The Journal of Supercomputing
https://doi.org/10.1007/s11227-020-03497-3

# A CSO-based approach for secure data replication in cloud computing environment

**N. Mansouri[1] · M. M. Javidi[1] · B. Mohammad Hasani Zade[1]**

Accepted: 26 October 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

## Abstract

Cloud computing has a significant impact on information technology solutions for both organizations and researchers. Different users share critical data over the cloud where failures are normal rather than exceptional. Therefore, data fragmentation and data replication algorithms are useful to enhance data security. Three important questions need to be answered carefully: (1) Which files should be replicated; (2) how many appropriate new replicas should be placed; (3) where the new replicas should be stored. In this paper, we propose a CSO-based approach for secure data replication (SDR) that determines suitable data center for new replica by designing a smart fuzzy inference system with four inputs as centrality, energy, storage usage, and load. In addition, a high-quality knowledge base is designed to describe the fuzzy system of CSO algorithm. To obtain a higher level of security, we partition each popular file into several fragments with different sizes based on the ability of data centers. Then, these fragments are stored based on the T-coloring concept to prevent an attacker from determining the locations of the fragments. Consequently, SDR protects the data file without any encryption technique since each data center has a single fragment of a particular file and no meaningful data are achieved in a successful attack. We evaluate the proposed algorithm with CloudSim toolkit, and the experiments show that SDR strategy can reduce the total energy consumption and response time by 31% and 28% (on average) compared to other related algorithms, respectively. In terms of storage usage, effective network usage, hit ratio, mean latency, load variance, number of replications, efficiency, and bandwidth consumption, the obtained results indicate that our strategy outperforms previous replication methods by a significant margin. The main reason is that SDR successfully balances the trade-offs among objectives by the fuzzy system.

**Keywords** Heuristic · Replication · Cloud computing · Security · Fuzzy

✉ N. Mansouri
najme.mansouri@gmail.com; n.mansouri@uk.ac.ir

[1] Department of Computer Science, Shahid Bahonar University of Kerman, Box No. 76135-133, Kerman, Iran

Published online: 13 November 2020

## 1 Introduction

Cloud computing is getting more and more attention as a popular computing paradigm [1]. The first aim of cloud system is to decrease the cost and provide all the useful services for users and help them to focus on their main part of business [2–5]. Memory-aware resource management, energy-aware autonomic resource allocation, effective task scheduling, and data management are common challenges in such an environment [6–8].

There are growing trends in using data-intensive applications that work with large-scale datasets in cloud system. Therefore, a data management strategy is necessary to provide high data availability and efficient accesses in such applications. To alleviate this problem, dynamic data replication is a promising technique, which stores various replicas at different data centers to improve system load balancing [9].

Usually, cloud providers (e.g., Google Cloud Platform) apply encryption service on the server-side and encrypt files of users based on the security level determined by the length of encryption key [10]. But, this schema leads to a lot of delay and overhead due to the encryption process and more effort is needed to manage these keys. In other words, the use of traditional and complex cryptographic methods will lead to an increase in the execution time of the main operations (i.e., placement and retrieval) of data. Therefore, the first goal of this paper is to use an intelligent partitioning technique that not only relieves the system of computationally expensive methodologies but also improves data security. Furthermore, by storing different parts of a particular file in various data centers of the system, the data transfer time can be reduced by using the parallel download technique.

Another important crucial issue of cloud computing is energy consumption [11]. Recent industry reports present that enormous energy is consumed by large-scale data centers since the popularity of data-intensive applications and services is increased. This trend of energy consumption in the future will certainly continue, and hence, the energy problem of data centers will become even serious. However, none of the data replication algorithms simultaneously focus on energy efficiency, security, and execution time inside data centers.

To address these gaps, we present a dynamic data replication algorithm which partitions a popular file into several fragments and stores them on suitable data centers based on centrality, energy, storage usage, and load. To overcome the security problem of fragments placement, it considers a certain distance among the data centers storing the fragments using graph T-coloring technique. Therefore, the proposed algorithm improves data security without any encryption strategy since each data center has only one fragment of the file and even in a successful attack, useful information is not obtained.

In other words, the proposed algorithm simultaneously focuses on security and performance. So an attacker cannot find the precise locations of the fragments. Most of the previous replication algorithms for data centers optimized system bandwidth and data availability between geographically distributed data

centers [12]. The investigation for energy efficiency and replication techniques inside data centers is limited. We note the high proportion of works related to energy efficiency in distributed systems focus only on device- or network-specific solutions. For example, they deactivated some elements such as Low Power Idle and Smart Sleeping or reduced the adaptive rate to save energy with respect to an always-on and over-provisioned network. We can classify data center-oriented methods into two classes. First category methods try to use the minimum set of data centers in the network for job execution. But this idea may cause performance degradation. Methods of the second class utilize all the data centers in the network. When job execution is completed, the data centers are power down to avoid high-energy consumption at idle state. It is necessary to consider energy consumption in the design of job scheduling and data replication methods in addition to device-specific aspects. So, the proposed replication algorithm applies the energy consumption of data center during replica placement.

### 1.1 Contributions

There has been a few research that applies data replication to reduce energy consumption, transfer time as well as achieving data security in a cloud environment. Most of the service providers like Amazon S3 use the security schema and encrypt the file based on the security level, and so significant delay is appeared. We propose a new data replication strategy for cloud computing which offer data security while minimizing the overhead of the security service by considering important factors and concepts. The major contributions can be listed as follows.

- Model the energy consumption of the data center during data replication in the cloud environment.
- Consider the centrality measure to improve retrieval time.
- Assign merit value to all data centers by designing a fuzzy inference system with four inputs as load, storage usage, energy, and centrality. A fuzzy inference system is useful in cases where the control processes and decisions are complicated to analyze by conventional quantitative methods.
- High-quality fuzzy rules are determined by CSO algorithm in the design of flexible fuzzy system.
- Develop a scheme for data partitioning and take into account the data center suitability in fragment size determination.
- The selected data centers are separated based on the T-coloring approach to prevent an attacker from determining the locations of the fragments. Therefore, the proposed algorithm addresses achieve fast data retrieval while guarantees the security of data. Let there are $K$ data centers in cloud, and a file is divided into $n$ fragments. Consider the number of successful intrusions on data center is $m$, such that $m > n$. $P(m,n)$ indicates the probability that $m$ number of victim data centers have all of the $n$ data centers containing the fragments of a file.

$$P(m,n) = \frac{\binom{m}{n}\binom{K-m}{m-n}}{\binom{K}{m}} \quad (1)$$

If $K=30$, $m=10$, and $n=7$, then $P(10,7)=0.0046$. If $K=50$, $m=20$, and $n=15$, then $P(20,15)=0.000046$. We can see that the higher the value of $K$, the less probable that an assailant will achieve the total file. Therefore, the probability for an attacker to achieve a high number of the complete file decreases significantly since cloud environment includes thousands of data centers.

- Performance evaluation of the developed replication strategy in terms of average response time, storage usage, effective network usage, energy consumption, hit ratio, mean latency, load variance, number of replications, efficiency, and bandwidth consumption through CloudSim simulator [13].

The rest of paper is structured as follows. Section 2 consists of related works on data replication of cloud computing systems. Section 3 provides the details of our proposed solution for secure data placement. Section 4 presents a formalization of a cloud system model and parameter setup. Section 5 addresses the simulation environment and performance evaluation. Finally, conclusions and future works are given in Sect. 6.

## 2 Related works

In today's world, new data-intensive applications can benefit if cloud scheduler uses data replication strategies in executing their workflows. In this environment, storing replicas at multiple data centers and then accessing them from the appropriate data center provides efficient data access without a large consumption of bandwidth. Replicating all files in all data centers is not realistic since this solution consumes high bandwidth. Also, data centers have not always enough space to store all these data files. Dealing with these problems, it replicates critical data into the best locations. To achieve reasonable QoS, many researchers have focused on data replication in a cloud environment [14, 15].

Long et al. [16] designed a Multi-objective Optimized Replication Management (MORM) strategy by balancing the trade-offs among the different parameters in a cloud environment. They proposed formula that is a combination of various optimization factors such as data availability, load, latency, service time, energy consumption, probability of failure. Users can set the weight of variables based to the requirements. For example, assigning a higher value on their performance objectives makes the replication strategy adaptable. The experimental results using the extended CloudSim showed that MORM strategy can enhance load balancing of system and reduce energy consumption. But they did not deal with the dynamic variation in file access characteristics. In addition, this work does not include deep application analysis.

Boru et al. [17] investigated data replication issues in a distributed environment and presented a new data replication method that considers traditional performance parameters, such as bandwidth utilization, as well as optimizes energy consumption. The server power consumption is found based on its CPU utilization. It considers idle power consumption, peak load power consumption, and server load in determining energy consumption of computing servers. In addition, they improved the quality of user experience of cloud applications by optimizing communication delays. The proposed strategy evaluated using a mathematical model and GreenCloud simulator, which is focused on energy consumption, and communication processes in data centers [18]. The simulation results demonstrated that the proposed strategy could reduce bandwidth usage and energy consumption due to replicating files in a location close to the requester. The main weakness of this approach is that it ignores the load balancing of system resources. Moreover, the leakage of sensitive data files in case of improper sanitization and malicious data centers is not handled. But, our proposed replication algorithm (SDR) handles the leakage of sensitive files by breaking them and using several nodes to store a single file.

Casas et al. [19] presented a Balanced and file Reuse-Replication Scheduling (BaRRS) algorithm to schedule tasks and replicate necessary files in data centers. BaRRS tries to divide large-scale workflows into small parts. Then, it applies a paralleling approach for enhancing the resource utilization of the cloud environment. In addition, BaRRS decides about task scheduling according to the execution time, dependency patterns, and file size. Finally, BaRRS algorithm replicates files when data transfer time is higher than execution time. For simulation, four scientific workflows with different size and dependency patterns are used. The experiment results indicated that the efficiency of the scheduling algorithm is highly dependent on the workflow dependency patterns. This algorithm does not pay attention to the dynamism of the number of VMs, and so it cannot be used to analyze various scheduling configurations.

Manjul et al. [20] introduced a replication strategy that will help to achieve data privacy using data division. They used the RSA cryptographic technique for improving security. Ron Rivest, Adi Shamir, and Leonard Adleman (RSA) designed the Public key strategy in 1997. In this regard, it is asymmetric that the public key is sent to those who want to encrypt the message. By using the private key, encryption is performed and therefore not shared by anyone. They encrypted file based on RSA and divided into several different blocks and then stored on multiple locations. Cloud manager cannot process the data, so the confidentiality of data is enhanced. When the part of data is revealed, the hacker cannot know what data it contain as it is encrypted. In addition, they used the automatic update technique without downloading the file again. Experiment results showed that the proposed strategy increases security in various levels. While our approach (SDR) is not dependent on traditional cryptographic methods to provide data security. But they did not store parts of data on appropriate cloud service providers to enhance data security.

Nivetha et al. [21] presented an Efficient Fuzzy Replication (EFR) algorithm to enhance data availability in cloud data centers. The EFR algorithm consists of three main steps. In the first step, EFR stores the replica according to the data availability. The second step includes fuzzy-based replica selection to determine the optimum

number of replicas. The third step replaces of replica to provide enough space for new replicas. Neglecting the energy consumption issue in data center is the main disadvantage of the EFR strategy. Moreover, our proposed replication algorithm (SDR) focuses on energy consumption within the cloud computing that is not considered in [21].

Tos et al. [22] presented a Performance and Profit-Oriented Data Replication (PEPR) algorithm to consider the performance of system and provider benefits in cloud environment. PEPR checks the benefit of providers when the estimated execution time is greater than the guaranteed execution time in the contract. Then, it replicates new replicas of files when the profitability is greater than the minimum threshold. It considers the number of file requests during the replica placement process. Results of the experimental evaluation showed PRPR could satisfy the contract commitment from both the customer and the supplier. But PRPR did not pay attention to requests that consist of dependent operations like join operations. Because this approach only assumes a fixed number of resources for each service, it cannot provide a guarantee for the optimal use of resources. On the other hand, consumers also have no information on how to improve efficiency within a budget when choosing a type of service.

Mansouri et al. [23] introduced a Dynamic Popularity-aware Replication Strategy (DPRS) to reduce access time in the cloud system. At regular intervals, DPRS calculates the popularity value for each file. Then, it replicates the most popular files in a suitable data center based on storage usage, number of accesses, and data center location. Results of simulation demonstrated that DPRS could improve response time. But they did not mention the energy consumption problems.

Li et al. [24] presented an effective cost-oriented replica selection strategy named EDR (Energy-Aware Distributed Running system), with two distributed optimization techniques. In the first part, EDR considers the energy cost model for cloud data center. The second part formulates the replica selection issue using a convex optimization problem, which optimizes the total energy consumption of data centers based on the bandwidth utilization and network latency of each data center. Simulation results confirmed that EDR could reduce 12% energy consumption for data-intensive applications such as online video streaming. The limitation of EDR algorithm is that it considers only the access cost in the main decision. This work did not protect the important information against problems such as tempering and loss of data resulting from virtualization and multi-tenancy.

