Mekhlal and A. A. Khwaja, “A synthesis of big data definition and characteristics,” in Proc. IEEE Int. Conf. Comput. Sci. Eng. (CSEE) IEEE Int. Conf. Embedded Ubiquitous Comput. (ECU), Aug. 2019, pp. 314–322, doi: 10.1109/CSE/EUC.2019.00067.
[22] Cisco Global Cloud Index: Forecast and Methodology, 2016–2021, Cisco Syst., San Jose, CA, USA, 2017.
[23] Z. A. Mann and M. F. R. Pedersen, “Big data analytics in MapReduce: Literature review,” in Proc. 1st Int. Conf. Adv. Eng., Manag., Sci. Technol. (ICEMST), Sep. 2021, vol. 3, no. 9, pp. 38–42.
[24] A. Zarei, S. Safari, M. Ahmadi, and F. Mardukhi, “A method of virtual machine placement for fault-tolerant cloud applications,” Intell. Autom. Soft Comput., vol. 22, no. 4, pp. 587–597, Mar. 2016, doi: 10.1016/j.tsc.2016.11.075.
[25] D. Gupta and R. Rani, “A study of big data evolution and research challenges,” J. Inf. Sci., vol. 45, no. 3, pp. 322–340, Jun. 2019.
[26] M. Al-Mekhlal and A. A. Khwaja, “A synthesis of big data definition and characteristics,” in IEEE Int. Conf. Comput. Sci. Eng. (CSEE) IEEE Int. Conf. Embedded Ubiquitous Comput. (ECU), Aug. 2019, pp. 314–322, doi: 10.1109/CSE/EUC.2019.00067.
[27] Cisco Global Cloud Index: Forecast and Methodology, 2016–2021, Cisco Syst., San Jose, CA, USA, 2017.
[28] S. Janani and D. D. F. X. Christopher, “Big data analytics in cloud computing,” Int. J. Comp. Netw. Appl., vol. 29, no. 3, pp. 587–597, Mar. 2016, doi: 10.1007/s13204-017-0229-z.
[29] D. Gupta and R. Rani, “A study of big data evolution and research challenges,” J. Inf. Sci., vol. 45, no. 3, pp. 322–340, Jun. 2019.
A. A. Temghart, D. A. Omar, and M. Marwan, “Optimal virtual machine placement policy for data-intensive applications in cloud computing,” *J. Netw. Comput. Appl.*, vol. 180, Apr. 2021, Art. no. 103025, doi: 10.1016/j.jnca.2021.103025.

T. Duong-Ba, T. Tran, T. Nguyen, and B. Bose, “A dynamic virtual machine placement and migration scheme for data centers,” *IEEE Trans. Services Comput.*, vol. 14, no. 2, pp. 329–341, Mar. 2021, doi: 10.1109/TSC.2018.2817208.

Y. S. Patel and R. Misra, “Distributed algorithm for balanced VM placement for heterogeneous cloud datacenters,” in *Proc. 19th Int. Conf. Distrib. Comput. Netw.*, Jan. 2018, pp. 1–4.

X.-F. Liu, Z.-H. Zhan, J. D. Deng, Y. Li, T. L. Gu, and J. Zhang, “An energy efficient ant colony system for virtual machine placement in cloud computing,” *IEEE Trans. Evol. Comput.*, vol. 22, no. 1, pp. 113–128, Feb. 2018, doi: 10.1109/TEVC.2016.2623803.

Q. Zheng, R. Li, X. Li, N. Shah, J. Zhang, F. Tian, K. M. Chao, and J. Li, “Virtual machine consolidated placement based on multi-objective biogeography-based optimization,” *Future Gener. Comput. Syst.*, vol. 54, pp. 95–122, Jan. 2016, doi: 10.1016/j.future.2015.02.010.

T. Fukunaga, S. Hirahara, and H. Yoshikawa, “Virtual machine placement for minimizing connection cost in data center networks,” *Discrete Optim.*, vol. 26, pp. 183–198, Nov. 2017, doi: 10.1016/j.disopt.2017.05.004.

L. Chen and Z.-H. Liu, “Energy- and locality-efficient multi-job scheduling based on MapReduce for heterogeneous datacenter,” *Service Oriented Comput. Appl.*, vol. 13, no. 4, pp. 297–308, Aug. 2019, doi: 10.1016/j.soca.2017.11.005.

S. M. Seyyedsalehi and M. Khansari, “Analytical comparison of virtual machine and data placement algorithms for big data applications based on cloud computing,” in *Proc. Int. Conf. High-Perform. Comput. Big Data Anal.*, Tehran, Iran, 2019, pp. 143–155.

W. Yao, Z. Guo, and D. Wang, “An energy efficient virtual machine placement algorithm based on graph partitioning in cloud datacenter,” in *Proc. IEEE Int. Symp. Parallel Distrib. Process. Appl. IEEE Int. Conf. Ubiquitous Comput. Commun. (ISPAC/UCC)*, Guanzhou, China, Dec. 2017, pp. 412–416.

J. Zhang, H. Huang, and X. Wang, “Resource provision algorithms in cloud computing: A survey,” *J. Netw. Comput. Appl.*, vol. 64, pp. 23–42, Apr. 2016, doi: 10.1016/j.jnca.2015.12.018.

Z. Usmania and S. Singh, “A survey of virtual machine placement techniques in a cloud datacenter,” in *Proc. Int. Conf. Inf. Secur. Privacy (ICISP)*, Nagpur, India, 2015, pp. 491–498.

A. Chatzipapas, D. Pediaditakis, C. Rotos, V. Mancuso, J. Crowcroft, and A. Moore, “Challenge: Resolving data center power bill disputes: The energy-performance trade-offs of consolidation,” in *Proc. Int. Conf. Future Energy Syst.*, Bengaluru, India, 2015, pp. 89–94, doi: 10.1145/2768510.2770933.

M. Dayarathna, Y. Wen, and R. Fan, “Data center energy consumption modeling: A survey,” *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 732–794, 1st Quart., 2016, doi: 10.1109/COMST.2015.2481183.

B. Pourgehleb, A. A. Anvigh, A. R. Rantim, and B. Mohammadi, “The importance of nature-inspired meta-heuristic algorithms for solving virtual machine consolidation problem in cloud environments,” *Cluster Comput.*, vol. 24, pp. 2673–2699, May 2021, doi: 10.1007/s10586-021-03294-4.

A. Rajagopalan, D. R. Modale, and R. Senthilkumar, “Optimal scheduling of tasks in cloud computing using hybrid firefly-genetic algorithm,” in *Advances in Decision Sciences, Image Processing, Security and Computer Vision (Learning and Analytics in Intelligent Systems)*, vol. 4, Cham, Switzerland: Springer, 2019. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-24318-0_7?_citesear