Limam et al. [25] introduced a data replication that tries to provide a balance among availability, performance tenant requirements, and provider profit. The authors defined the multi-tier structure that is based on the bandwidth network-level locality. The proposed replication algorithm is triggered when the estimated response time exceeds a threshold (*RespT*). For calculating the number of replicas, it stores new replicas as long as the SLA is not satisfied. In other words, if the profit of provider is small and QoS of queries is guaranteed, then the creation of replicas is stopped. The proposed strategy takes into account network usage, CPU usage, storage usage, and the payment of probable penalties for provider profit estimation. In the replica replacement phase, it finds a set of queries that the estimated response time for them is lower than *RespT* and then deletes replicas of data that are needed

by this set. The results of experiments indicated that the proposed replication algorithm can reduce bandwidth consumption and response time. Since it categorizes the queries based on their region and then submits these queries to data centers in the same region, nevertheless, the scheme focuses only on performance and provider benefit. Moreover, the SDR strategy considers the security of the files that are not considered in [25].

Mansouri and Javidi [26] presented a new Prefetching-aware Data Replication (PDR) to reduce the number of communications in cloud. PDR tries to find the most related files based on access history. It defines a dependency graph for each task on a data center and then determines the most popular group of files. For the replacement step, it uses a fuzzy system with four inputs as number of accesses, replica cost, access time, and availability of data. Simulation results with CloudSim demonstrated that PDR can improve the hit ratio. But, creating several copies for a file in different data centers increases the attack surface for that particular file. Hence, we propose SDR strategy to divide a file into several fragments.

Liang et al. [27] presented a data protection method that replicates sensitive files in various virtual machines to improve data survivability. The proposed strategy assumes that user's and attacker's virtual machines are performed in different data centers randomly. The proposed approach tries to provide a balance between data survivability and data security by optimizing data replication. It selects one level of available data protection and the corresponding data replication strategy to guarantee the requirement data survivability and security with minimum cost. Numerical examples showed the impacts of various model parameters on data security. But it did not protect data from unauthorized access by data partition techniques.

Sun et al. [28] indicated a dynamic adaptive replica strategy (DARS) to obtain high load balance in a cloud environment. DARS introduces the overheating similarity concept to show a probability that a data center changes into an overloaded data center. Then, it uses a fuzzy membership function for determining the opportune moment based on the overheating similarity. The membership functions are defined based on the load of data center, and DARS obtains a similarity interval for relieving data center overload. DARS stores replicas in a data center before overload, and so it can effectively decrease the probability of the data center overload. On the other hand, placing replicas in a data center that has the lowest overheating similarity value may increase access delay. Therefore, DARS considers the file access probability in replica placement. But the file popularity is not addressed in this work.

He et al. [29] presented a new data replication (DPRS) to predict the future access of each file and determine a suitable number of replicas by a prediction method and access history. DPRS applies an exponential smoothing approach in predicting phase to minimize the number of replica creation and deletion. In addition, it defines a threshold to ensure that the response time is not too high. It decides about the number of replicas with three operations: creation, deletion, and no action based on the numbers of requests in the current period of time and next period. Simulation results with CloudSim proved that DPRS can reduce response time and storage cost. In [29], only the execution time reduction is cared about, while in real large-scale systems, security is a critical issue. Therefore, the proposed strategy (SDR) considers the issue of security and performance simultaneously.

Table 1 summarizes the reviewed replication algorithms based on various features:

- Bandwidth consumption tells whether the bandwidth factor is used as the main parameter in the replication process.
- Response time shows whether the authors have focused on the response time in the proposed replication algorithm.
- Load balancing specifies whether the proposed strategy considered balancing the system load globally.
- Storage assumption specifies whether the size of storage is limited or unlimited.
- Energy efficiency represents whether the proposed strategy modeled energy consumption of cloud data centers.
- Security describes whether the authors have combined the proposed replication strategy with the security technique.
- Fuzzy determines whether the proposed replication applied the fuzzy theory.
- Simulator option refers to the toolkit/environment that is used for performance evaluation.

From the analyzed works [14–29], it seems that there is no single dynamic replication algorithm that overcomes all data replication challenges. Execution time is mostly the center of attention in the literature for data replication process. Furthermore, in [30] security and scalability factors have been enhanced, while some totally do not concentrate on these parameters. Some replication methods have main attention to the bandwidth and storage consumption, while some strategies have wasted more bandwidth than average [31]. So, proposing a comprehensive replication algorithm based on the main performance parameters in the cloud is essential.

As increased datasets the data placement and data security are very critical problems in the distributed system to overcome from these, cloud computing comes in front. Traditional strategies have done that such as reducing response time, improving network usage, they did not consider security factors. Outsourcing in cloud environment leads to increase the importance of security parameter in the QoS. Since the data files may be revealed by attackers, this may affect the cloud performance such as increasing the data retrieval time. While guaranteeing the security of data, it is necessary to consider the data retrieval time.

In this paper, we try to solve both problems by storing fragments of file in the best cloud data centers based on the energy consumption, load, and centrality. In the cloud environment, the replication process (i.e., finding the most appropriate place for a strong new replica) must consider many criteria that may conflict with one another and information is uncertain, rendering decision-making processes complex. Therefore, we use CSO-based fuzzy inference system to address such problems. The proposed strategy divides each file into different fragments in such a way that it is impossible to find a complete file in one try. Also, we consider the certain distance between the data centers that stored the fragmentations of a particular file based on the T-coloring, since the illegal users cannot track the next fragment location.

**Table 1** Summary of the analyzed replication strategies in cloud

| Mechanisms | Main idea | Bandwidth consumption | Response time | Load balancing | Storage assumption | Energy efficiency | Security | Fuzzy | Simulator |
|---|---|---|---|---|---|---|---|---|---|
| Long et al. [16] | Modeling multi-objective optimized replication management | Yes | Yes | Yes | Yes | Yes | No | No | CloudSim |
| Boru et al. [17] | Modeling of energy consumption for data center IT infrastructures | Yes | Yes | No | Yes | Yes | No | No | GreenCloud |
| Casas et al. [19] | Analyzing workflow features | Yes | Yes | Yes | Yes | No | No | No | Real environment |
| Manjul et al. [20] | Using the RSA cryptographic algorithm | No | Yes | Yes | Yes | No | Yes | No | JAVA and CloudME |
| Nivetha et al. [21] | Modeling fuzzy-based replication | Yes | Yes | No | Yes | No | No | Yes | Cloud Analyst |
| Tos et al. [22] | Estimating the response time | Yes | Yes | Yes | Yes | No | No | No | CloudSim |
| Mansouri et al. [23] | Considering the access popularity | Yes | Yes | No | Yes | No | No | No | CloudSim |
| Li et al. [24] | Considering the energy cost in replica selection | Yes | Yes | Yes | No | Yes | No | No | Unknown Simulator |
| Limam et al. [25] | Modeling tenant budget requirements | Yes | Yes | Yes | Yes | No | No | No | CloudSim |
| Mansouri and Javidi [26] | Pre-replicating popular files | Yes | Yes | No | Yes | No | No | Yes | CloudSim |
| Liang et al. [27] | Addressing co-residence attacks | No | No | No | Yes | No | Yes | No | Numerical examples |
| Sun et al. [28] | Using self-adaptive manner | Yes | Yes | Yes | Yes | No | No | No | OptorSim |
| He et al. [29] | Proposing future accesses prediction method | Yes | Yes | No | Yes | No | No | No | CloudSim |

## 3 Secure dynamic replication (SDR)

SDR is a heuristic and a security-based replication method that has three main steps as follows:

(1) *Determination of popular file* The most needed files are selected for replication.
(2) *Replica placement and optimize fuzzy rules with Competitive Swarm Optimizer (CSO)* The popular file is divided into some fragments, and then, the suitable locations based on the T-coloring concept and CSO technique for storing fragments are determined.
(3) *Replica replacement* Due to the limited storage space, less valuable replicas are determined for deletion.

### 3.1 Determination of popular file

One of the serious problems in cloud with high-speed growth of data is placing replicas of the whole files in storage with bounded capacity. Therefore, it is necessary to find the most needed files in the replication process. The "popularity" of a file is defined based on its access rate by the users. The proposed strategy lists files based on popularity value in decreasing order. The top 20% of frequently requested data files are selected according to the 80/20 concept. The reason why 20% is described later.

### 3.2 Replica placement and optimize fuzzy rules with CSO

In this section, firstly a description of CSO algorithm performance is presented, and then the replica placement problems and optimize fuzzy rules have been described in detail.

#### 3.2.1 The canonical PSO algorithm

One comprehensive tool for solving hard optimization problems with affordable time and cost is heuristic methods [32, 33]. Kennedy and Eberhart introduced canonical PSO algorithm for solving different optimization problems based on emulating social swarm behaviors of animals such as bird flocking [34]. Each individual in the PSO swarm has a position and velocity, and the strategy searches each case to determine the best outcome according to the current swarm. Then the whole swarm moves to the potentially better solution. The new velocity vector and position are determined by the following formulas:

$$v_i^{t+1} = wv_i^t + \phi_1 R_1^t(\hat{g}^t - x_i^t) + \phi_2 R_2^t(\hat{x}_i^t - x_i^t) \tag{2}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{3}$$

where $t$ is the number of iterations, $x_i^t$ is the position of the $i$th particle for the $t$th iteration, $v_i^t$ is the velocity of the $i$th particle, $w$ shows inertia weight, $\phi_1$ and

$\phi_2$ denote acceleration constants, $R_1^t$ and $R_2^t$ indicate two random vectors, and each entry takes the values between 0 and 1. Although PSO algorithm has shown many successes over the last few years, its performance is falling on complex search space or high-dimensional optimization problems [35]. To improve this algorithm, various versions have been presented, including chaos immune particle swarm optimization [36], the new topological structure-based variants [37] and the hybridization-based variants [38].

### 3.2.2 The CSO algorithm

Cheng and Jin introduced [35] CSO algorithm for large-scale optimization. In each iteration, CSO randomly divides the swarm into two partitions and each pairwise competition is performed between the particles from each partition. Then each particle that wins the competition goes to the next iteration, and the losing particle will update its position and velocity based on the winner particle. Therefore we have [35]:

$$v_i^{t+1} = R_l^t v_l^t + R_2^t(x_w^t - x_l^t) + \phi R_3^t(\overline{x}^t - x_l^t) \tag{4}$$

$$x_l^{t+1} = x_l^t + v_l^{t+1} \tag{5}$$

where $t$ indicates the number of iterations, $R_1^t$, $R_2^t$, and $R_3^t$ denote three numbers that are randomly generated between 0 and 1. $x_w^t$ and $x_l^t$ are the winner particle and the loser particle, respectively. $\overline{x}^t$ is the average position of swarm for $t$-th iteration, and $\phi$ is variable for controlling the influence of $\overline{x}^t$. In summary, particles in CSO learn based on randomly selected competitors instead of personal best or global position.

### 3.2.3 Problems description

Now, we intend to solve three problems as follows:

- Determine appropriate data centers for store replica of popular files,
- Determine appropriate rules for the fuzzy evaluator system,
- Place replica of popular files in appropriate data centers in a secure way.

The first two problems have been solved with a meta-heuristic method called CSO, and the third one is solved with the T-coloring technique. The details of problem-solving methods are presented in the following subsections.

### 3.2.4 Determine appropriate data centers and fuzzy rules with CSO

In this subsection, the steps of CSO for determining appropriate data centers to store replica files and appropriate rules for the fuzzy evaluator system are explained in detail.

*Individual representation* In this paper, a CSO-based method is applied to find a fuzzy system with an appropriate number of fuzzy rules and identify appropriate data centers for placing a replica of popular files. In the CSO-based method, each

individual has two parts. The first part keeps merit values of data centers for placing popular file $F_i$. The second part represents rules information ($r_i$) and a vector which is responsible to suggest whose rules are selected ($g_i$), these two parts are updated in several iterations to determine the most appropriate rules and data centers.

The individual $P_h$ contains three vectors as $d_h$, $r_h$ and $g_h$(i.e., $P_h = [d_h, r_h, g_h]$). Vector $d_h$ is $N$-dimensional vector, and $i$-th dimension of this vector keeps the merit value of $i$-th data center. At first, each dimension of $d_h$ vector initializes with 0 ($d_h^i = [0]\,,\ i = 1, 2, 3, ..., N;\quad N = number\ of\ datacenter$).

Vector $r_h = [r_1^h, r_2^h, ..., r_j^h, ..., r_B^h]$ contains of the premise and consequent parts of the candidate fuzzy rules, where $B$ represents a positive number introduced by the user to determine the maximum number of rules in the rule base that is proposed by the individual $P_h$. Assume $r_j^h = [m_{j1}^h\ m_{j2}^h\ \cdots\ m_{ji}^h\ \cdots\ m_{jM}^h]$ is a vector that defines the membership functions for the $j$th rule and $m_{ji}^h = [1, 3]$ shows a vector for introducing the fuzzy set variables of the $i$th input (Low = 1, Medium = 2, High = 3) and finally $m_{jM}^h$ is output fuzzy set variable for $j$-th fuzzy rule.

Consider vector $g_h = [g_1^h\ g_2^h\ \cdots\ g_j^h\ \cdots\ g_B^h]$ that selects the fuzzy rules from the candidate rules $r_h = [r_1^h, r_2^h, ..., r_j^h, ..., r_B^h]$ to generate fuzzy rule base, where $g_j^h \in [0, 1]$ and decides whether the $j$th candidate rule ($r_j^h$) is inserted to the rule base. It adds the $j$th candidate rule ($r_j^h$) to the rule base, if $g_j^h \geq 0.5$. In other words, the total number of fuzzy rules in rule base is equal to the total numbers of $g_j^h (j = 1, 2, ..., B)$ whose values are greater than or equal to 0.5.

$$P_h = [d_h, r_h, g_h] \tag{6}$$

where $r_h = [m_{(1,1)}^h,\ m_{(1,2)}^h, ... m_{(1,M)}^h, ....., m_{(B,1)}^h,\ m_{(B,2)}^h, ... m_{(B,M)}^h]$ and $g_h = [g_1^h,\ g_2^h, ..., g_B^h]$.

Assume that we have $M$ evaluation metrics and the number of maximum rules is $B$, then the fuzzy value of the $k$th metric in the $j$th rule is determined with $m_{(j,k)}^h$ and generated as follows:

$$m_{(j,k)}^h = rand([1, 3]) \tag{7}$$

where function *rand* generates a random number between 1 and 3.

The determinant parameter for selecting the $j$th rule and data centers' merits is presented with $g_j^h$ and $d_h$, respectively, and they are generated as follows:

$$g_j^h = rand([0, 1]) \tag{8}$$

$$d_h = [d_1^h, d_2^h, d_3^h, ....., d_N^h] = [0, 0, 0, ....., 0] \tag{9}$$

where $N$ shows the total number of data centers.

The initial velocity vector ($V_h$) for the $h$th individual is expressed as follows [35]:

$$V_h = [\lambda_h, \chi_h] \tag{10}$$

where $\lambda_h$ shows the velocity of fuzzy set part and is expressed as $\lambda_h = [\lambda^h_{(1,1)}, \lambda^h_{(1,2)}, \ldots \lambda^h_{(1,M)}, \ldots\ldots, \lambda^h_{(B,1)}, \lambda^h_{(B,2)}, \ldots \lambda^h_{(B,M)}]$. $\chi_h$ indicates determinant parameter part and is expressed as $\chi_h = [\chi^h_1, \chi^h_2, \ldots, \chi^h_B]$.

Velocity for $k$th metric in the $j$th rule shows with $\lambda^h_{(j,k)}$ and is generated as follows:

$$\lambda^h_{(j,k)} = rand([1, T]) \tag{11}$$

where $T$ is any positive integer number.

The velocity of the $j$th determinant parameter shows with $\chi^h_j$ and is generated as follows:

$$\chi^h_j = rand([0, 1]) \tag{12}$$

Figure 1 shows an example of a population with 10 individuals for a system with 5 data centers, 5 maximum number of rules, 2 inputs (i.e., two evaluation metrics) and one output for a fuzzy system. Each input and output variables have 3 fuzzy sets (i.e., low, medium, high).

Algorithms 1–4 show how individuals are generated. Algorithm 5 indicates how the velocity vector is generated for individuals.

**Algorithm 1. Initialize_individuals ()**

```
Input: Number of individuals (L), Maximum number of rules (B), Number of datacenters (N), T, q
Output: Population P
Struct P
{
  d,
  r,
  g,
  fit
}
Struct r
{
  id,
  fuzzy_sets
}
for (i=1; i<=L; i++)
{
      P[i]. d= Build_merit_array (N);  //Create merit array for i-th individual
      P[i]. r= Build_fuzzy_set_array (B, T);  //Initial rules for i-th individual
      P[i]. g= Build_selection_array (B);  //Create an array for selection rules that are generated by i-th individual
      P[i]. fit= 0;
} //end for
```

| Individual number | d | | | | | r | | | | | | | | | | | | | | g | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | id | | | | | | | | | | | | | | | | | | |
| | DC [1] | DC [2] | DC [3] | DC [4] | DC [5] | 1 | | | 2 | | | 3 | | | 4 | | | 5 | | | | | | |
| | | | | | | Fuzzy set | | | | | | | | | | | | | | | | | | |
| 1 | 0 | 0 | 0 | 0 | 0 | 1.4 | 0.3 | 1.1 | 2.7 | 2.1 | 2.4 | 1.8 | 0.41 | 0.36 | 0.4 | 1.7 | 1.33 | 2.3 | 1.7 | 1.5 | 0.55 | 0.46 | 0.73 | 0.1 | 0.15 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0.5 | 1.8 | 2.4 | 0.4 | 0.8 | 1.6 | 2.35 | 1.6 | 0.9 | 2.2 | 1.4 | 2.98 | 2.2 | 2.1 | 2.7 | 0.4 | 0.90 | 0.56 | 0.77 | 0.83 |
| . . | . . | . . | . . | . . | . . | . | | | | | | | | | | | | | | | . | | | | |
| 10 | 0 | 0 | 0 | 0 | 0 | 2.4 | 0.98 | 0.56 | 1.9 | 1.5 | 2.06 | 2.76 | 0.22 | 0.76 | 1.6 | 2.9 | 2.2 | 0.3 | 1.1 | 1.8 | 0.33 | 0.85 | 0.44 | 0.69 | 0.66 |

**Fig. 1** Population example

**Algorithm 2. Build_merit_array ()**

```
Input: N
Output: d
for (i=1; i<=N; i++)
{
   d[i]= 0;
}
```

**Algorithm 3. Build_fuzzy_set_array ()**

```
Input: B, T, q
Output: Struct r
for (i=1; i<=B; i++)
{
      r[i]. id=i;
      for (j=1; j<= q+1; j++)
        {
            r[i]. fuzz_sets [j]= rand ([1, T]);
        }
}
```

**Algorithm 4. Build_selection_array ()**

```
Input: B
Output: g
for (i=1; i<=B; i++)
{
    g[i]= rand ([1, 0]);
}
```

**Algorithm 5. Initialize_velocity ()**

```
Input: L, B, q
Output: Struct V { λ, χ }
for (i=1; i<=L; i++)
 {
   for (j=1; j<=B; j++)
    {
        V[i]. χ[j] = rand([0,1];
    }
   for (t=1; t<= q+1; t++)
    {
        V[i]. λ[t]= rand ([1, T]);  //T is any positive integer number
    }
 }
```

*Evaluation metrics* The various and conflict factors (i.e., load, storage usage, energy, and centrality) are considered to evaluate individuals in a replica placement decision.

- *Load*

If the data center is overloaded, then a lower value is assigned to this data center. Since this data center does not have a good performance in reading and writing operations with high load, the load is determined using the access rate and service time [18]. Therefore, *Load* (*i, j*) of $f_i$ which is on the data center $S_j$ is calculated according to Eq. (13):

$$Load(i,j) = Acc(i,j) \times ST(i,j) \tag{13}$$

where *Acc(i,j)* indicates access rate for data center $S_j$ that requests file $f_i$. Also *ST (i, j)* shows the expected service time of file $f_i$ in data center $S_j$ ($1 \le j \le m$), It can be determined by Eq. (14):

$$ST(i,j) = v(i,j) \times \frac{size_i}{tp_j} \tag{14}$$

where *v(i, j)* has value 1 if the file $f_i$ is available in data center $S_j$; otherwise, it has value 0, $size_i$ indicates the size of file $f_i$ and transfer rate of data center $S_j$ is indicated by $tp_j$.

Now load of data center $S_j$ is defined as [16]:

$$Load_j = \sum_{i=1}^{n} Load(i,j) \tag{15}$$

where $n$ is total number of files.

- *Storage Usage*

It is obvious that storing replicas in a data center with low storage usage can reduce the waiting time.

- *Energy*

We know that the replication advantages do not come without some drawbacks. One of the main challenges is that creating numerous replicas across the network may lead to high-energy consumption. In these systems, large storage space requires to keep necessary replicas to improve latencies. Besides, most of the storage and computing elements of such systems should be active during processing, in order to provide the necessary services and allow reliable access to data. Therefore, forcing them into low power states for improving energy consumption may not always be a reasonable solution. It is because they usually process and transfer most of their operation time. The total energy consumption can be determined using renewable energy consumption (RE) and cooling energy consumption (CE) [39]. Previous researches have shown that the server power consumption can be calculated using a linear relationship between power consumption and utilization [40]. The energy consumption of computing equipment in data center $S_j$ is shown by *Energy (j)* [39].

$$Energy_j = \frac{E_j}{Q} \tag{16}$$

where $E_j$ is obtained by Eq. (17) [29].

$$E_j = \sum_{i=1}^{n} v(i,j) \times R(i,j) \times u \times \left(Pw_{\max}(j) - Pw_{idle}(j)\right) + Pw_{idle}(j) \tag{17}$$

where $Pw_{max}(j)$ shows the maximum power consumption of data center $S_j$ at the peak load, and $Pw_{idle}(j)$ shows the power consumption of data center $S_j$ at idle state. $R(i,j)$ shows the number of requests coming from data center $S_j$ asking for file $f_i$. $u$ is CPU utilization per request.

The consumed energy is then converted to heat. Let the coefficient of performance (COP) of data center $S_j$ is indicated by $Q$. The high value of COP shows that process of thermodynamic is more efficient in the cooling system of $S_j$. COP depends on the indoor and outdoor temperature ($\text{Tem}_{\text{in}}$ and $\text{Tem}_{\text{out}}$) based on the thermodynamic concept. $Q$ is given by the following [41]:

$$Q = \frac{1}{\frac{\text{Tem}_{\text{out}}}{\text{Tem}_{\text{in}}} - 1} \tag{18}$$

- *Centrality*

On the basis of the centrality of a node, it is possible to determine the importance of it in the system [42]. SDR strategy considers the centrality to reduce retrieval

time. There is a wide variety of criteria for determining centrality. The most important ones are closeness centrality, degree centrality, betweenness centrality, and eccentricity centrality. We only consider the closeness metric in the replica placement process. If a data center has the lowest total distances from all of the other data centers in the system, then it is introduced as a closeness node.

The closeness centrality for data center $x$ can be obtained by Eq. (19).

$$Centrality(x) = \frac{N-1}{\sum_{x \neq y} d(x,y)} \tag{19}$$

The parameter $N$ is used to indicate the total number of data centers in the system, and $d(x,y)$ shows the distance between data center $x$ and data center $y$. Figure 2 shows example of centrality calculation.

*Individuals' evaluation* After individuals are generated (like Fig. 1), we have to calculate the fitness value of each individual with a fuzzy rule-based system. System knowledge and the interaction between variables can be defined by a fuzzy rule-based system. Due to the inherent dynamic structure and the typical complex search spaces of the cloud environment, the design of the automated fuzzy system is beyond the interest of researchers. The main objective for a designing fuzzy inference system in this part of the proposed algorithm is assigning *Merit* to each data center according to the *Load*, *Storage Usage*, *Centrality,* and *Energy.* It can manage and tune several parameters easily and carefully.

SDR algorithm computes the *Merit* of all data centers for storing poplar file $F_i$ by a fuzzy system. Then, it determines the most appropriate data center by considering security and performance. The procedure of placing popular file $F_i$ is explained in the following steps:

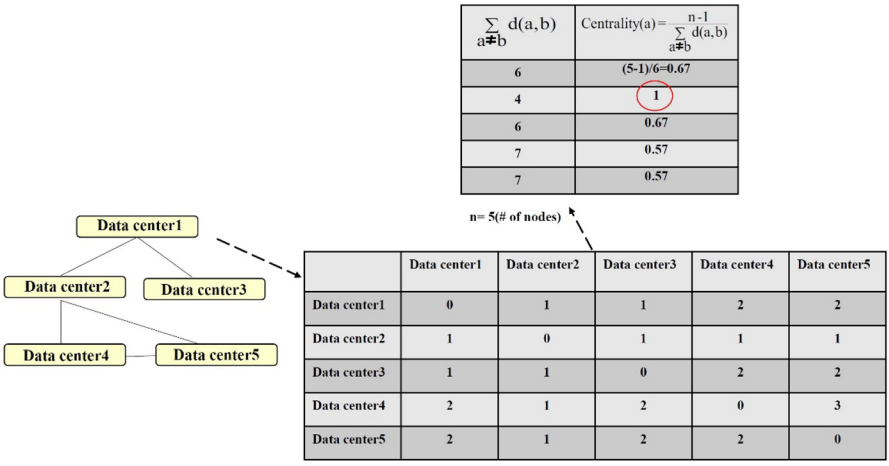


**Fig. 2** Example of centrality calculation

1. For each data center, *load*, *storage usage, centrality,* and *energy* are calculated.
2. Normalize all parameters between 0 and 1. (See Algorithm 12 in appendix).
3. To calculate the *Merit* of each data center for placing $F_i$, a fuzzy inference system will be designed. The proposed fuzzy system considers the values of four parameters (i.e., *load*, *storage usage, centrality,* and *energy*) as inputs and *Merit* as an output parameter. Figure 3 indicates the structure of the fuzzy system that obtains the *Merit* of the data center.

We use Mamdani model that is one of the most useful models to make a fuzzy inference system. It obtains the fuzzified inputs with membership functions. Then, it establishes the rule strength by a combination of the fuzzified inputs based on the fuzzy rules. Now, it combines all the consequents to get output. Finally, a crisp output is determined by using defuzzification methods such as center of area. Figure 4 describes the membership functions for parameters of fuzzy system that are obtained using the fuzzy logic toolbox of MATLAB.

All parameters have three membership functions: low, medium and high. For example, in Fig. 4 the *Load* 0.4 indicates membership degree 0.03 in the *High* interval, 0.86 in the *Medium* interval and 0.2 in the *Low* interval. The designed fuzzy system has 64 rules, and some of them are presented in Table 2.

Algorithm 6 shows how to calculate this fitness.

**Algorithm 6. Fitness calculation**

**Input**: Population $P$ with $L$ individuals
**Output**: $d$ *//Merit of datacenters*, $fit$ *//Fitness of individuals*
for (j=1; j<=$L$; j++) *//L is total number of individual*
{
  for (i=1; i<=$N$; j++) *//L is total number of individuals*
  {
    $RB_j$ =Build_rule_base ([$r_j$, $g_j$]); *//Build a rule base based on information in j-th individual*
    [$d_i$, $fit_j$]= Calculate_merit ($RB_j$); *//Calculate a merit for i-th data center and at the same time calculate j-th individuals' fitness*
  }
}

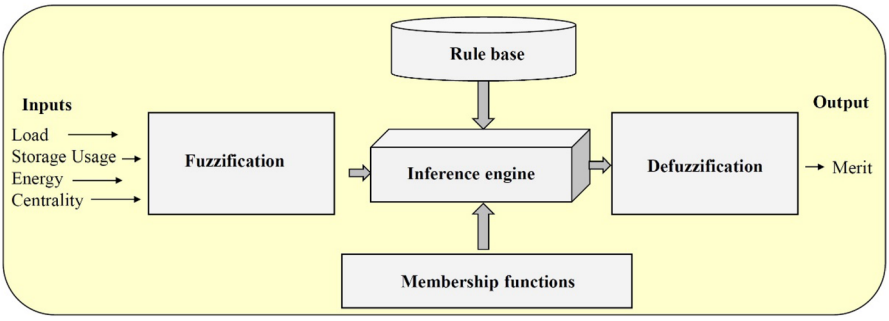


**Fig. 3** The proposed fuzzy inference system

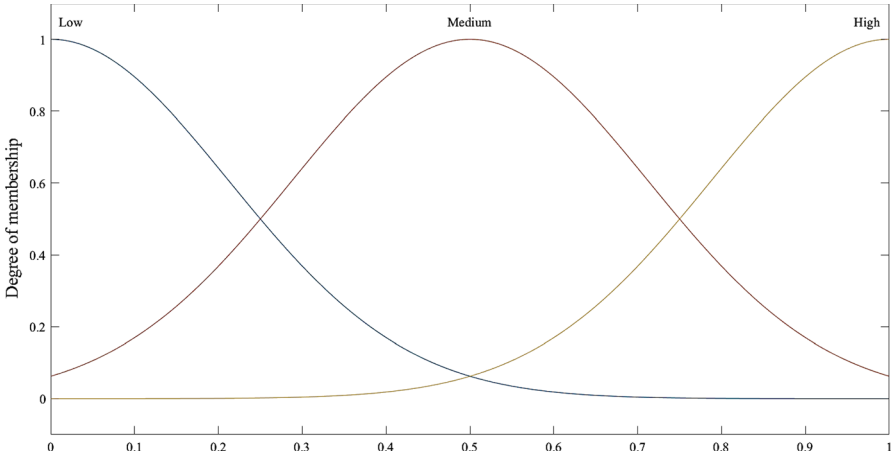


**Fig. 4** Membership functions

**Table 2** Fuzzy rules of controller system for suitability evaluation

| Load | Storage usage | Energy | Centrality | Merit |
|---|---|---|---|---|
| Low | Low | Low | High | High |
| High | High | Low | Medium | Low |
| Medium | Low | Medium | High | Medium |
| Low | Medium | Low | High | High |
| Medium | High | High | Low | Low |
| Medium | Medium | Medium | Medium | Medium |
| High | High | High | Low | Low |
| Low | Low | High | High | Medium |
| High | Medium | Medium | Medium | Low |
| Medium | High | Low | High | Low |
| Low | Low | Medium | Medium | High |
| High | High | High | High | Low |

In each individual, the merit of each data center is evaluated and at the same time the appropriate rules for each data center are generated. The fitness value will guide individuals into two directions, finding an appropriate number of rules for each data center and attempts to achieve data centers with high merit value for storing a replica of popular files. The procedure is described in the following steps.

Figure 5 shows an example of calculating the merit of data centers and fitness for the *i*th individual.

From Fig. 5, at first $g_i$ is used to identify the rules of fuzzy inference system and the evaluation metrics are calculated. Algorithm 7 shows the process of calculating merit for data centers.

**Algorithm 7. Calculate_merit ()**

```
Input: N //Number of datacenters, DC //Set of datacenters, F //Set of files
Output: d //Merit of datacenters
 for (i=1; i<=N; i++)
 {
     Load= Calculate_ Load (DC[i], F. size);
     SU= Calculate_storage_usage (DC[i]);
     Energy= Calculate_energy (DC[i]);
     Centrality= Calculate_centerality (DC);
    d[i]= Calculate_merit_Fuzzy (Load, SU, Energy, Centrality);
 }
```

For example, we want to determine the merit value of data center 1, for placing a replica of the popular file $F_1$, in a system with 5 data centers. Assume that $r_h$ and $g_h$ are denoted as, $[r_1, r_2, r_3, r_4, r_5, r_6, r_7]$ and [0.64, 0.13, 0.54, 0.48, 0.33, 0.85, 0.77], respectively. According to $g_h$, the generated rule base has four fuzzy rules and we have $\{I_1^h, I_2^h, I_3^h, I_4^h\} = \{1, 3, 6, 7\}$. Assume that based on generated rule base, vector $d_h = [0.73, 0, 0, 0, 0]$ is obtained. It means a rule base generated by individual $h$ with the first, third, sixth and seventh rules obtained merit value 0.73 for data center 1.

*Update individuals* After the fitness of each individual is evaluated, the current individuals are updated so that more places in search space can be explored. In this regard, two different individuals from population $P$ randomly are selected. Then, the winner and loser particle between these two chosen individuals are determined. The winner directly goes to the next iteration, but the loser one must be updated and replaced with a new individual. Thus, its velocity and position must be updated. The velocity of loser particle shows with $V_{loser}$ and includes $\lambda_{Loser}$ and $\chi_{Loser}$.

The velocity vectors are updated with Eqs. (20–23):

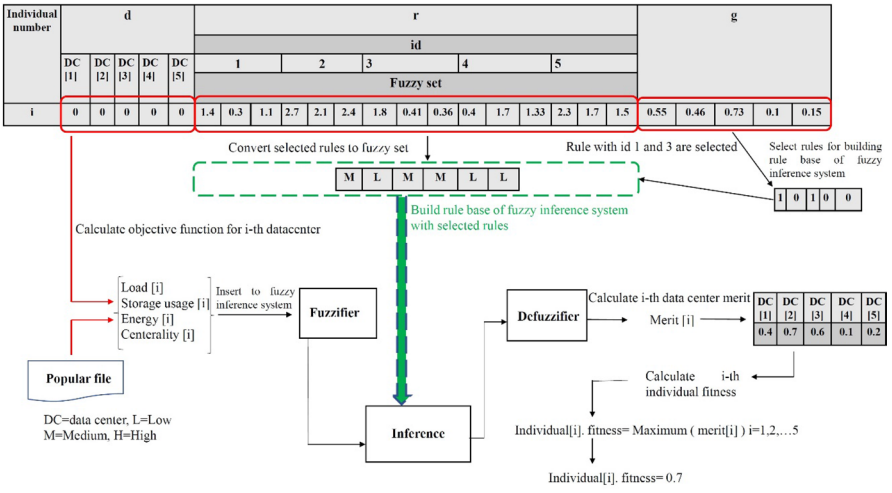


**Fig. 5** An example for fitness calculation process

$$new\_\chi_{Loser} = R_1 \times \chi_{Loser} + R_2(g_{winner} - g_{Loser}) + \emptyset \times R_3(g_{mean} - g_{Loser}) \quad (20)$$

$$new\_g_{Loser} = g_{Loser} + new\_\chi_{Loser} \quad (21)$$

$$new\_\lambda_{Loser} = R_1 \times \lambda_{Loser} + R_2(r_{winner} - r_{Loser}) + \emptyset \times R_3(r_{mean} - r_{Loser}) \quad (22)$$

$$new\_r_{Loser} = r_{Loser} + new\_\lambda_{Loser} \quad (23)$$

The process of updating velocity vector, $g$ and $r$ vector for loser individual is represented in Algorithm 8 and Algorithm 9, respectively. Also, an example for updating these vectors can be seen in Fig. 6.

**Algorithm 8. Update_velocity ()**

**Input:** $P_{loser}, V_{Loser}, P_{winner}, R_1^t, R_2^t, R_3^t, \phi$,

**Output:** $new\ \_V_{Loser}$

$new\ \_\chi_{loser} = R_1 \times V_{loser}.\chi + R_2(P_{winner}.g - P_{Loser}.g) + \varnothing \times R_3(g_{mean} - P_{Loser}.g)$;

$new\ \_\lambda_{loser} = R_1 \times V_{Loser}.\lambda + R_2(P_{winner}.r - P_{Loser}.r) + \varnothing \times R_3(r_{mean} - P_{Loser}.r)$;

$new\ \_V_{Loser} = [new\ \_\lambda_{loser}, new\ \_\chi_{loser}]$

return $new\ \_V_{Loser}$

**Algorithm 9. Update_position**

**Input:** Loser individual ($P_{Loser}$), Updated velocity ($new\ \_V_{Loser}$);

**Output**: Updated individual ($new\_P_{Loser}$)

$new\ \_P_{Loser}.g = P_{Loser}.g + new\ \_\chi_{loser}$;

$new\ \_P_{Loser}.r = P_{Loser}.r + new\ \_\lambda_{loser}$;

return $new\_P_{Loser}$

The new population $P$ for the next iteration includes winners' particle and the updated loser particles. This update process continued until the condition (number of iteration) is not met; then, the best individual is returned.

The best individual contains the highest value of merit for data centers, and the proposed method supposes that the replica of the popular file $F_i$ should be placed on that data center. Algorithm 10 shows the main process of the proposed method.

**Algorithm 10. Main ()**

**Input:** number of individuals ($L$), the maximum number of rules ($B$), the number of generations ($K$), $R_1^t$, $R_2^t$, $R_3^t, \phi$, number of data centers ($N$), number of fuzzy sets in membership function of input and output variable ($T$), number of objective functions ($q$), datacenters ($DC$), Popular Files ($PF$)
**Output:** Best_individual, Rules
Struct $DC$
{
$id$,
$RAM$,
$tp$, *//tp istransfer rate*
$PW_{\max}$ ,
$PW_{idle}$
}
Struct Rules
{
$id$,
$set\_rule$
}
Struct $PF$
{
$id$,
$size$
}
for (s=1; s<=$size$ ($PF$); s++) *//size function returns number popular files*
{
$P$= Initialize_individuals ($L$, $B$, $N$); *//Generate initial population based on Algorithm 4 and like Fig. 5*
$V$= Initialize_valocity ($L$, $B$); *//Initialize velocity vectors*
for (i=1; i<=$K$; i++) *//Stopping criteria is reached to k-th generation*
{

Fitness_calculation(P); *//Calculate fitness of each individual base on Algorithm 2*
Best individual= Best ($P$); *//Return the best individual*
While $P \neq \varnothing$ do
{
[$P[b]$, $P[c]$]= Select_random_individual ($P$);

if ($P[b]$. fitness > $P[c]$. fitness)
{
$P_{winner} = P[b]$;
$P_{loser} = P[c]$;
}
else
{
$P_{winner} = P[c]$;
$P_{loser} = P[b]$;

}
new_$V_{\text{loser}}$ = Update_velocity ($P_{loser}$, $V$, $P_{winner}$, $R_1^t$, $R_2^t$, $R_3^t$, $\phi$, $c$); *//Update velocity of loser individual based on Algorithm 10*
new_$V_{\text{loser}}$ = Update_position ($P_{loser}$, $new\_V_{loser}$); *//New position for loser individual based on Algorithm 11*
$P$= New_population ($new\_P_{loser}$, $P_{winner}$); *//Create new population for next iteration*
}
} *//end while*
} *//end for*
Rules[s]. $id$= $s$;
Rule_index= Find_idx (Best_individual. $g \geq 0.5$); *//Find index rules of the best individual*
Rules[s]. $set\_rule$= Select_rule (Best_individual. $r$, Rule_index); *//Select rules of the best individual based on Rule_index*

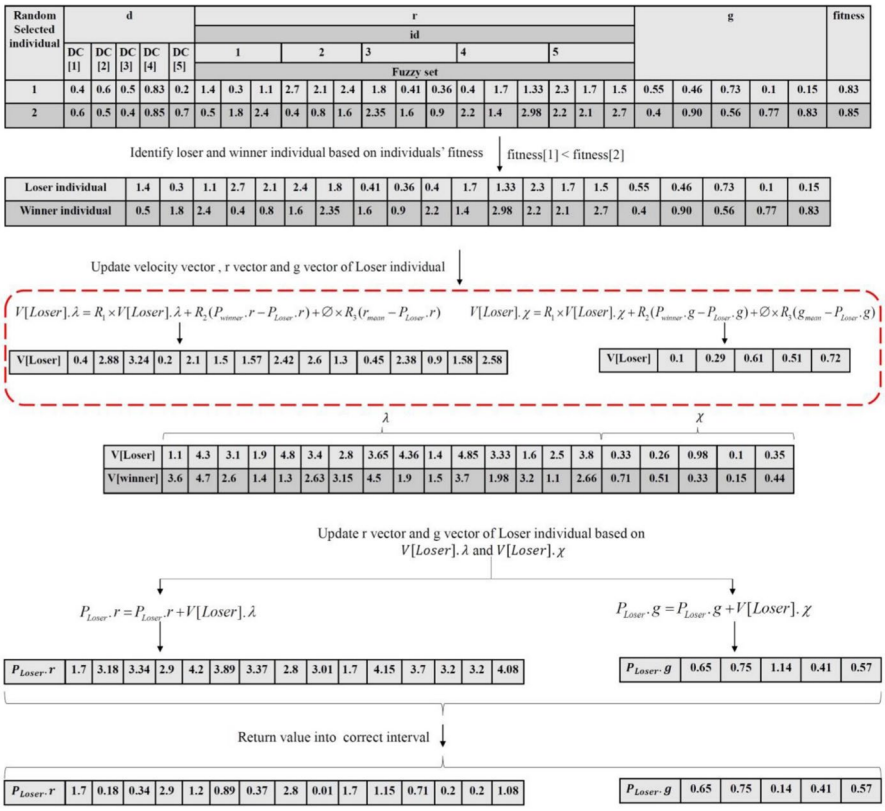


**Fig. 6** An example for updating velocity, *r* and *g* vectors

### 3.2.5 Secure replicas based on T-coloring

In the proposed SDR strategy, the data centers with high *Merit* value in the cloud network are selected to place replica files, so that response time of the system can be improved. To guarantee the security of popular files, they are fragmented into *N* parts. To overcome the security problem of fragments placement, the proposed method applies the T-coloring idea.

*T-coloring description* Hale [43] proposed T-coloring concept for the channel assignment issue so that the interference is avoided. Different graph coloring models are widely applied in various fields (e.g., in frequency assignment or scheduling) due to their practical motivations, special theoretical, and structural features. We divide each popular file into several fragments with various sizes according to the data center capabilities. Then, these fragments are replicated by using the T-coloring concept so that the attacker cannot predict where they will be located. Therefore, SDR algorithm enhances the security level of data without

any encryption technique since each data center has only one part of the file, and as a result, even in a successful attack, useful information is not discovered.

Let $G=(V, E)$ is graph and $T$ is set of non-negative integers. The T-coloring is defined as a mapping function $g$ from the vertices of $V$ to the non-negative integers of set, such that $|g(x) - g(y)| \notin T$, where $(x, y) \in E$. Therefore, $g$ assigns a color to a vertex. The main feature of this concept is that the distance between colors of the adjacent vertices must not be available in $T$.

Initially, we construct $T$ from zero to the non-negative random number. The set $T$ limits the selection of data center to those data centers that are at hop distances not available in set $T$. Also, we assign red color to all of data centers. When we store a fragment in the data center, all of the neighborhood data centers at a distance belonging to $T$ are set by black color. This process is explained in Fig. 7.

In this way, it is possible to reject some of the central data centers that may improve the access time, but the security is enhanced. If the attacker compromises a data center and gets a fragment, then the other fragments location cannot be found. However, the prosperous coordinated attack probability is extremely minute. In fragment placement based on the centrality, the proposed strategy also considers load, storage usage, and energy consumption to improve availability and reduce energy consumption.

SDR strategy can handle the following cloud-specific attacks:

- *Data Recovery* It leads to a rollback of virtual machine to a previous state and reveals the previous files.
- *Cross-virtual machine attack* It causes to data breach by malicious virtual machine invading co-resident virtual machine.
- *Improper media sanitization* It leads to revealing data due to unworthy sanitization of storage elements.
- *E-discovery* It affords the disclosing data of one customer by confiscating hardware to investigate related to some other customers.
- *Virtual machine escape* It leads to access to storage and computing elements by a malicious user or virtual machine escapes from the management of the virtual machine administrator.

The common point in all the above cloud attacks is that penetrators try to expose data in different ways. But even in case of successful attacks our proposed algorithm ensures that the attacker gets only a part of the file.

The proposed algorithm (SDR) divides a file into several parts that a single part does not contain any meaningful information and replicates them on different data centers. Each of the data centers keeps only a single part of a specific file to guarantee that even in case of a successful attack, no useful information is achieved by the attacker. In addition, a successful attack on a single data center cannot find other parts in the system since we store parts of a file based on the graph T-coloring. In this way, data centers that store parts of a particular file are not adjacent and are at a certain distance from each other. In SDR algorithm, an attacker must compromise several data centers to achieve valid information. Also, the number of compromised data centers must be larger than n since each of the

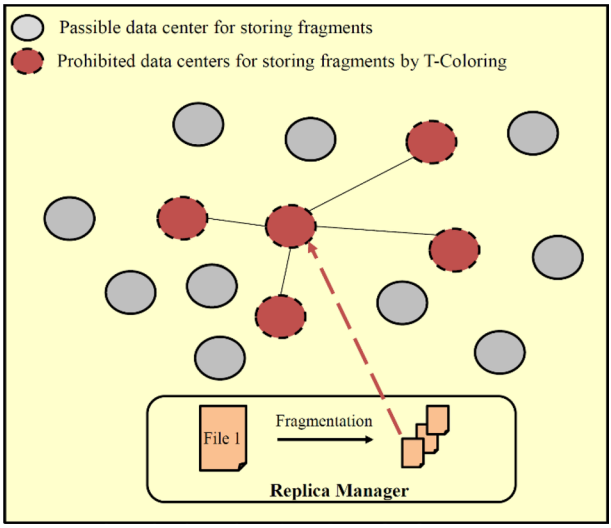


**Fig. 7** Replica placement based on the T-coloring

compromised data centers may not give part in the proposed algorithm as the data centers are separated according to the T-coloring approach.

The necessary effort of an attacker to threat a data center (in environments that keep parts/shares of data) can be found as follows [44].

$$E_T = \min(E_A, n \times E_B) \tag{24}$$

where $E_T$, $E_A$ and $E_B$ indicate the necessary effort to compromise the confidentiality, authentication and a single data center, respectively.

The main goal of paper is data security in cloud, and we do not focus on the authentication system. Hence for obtaining $n$ parts of a file, the attacker effort increases by a factor of $n$. In addition, in our proposed system, the attacker must correctly suggest the locations storing parts of file. The worst case scenario is that the set of data centers compromised by the attacker will have all the file parts. Based on Eq. (24), we can see that the probability of the worst case happen is very low and the worst, average and best cases are dependent on the number of data centers keeping parts that are chosen for an attack.

*Store replica fragment parts* SDR divides the file into predefined fragments and stores them in suitable data centers by considering security and response time. The partitioning of a file into fragments is specified according to the file owner criteria. The user determines the threshold of division with a specific percentage or size of each fragment. For example, the user specifies that the size of each fragment will be of 10% of the size of file. Alternatively, the user can dedicate each fragment size, for example, fragment_1 of size 1000 Bytes, fragment_2 of size 1500 Bytes. The file owner can divide the file such that each partition does not have critical information, as the user knows all of the facts about data file. If the owner does not determine the threshold of fragmentation, SLA uses the default value for percentage fragmentation.

**Algorithm 11. SDR**

```
1 Input: Fp= {F1, F2, …, FS}   //Fp is list of popular files , DC={DC1,DC2,...DCN} //Set of datacenters obtained by Best_individual from Algorithm 3
2 Output: Store fragments based on T-coloring
3 C={Red_color, Black_color};
4 DCs_path= Find_short_path (DC) //Return the shortest path of data centers to each other
5 for each Fk in set Fp
6 {
7  Store all data centers in DC into Set1;
8  Set color of all DCs to Red_color;
9  k=1;
10  for(k=1; k<=F; k++) //F is number of fragments
11    {
12      Select the k-th data center (DCk) from Set1;
13      if (color of DCk = Red_color)
14        {
15          Data_centerX* = Distance(DCs_path, T);   //Return all data centers at distance T from DCk and store them in set Data_ centerX *
16          Change color of all data centers in set Data_centerX* form Red_color to Black_color;
17          Store the selected data center (Data_centerX) into Set2;
18          Remove DCk from Set1;
19          Insert DCk to Set3;
20        }
21    }  //end while
   //Divide Fk into N fragments based on the merit of data centers in Set2
22 R_F= Replica (Fk )//Create a replica from selected popular file
23 Divide R_F into F fragments;
24 Calculate Pn for each datacenters in Set3;
25 } //end for
```

We use the fragmentation technique described in our earlier work [23]. Algorithm 11 indicates SDR strategy. SDR strategy sorts all data centers according to the *Merit* in descending order, and it stores the sorting result into *Set1* (line 8). Then, it selects the $N$ data centers from the top of *Set1* by considering the color of them (lines 9–21) to store a fragment of replicas. SDR store $N$ fragment of replica into $N$ selected data centers (lines 22–24). The portion of the replica fragment can be obtained as the following formula:

$$P_x = \frac{T_x}{\sum_{j \in Set2} T_j} \tag{25}$$

where $T_j$ denotes the merit value of data center $j$.

We explain the proposed strategy with the following example based on $N=3$. Consider list of popular files is $F=\{file6, file3, file5\}$, as shown in Fig. 8. For *file6*, it selects three data centers (*Data center*2, *Data center*1 and *Data center*4) from *Set1* that is sorted based on *Merit*. Using Eq. (25), the portion of *file6* to be replicated on *Data center*2 is 50%. So we can replicate the first 50% of *file6* on *Data center*2,

the remaining 30% of *file6* on *Data center*1 and the remaining 20% of *file6* on *Data center*4.

It is interesting that even in successful violation, the SDR strategy guarantees that the hacker gets only one partition of file. This is because it keeps only one partition of a particular file on the data center. Figure 9 indicates the flowchart of SDR strategy.

### 3.3 Replica replacement

Due to storage space constraints, a replica replacement process is required. We assign a value to each file based on a fuzzy system with three inputs as Number of Accesses (*NA*), Cost of Replication (*CR*) and Last Access Time to File (*LATF*). *CR* is obtained based on Eq. (26) [45].

$$CR = \frac{S}{Bw(p,q)} + DT(p,q) \tag{26}$$

where *s* is size of file, and *Bw(p,q)* and *DT(p,q)* show the bandwidth and propagation delay time between data centers *p* and *q*, respectively.

The high *NA* value and the high *LATF* value for a file indicate that this file will request more in the future. The high *CR* value for a file means that if a data center needs this file, the cost of copying it will be high. Figure 10 indicates the designed fuzzy system for replica replacement process. We consider 20 fuzzy rules, and Table 3 shows some examples of them for value assignment to files.

After computing the value of all the files, they are sorted in ascending order. When there is not enough storage space to store the new replica in the selected data center, we begin to delete the files from the beginning of the sorted list until sufficient space is provided.

### 3.4 Time complexity of SDR algorithm

There are *N* data centers $D=\{D_1,..., D_N\}$ and *M* files $F=\{f_1, ..., f_M\}$. We have three main steps (i.e., determination of popular file, replica placement and replica replacement). For each step, the time complexity is explained as follows.

(1) *Determination of popular file* This step has $O(M \log M)$ time, since *M* files must be sorted based on access count in descending order.

(2) *Replica placement* Consider popular files set $Fp=\{f_1,..., f_S\}$, maximum generation *K*, *L* individuals and *B* maximum number of rules.

- Generation of the initial population for each popular file takes $O(LB + LN)$ time (lines 21–24 of Algorithm 10).
- The time complexity for finding high fitness value is $O(N \log N)$ (line 28 of Algorithm 10).
- Array *d* must be sorted for calculating fitness of each individual (line 29 of algorithm 10), so we have $O(L \times (N \log N))$.

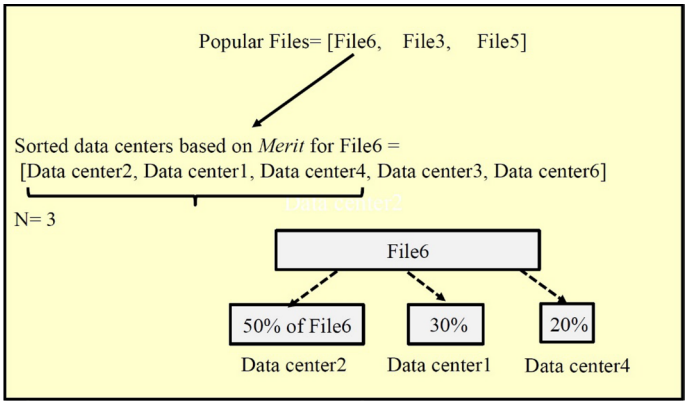


**Fig. 8** Example of fragment placement

- Generation of winner and loser population takes $O(L/2)$ time (lines 32–43 of Algorithm 10).
- Updating velocity and position of loser individual for each popular files takes $O(B \times (L/2) + N \times (L/2))$ time (lines 44–45 of Algorithm 10).
- Finding the best individual takes $O(L \log L)$ time (line 52 of Algorithm 10).
- Consider that lines 26–56 must repeat $K$ times, so we have:

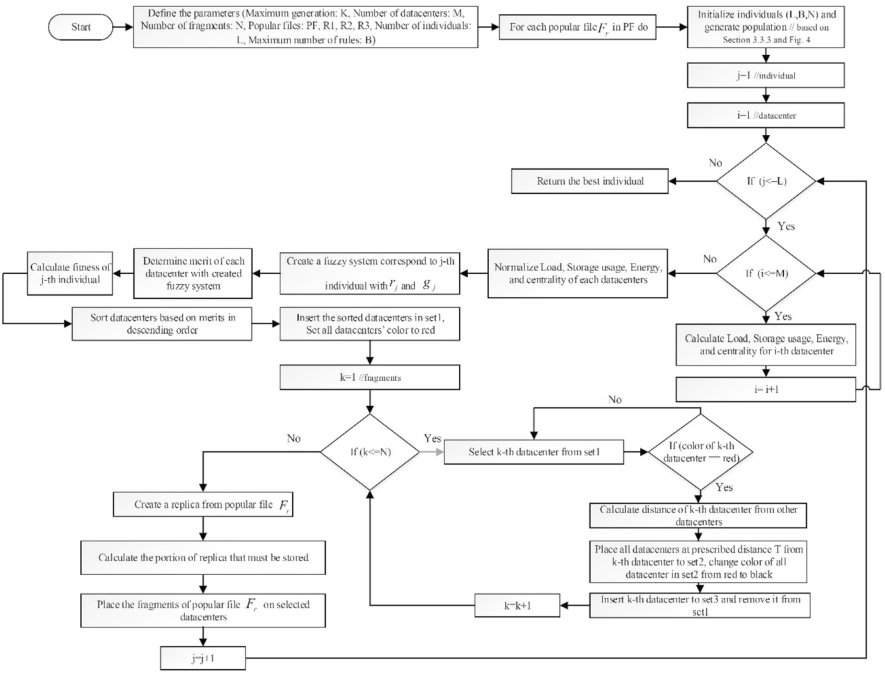


**Fig. 9** Flowchart of SDR strategy

$$O(K \times L \times (N \log N)) + O(K \times (L/2)) + O(K \times B \times (L/2)) + O(K \times (L \log L))$$

Now, we do all the above operations for each $S$ popular files. As $K$, $B$, and $L$ are the prior configured constant for the SDR which has nothing to do with the problem size, the worst-case time complexity can be simplified as $O(SN \log N)$.

- In Algorithm 11, the shortest path of data centers to each other must be calculated, so we have $O(|e| + |N| \log |N|)$ where $|e|$ is the number of links between data centers (Dijkstra algorithm for line 3).

From lines 5 to 20, appropriate $F$ data centers for each popular file must be calculated. So in worst case takes $O(S \times N \times F)$, where $F$ indicates number of fragments.

From lines 22 to 24, the size of fragment must be calculated based on merit of specified data centers, so it takes $O(S \times F)$ times to execute these lines.

At the end, the complexity of this part is equal to:

$$O((|e| + |N| \log |N|)) + O(SNF) + O(SF)$$

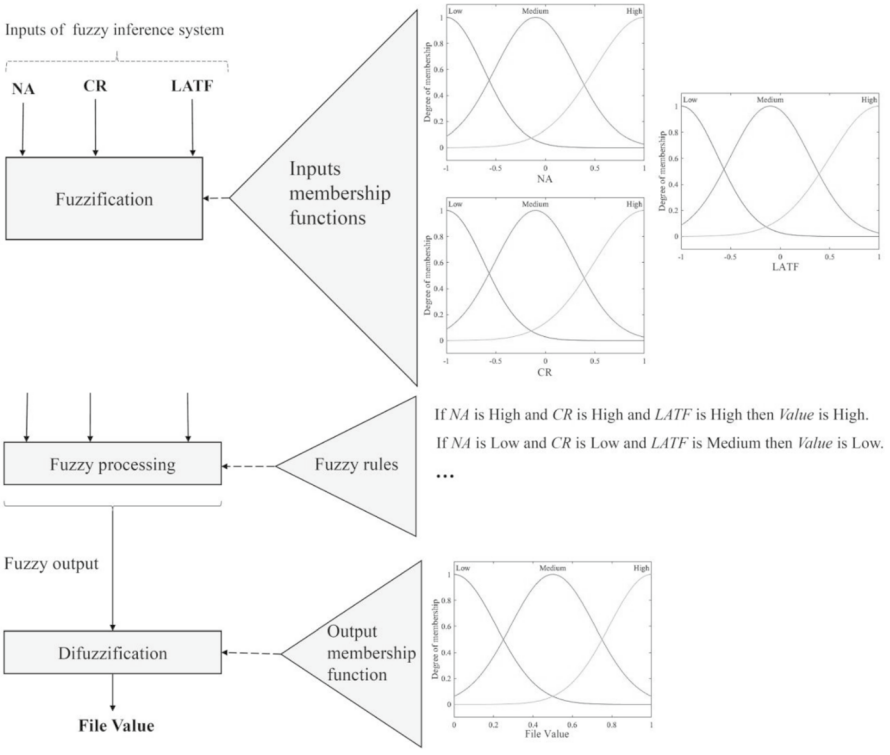


**Fig. 10** Fuzzy system for replica replacement process

**Table 3** Examples of rules in fuzzy system of replica replacement

| NA | CR | LATF | Value |
|---|---|---|---|
| High | High | High | High |
| High | Medium | High | High |
| High | High | Medium | High |
| Low | High | High | Medium |
| High | Medium | Medium | Medium |
| Low | Low | Medium | Low |
| Low | Medium | Low | Low |

As $F$ (i.e., number of fragments) is the prior configured constant and has nothing to do with problem size, it can be ignored for calculating complexity.

*(3) Replica replacement* This step takes $O(M \log M)$ time. Because SDR sorts all files based on the value of replica.

Finally, time complexity is $O(SN \log N) + O((|e| + |N| \log |N|)) + O(M \log M)$.

## 4 System model

Cloud environment usually includes power-consuming data centers that present the elasticity and scalability required by its users. Also, some of data centers process huge data-intensive applications and MapReduce [46] based on such GFS [47], HDFS [48] distributed storage system, which has been considered as a primary element for developing cloud services or applications. Today, there are several cloud simulators available for specific purposes. Table 4 indicates a brief introduction to the well-known cloud simulation tools. One of the famous and popular cloud simulators is CloudSim that is used by most cloud researchers. CloudSim has good features like [13]:

- Simple and fast implementation of a cloud computing environment;
- Simulation large-scale data centers;
- Modeling of energy-aware computational resources;
- Providing different strategies for the management of different components of the cloud such as scheduling and provisioning;
- Providing different perspectives such as cost and application execution time to evaluate the competence of algorithms;
- Simulation of network connections among different system elements.

One of the main problems of this simulator is the lack of a graphical user interface, but extensions such as CloudReports [56] offer a GUI for CloudSim. In addition, to the benefits of CloudSim mentioned above, the main reason for choosing it in this work is that the estimations of the linear CPU power model of CloudSim

are very close to those presented by the linear interpolation model based on real measurements.

We extend different classes of CloudSim to support the requirements of data replication. For all replication algorithms, we implement *File makeCopy()* method in *Class File* of CloudSim to copy data files in appropriate data centers. *ReplicaManager()* method is added to manage all data manipulation on the system. Moreover, *Storage*, *NetworkTopology*, *Cloudlet*, *File* classes require a little change.

- *MORM* It uses a linear power model that is implemented by CloudSim at *Class PowerModelLinear* which implements *Class PowerModel*.
- *EFR* This method does not have a specific parameter and is easily implemented at the *Class File* in CloudSim simulator.
- *PEPR* All necessary parameters (such as storage cost, processing cost) at *Class Data center Characteristics* of CloudSim are defined based on [22].
- *EDR* We define the energy consumption (*Es*) model for a single machine based on Eq. (27).

$$E_s = \alpha p + \beta p^3 \tag{27}$$

where $\alpha = 1$, $\beta = 0.01$ and we have workload $p$ (size per request is 100 Mbytes). Therefore, we implement a new *Class PowerHost* extends *HostDynamicWorkload*. In addition, network latency is determined by *Latency Matrix* and *Class NetworkTopology* of CloudSim. This class can set a delay-topology storing every distance between connected nodes.

- *BaRRS* It considers execution time and monetary cost that can be modeled in CloudSim.
- *DPRS* We modify *Class FileAttribute* to design fragmentation approach.

The necessary dynamic parameters of SDR algorithm are obtained as follows.

- *Transfer rate* In *Storage* class of CloudSim, *getMaxTransferRate()* method returns the maximum transfer rate in MB/sec.
- *Storage usage* In *Storage* class of CloudSim, *getAvailableSpace()* method returns the available space in MB.
- *Access rate* Based on [16], the requests of file $f_i$ can be modeled by the Poisson process with a mean access rate *Acc (i)*. Therefore, $Acc(i) = \sum_{j=1}^{m} Acc(i,j)$.
- *CPU utilization* In *HostDynamicWorkload* class, *getUtilizationOfCpu()* method returns current utilization of CPU in percent.
- We define a table includes *file_name*, *file_location*, *access_count*, *file_value*, *last_access_time* to record file access information. This table is updated after each data file access. We use this table for finding the popular file and replica replacement step.

In this work, we assume that the cloud is composed of different independent heterogeneous data centers. It is considered that all the data are read-only, and no data

**Table 4** Cloud simulation toolkits

| Simulation Software | Language | GUI | License | Federation policy | Communication model | Energy model | Cost model | Description |
|---|---|---|---|---|---|---|---|---|
| CloudSim | Java | No | Open source | Yes | Yes | Yes | Yes | Support complex scheduling, Provide very large-scale clouds |
| TeachCloud [49] | Java | Yes | Open source | No | No | No | No | Provide MapReduce framework and the Rain cloud workload generator |
| GreenCloud [18] | C++ | Yes | Open source | Yes | Yes | Yes | No | Implement a full TCP/IP protocol |
| GDCSim [50] | C++ | No | Open source | No | No | Yes | No | Thermal analysis, Consider cyber-physical interdependency |
| iCanCloud [51] | Java | Yes | Open source | No | Yes | No | Yes | Support a wide range of storage models |
| CDOSim [52] | Java | Yes | Paid | No | No | No | No | Simulate SLA violations, Follow the knowledge discovery meta-model |
| NetworkCloudSim [53] | Java | No | Paid | Yes | Yes | No | Yes | Model the network and more fine-grained models |
| MDCSim [54] | Java | No | Paid | No | No | Yes | No | Thermal analysis, Workload management |
| DISSECT-CF [55] | Java | No | Open source | Yes | Yes | Yes | Yes | IaaS internal behavior, Fast evaluation of many scheduling |

consistency strategy is necessary. In our case, we assume 20 different files with size 500 MB. In total, 1000 jobs are submitted to the service providers based on the Poisson distribution and ensure their processing requirements higher than 10 MI. Each job requires [2–10] data files randomly. The number of replicas of each data file is one and stored in different data centers randomly at the first of simulation. Table 5 describes the topology setup. These parameters are defined according to the existing works [57] to show a typical cloud system.

## 5 Simulation results

In this section, we evaluate the proposed method in two main parts. In the first part, the results of the sensitivity analysis for the selected percentage about the number of popular files are discussed and then the proposed method is investigated in terms of main performance metrics (i.e., average response time, storage usage, effective network usage, energy consumption, hit ratio, mean latency, load variance, number of replications, efficiency and bandwidth consumption). In the second part, the proposed method is tested in terms of the structure of CSO-like population size and the number of iterations.

### 5.1 Performance metrics evaluation

#### 5.1.1 Sensitivity analysis

We do several tests to show the impact of selecting different percent of popular file on the functionality of the proposed method behavior. We provide a comparison between task failure percentage and percent of referral to selecting the popular file in Fig. 11. As can be seen in Fig. 11, by increasing the selective percentage, the number of referrals to those files decreases. The most referral to files is when 20 to 25% of popular file selects. Also, we can observe that by selecting 5–20% of popular files, numerous tasks refer to these files for their execution, and the number of referrals to these files is high, but a few tasks can successfully execute because they need other files for completing their execution. The most percent of task completion occurs when 20% of files are chosen as popular data.

Figure 12 shows the consequence of selecting a high percent of popular files on energy consumption. At the beginning of the chart, when 5–20 percent of the files are selected, the energy consumption is relatively high. Because the number of remote requests from data centers increases, on the other hand their processing unit is wasted for the execution of tasks. Between 20 and 25% of selecting popular file the energy consumption gets the lowest value, but as the percent of popular files increases, the energy consumption increases with a slight slope. Although the number of remote requests is reduced, the process of replication for a large number of files leads to high energy consumption. Therefore, the case of (20%) has the

**Table 5** Experimental setup specifications

| Parameters | Value |
|---|---|
| Number of regions | 3 |
| Number of data centers per region | 2–10 |
| Number of VMs per data center | 10 |
| Number of different files | 20 |
| Size of each file | 500 (MB) |
| VM storage capacity | 20 (GB) |
| Intra-data center bandwidth | 40 (GB/s) |
| Intra-region bandwidth | 5 (GB/s) |
| Inter-region bandwidth | 1(GB/s) |
| Intra-data center delay | 5 ms |
| Intra-region delay | 25 ms |
| Inter-region delay | 100 ms |
| Processing ability | 1000 MIPS |
| $\text{Tem}_{\text{in}}$ | 20 |
| $\text{Tem}_{\text{out}}$ | 30 |
| $\text{Pw}_{\text{max}}$ | 300–500 (W) |
| $\text{Pw}_{\text{idle}}$ | 50–60 (W) |

best energy consumption, showing that it can accurately reflect which files are more popular. As a consequence, we consider only 20% of files for replication.

### 5.1.2 Average response time

The response time is introduced as the interval between the job submission time and return time of the result. Therefore, the average response time of a system is given by the following:

$$\text{Average Response Time} = \frac{\sum_{j=1}^{m} \sum_{k=1}^{m_j} (ts_{jk}(rt) - ts_{jk}(st))}{\sum_{j=1}^{m} m_j} \tag{28}$$

where $ts_{jk}(st)$ and $ts_{jk}(rt)$ show the submission time and the return time of the result of task $k$ of the user $j$, respectively, and $m_j$ indicates the number of the tasks of user $j$.

Figure 13 compares the average response time of the replication strategies for the uniform distribution and Zipf distribution. We can see that MORM strategy outperforms the BaRRS algorithm by up to 13% when the number of tasks is 1000. Since MORM strategy replicates based on the file unavailability, service time, load variance, energy usage and latency, BaRRS achieves a better average response time compared to EFR due to creating a much greater number of copies in the cloud data centers. DPRS executes jobs on a data center with the most required files and has a slightly better average response time in comparison with PEPR strategy. However,

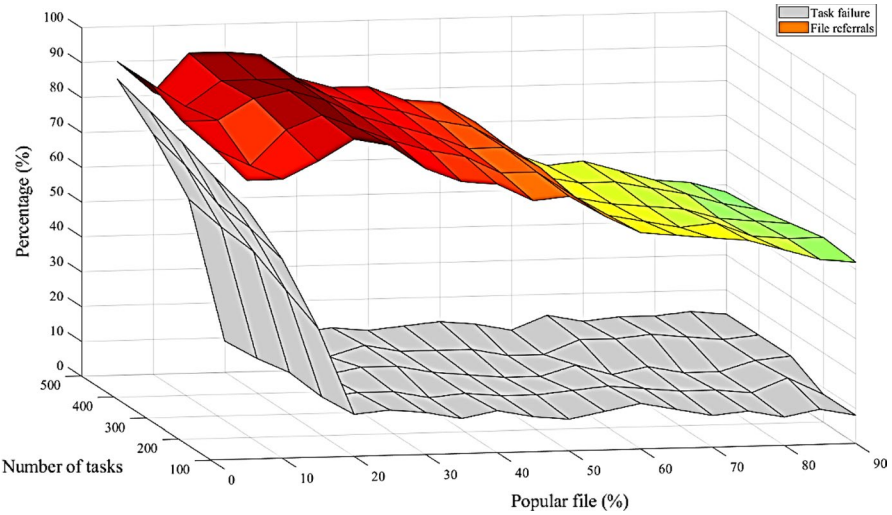


**Fig. 11** Task failure and file referrals percentage

it is necessary to highlight that the goal of provider is not only to achieve the best access time but also to satisfy the QoS with a high-security level. SDR strategy is designed based on the security and performance factors. It breaks the data file into different fragments and then parallel downloads replicated data fragments to reduce access time.

In the replica placement phase, it applies the centrality measure that ensures an improved response time and it seemed very meaningful and efficient than considering

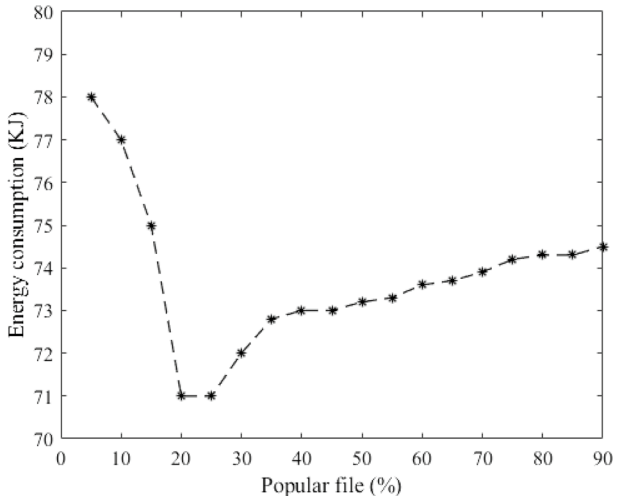


**Fig. 12** Energy consumption by different percentage of popular files

simple hop-count kind of metrics. Furthermore, SDR increases the total number of local accessed by replicating popular files in the best location.

Figure 14 shows the impact of size of files in response time for data replication algorithms based on Zipf distribution. We can observe that increasing the file size increases response time. Since the transfer of large files among data centers will increase the waiting time, SDR algorithm has a response time about 39% faster than EFR. As you can see in Fig. 14, the proposed method is much more prominent for the large-scale files that are similar to a real environment where huge datasets should be processed. The main reason for this improvement is the use of proper parallel techniques. The overhead of parallel downloads is very large for small files, and so a larger file shows a better download efficiency than a smaller one.

In Fig. 15, we show the response time of the replication strategies w.r.t the number of data centers. The results indicate that SDR algorithm has the best performance compared to other data replication strategies. In the best case, SDR reduces the response time by up to 46%. The results also show that when the number of data centers increases, the time difference between methods decreases.

### 5.1.3 Storage usage

Since one of the main resources in cloud is storage, tracking storage capacity is a key part of replication strategy. The percentage of storage usage is specified by Eq. (29):

$$Storage\ Usage = \frac{Total\_Storage\ Space - Available\_Space}{Total\_Storage\ Space} \times 100 \tag{29}$$

Because data centers have limited storage space, replica replacement and replica placement are critical processes, especially when the resource cost is proportional to the amount being consumed. Different replication algorithms lead to different storage resource usage as shown in Fig. 16.

MORM algorithm saves storage space about 20% compared with EDR method since MORM instead of storing many replicas in a lot of locations, it places new replicas in the best data centers. Referring to the resource consumption, by using DPRS, the storage usage is decreased by outperforming PEPR, 10%. This

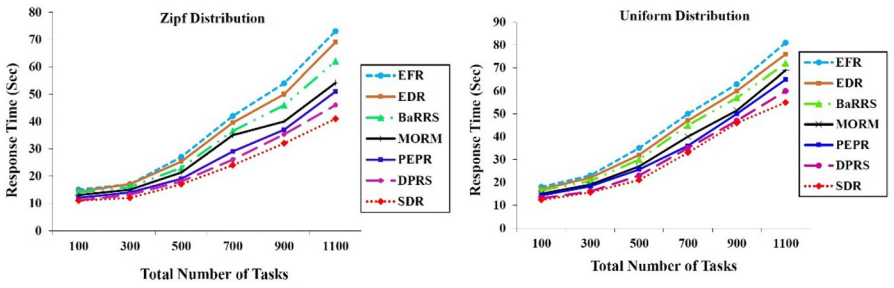


**Fig. 13** Average response time with different number of tasks

is because DPRS only keeps frequently accessed files that are a small part of the whole dataset; thus, the storage usage of replication is decreased. The storage space consumed by the SDR algorithm is approximately half that is consumed by the EFR algorithm. Since SDR strategy does not store replicas to so many data centers except the important one for saving storage space, in other words, it only replicates less than 20% of the overall popular files based on the 80/20 concept. Moreover, it stores only replica fragments instead of the complete replica on the server to reduce the storage usage.

### 5.1.4 Effective network usage

In general, effective network usage (ENU) is used to investigate the efficiency of network resource usage. It shows the rate of transferred files to the requested files. Therefore, if the replication algorithm can store replicas in critical data centers, then ENU has a low value. In Eq. (30), $N_{rfa}$ shows remote number accesses, $N_{fa}$ is the total number of file replication operations, and $N_{tfa}$ indicates the number of times that site reads a file [58].

$$E_{enu} = \frac{N_{rfa} + N_{fa}}{N_{tfa}} \tag{30}$$

The effective network usage of DPRS is better compared with MORM and PEPR strategies because of considering access costs in replica placement. As depicted in Fig. 17, the ENU of PEPR is lower by 38% compared to EFR algorithm. This is because PEPR transfers most of files at the intra-data center level using replicas that are stored in a local region with high bandwidth. If accessing a file from a remote

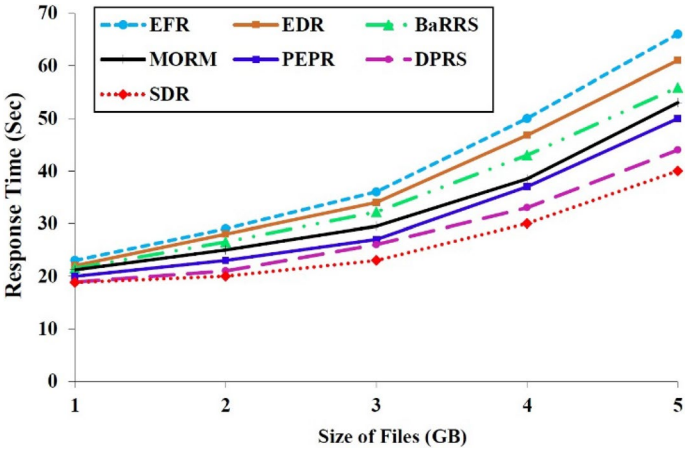


**Fig. 14** Average response time with different size of files

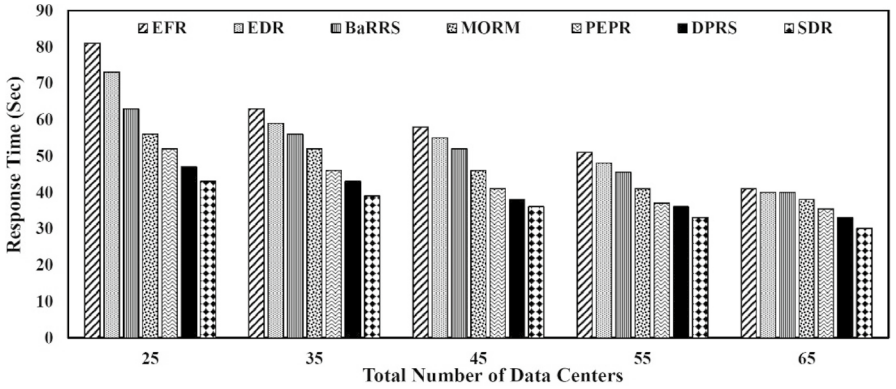


**Fig. 15** Average response time with different number of data centers

data center is more profitable than storing a new replica in the local data center, PEPR performs such an action to improve overall performance. When compared to the EFR, BaRRS, MORM, PEPR and DPRS algorithms, the SDR algorithm performs better because of considering storage usage and load rate in replica placement decisions. It can put replicas at proper data centers compared with others in different circumstances, so the number of remote replica accesses is decreased. Required files are available locally in most cases. Moreover, SDR algorithm stores high-value replicas and deletes low-value ones using a fuzzy-based replica replacement process.

### 5.1.5 Energy consumption

The servers execute cloud applications, which do a certain amount of computing job and make a single database query for successful completion. Figure 18

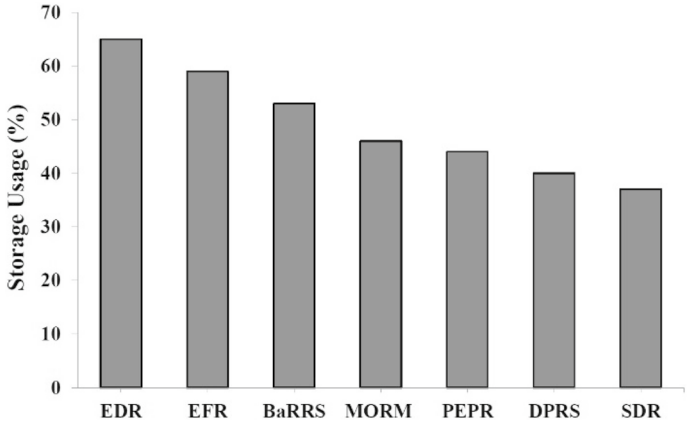


**Fig. 16** Storage usage of different replication algorithms

reports the energy consumption of computing servers for different data file size. Each server gets one data item every 300 ms and does not update the database. Two results can be observed in Fig. 18. The first result is that energy consumption increases with the increase in data size. The second is that energy consumption decreases as data are stored close to the computing server locations.

The communication delay is included in the execution time of the cloud application, which prevents the modification servers from the active state. These delays increase with increasing data size but can be decreased by shortening round-trip times to the database. As can be seen from Fig. 18, the energy consumption with the adoption of our replication strategy is significantly less than the other compared strategies. Also, we can get the obvious conclusion that both the replica selection systems using EDR and MORM can reduce the energy cost of data centers for data-intensive applications. In addition, the efficiency of the system implementation for SDR is much higher than that of EDR. This is because SDR strategy takes into account both total the energy cost of the entire system and bandwidth capacity for each replica in determining the system-wide energy model. In addition, it uses parallel downloading of files that are very important when the resources such as network bandwidth are limited. We conclude that it maintains a balance between the performance of system and the energy saving. The order of the evaluated algorithms in this diagram has completely changed, and the methods that have taken the energy parameter into account in their decision making have been more successful in terms of energy consumption. For example, EDR algorithm reduces energy consumption about 20% compared with PEPR method.

### 5.1.6 Hit ratio

Hit ratio is another common performance metric in replication works that is defined as follows:

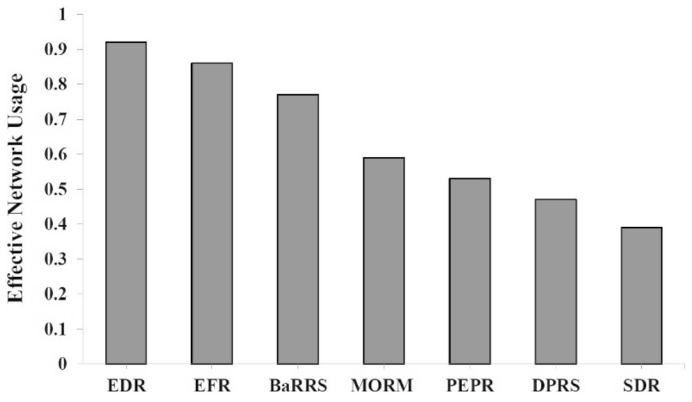


**Fig. 17** Effective network usage

$$Hit\_Ratio = \frac{Local\_Access}{Total\_Access} \tag{31}$$

We can see from Fig. 19 the hit ratio for SDR algorithm is the highest of all algorithms. When the number of files is considered 50, the hit ratio for SDR, DPRS, PEPR, MORM, BaRRS, EFR and EDR are 75, 73, 68, 66, 45, 40 and 37, respectively. In other words, the proposed method (SDR) in average increases the hit ratio by 36% in comparison with other methods. Since it properly deals with replica placement and thus the total number of local accesses has been increased, the hit ratio has been enhanced.

It can be seen from Fig. 19 that EDR algorithm obtains the lowest hit ratio compared with other methods (in average about 60% lower); on the other hand, we know that the higher value for hit ratio shows good performance of the replication method in placing replica files in a suitable location. EDR has bad performance in terms of hit ratio because it only focuses on bandwidth capacity during the replication process.

### 5.1.7 Mean latency

From Fig. 20, we can see that EFR algorithm performs better than EDR in average and reduces the latency by 6%. In EDR strategy, the amount of transferred data in a network is low and the average access latency of data is decreased ultimately. Furthermore, compared with EFR, the mean latency of MORM can reduce up to 12%. This phenomenon is because EFR does not consider latency as optimizing targets. Compared with MORM, SDR can provide up to 24% reductions in average latency.

The reason behind this result is that SDR considers the centrality factor in replica placement, whereas other strategies do not take into account this parameter and so the selected data center is not always the best replica provider since the link of the selected provider might have high traffic or maybe in an inappropriate position. It creates a number of additional, well-placed replicas and the subsequent accesses

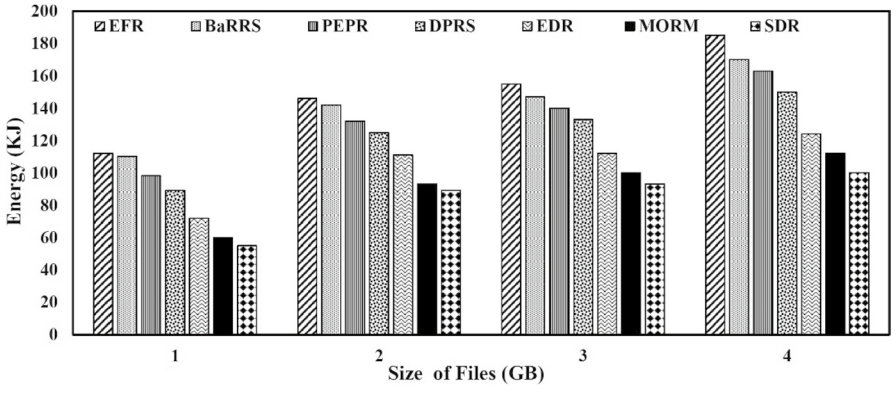


**Fig. 18** Energy consumption of servers

of popular files can be distributed across data centers with a low load rate and maintains a load balance among the data centers that have popular files and hence reduces latency value, while in other strategies replicas are centralized to a few hot data centers and hence data requests will wait a long time in the queue. The experimental results indicate that BaRRS strategy has acceptable results only when the network bandwidth is the unique metric for the network performance. However, this is not a realistic assumption in the cloud environment.

Several simultaneous factors can affect the latency and lead to a downward and upward trend, so there is no specific rule for the ascending or descending trend associated with the number of files. The downtrend at the beginning of the chart (the number of files from 10 to 40) can be justified which probably the number of replicas in the system is increased. Since latency has a reverse relationship with a number of replicas, on the other hand, the number of requests for a file by a data center in a fixed time interval is modeled based on Poisson distribution. For example, if there are in average 50 accesses for 10 files, then by an increase in the number of files, the number of accesses is also getting high. So, when we have 40 files, the number of accesses is 2000 times. It is expected that when the number of files is too high, the delay due to the increase in the number of accesses is also incremental. In other words, the amount of latency with the number of file accesses has a direct relationship.

It can be seen from Fig. 20 that SDR algorithm and MORM by increasing the number of files from 40 to 50, latency value is also increased with this difference. The increasing latency value for MORM occurred with a steeper slope compared with SDR (i.e., SDR is increased by 10% and MORM by 13.5%). This is because SDR algorithm creates a balance between these numbers of replication and latency by defining several fuzzy rules. Unlike SDR strategy, MORM ties to create a balance with weights, which highly depended on user opinions not the situation of cloud environment.

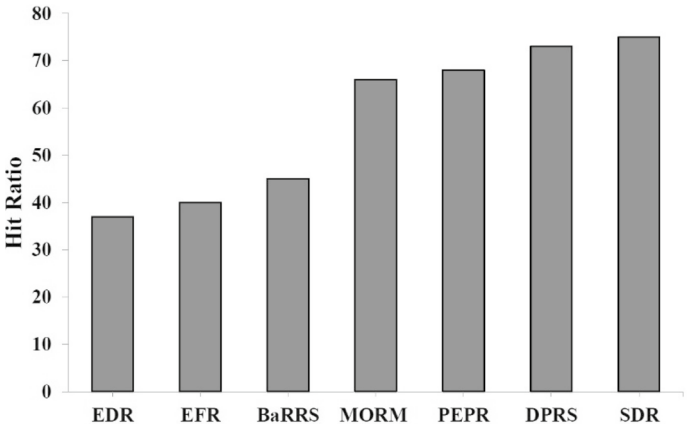


**Fig. 19** Hit ratio

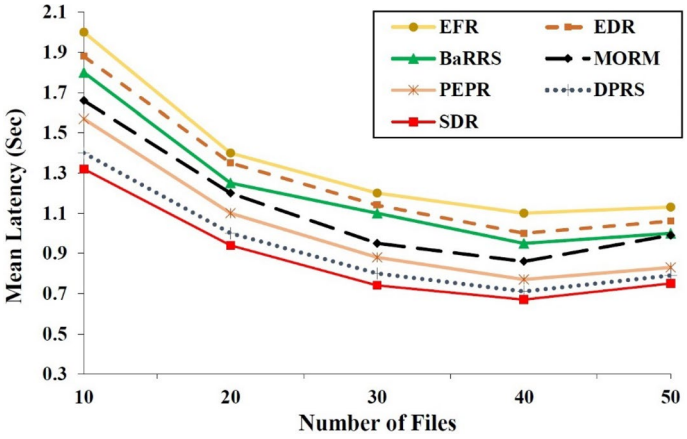


**Fig. 20** Mean latency with different number of files

### 5.1.8 System load variance

Figure 21 shows data center load variance that is defined as the standard deviation of data center load of all data centers in the cloud environment. We can observe that the proposed replication algorithm has the lowest value for load variance (in average reduces load variance about 26% compared with other methods) and so has good load balancing for example when number of files is set to 400, SDR, EFR,EDR, BaRRS, MORM, PEPR obtaining 0.22, 0.34,0.32, 0.28,0.29, 0.28 and 0.24 for variance parameter, respectively. It means that SDR reduces load variance by 35%, 31%, 21%, 22%, 21% and 9% compared with these methods, respectively. The main reason is that SDR considers load as one of the main parameters in replica placement.

It can be seen from Fig. 21 that BaRRS from 100 to 400 files has the worse load variance in comparison with MORM algorithm (BaRRS obtains about 8% higher variance compared with MORM). Then between 400 and 500 files it has better performance in comparison with MORM algorithm, but for the highest number of files (i.e., 600 files) MORM algorithm reduces load variance by 4%. One of the main reasons for this fluctuation between MORM and BaRRS in terms of load variance is for the heuristic nature of MORM, and so it cannot always find the global optimum (i.e., appropriate data center for placing replica so that it reduces the load variance).

### 5.1.9 Number of replications

Figure 22 represents the total number of replications for seven replication algorithms at various numbers of files. MORM strategy can achieve a reduction in the total number of replications between 14 and 7% when compared with EFR and EDR algorithms, respectively. Since it replicates files in appropriate locations, most of times the necessary files for job execution will be locally available. We can see that

the SDR has a lower number of replications about 8% and 13% than DPRS and MORM, respectively. SDR algorithm replicates only 20% of popular files to get the near-optimal objective value and avoids unnecessary replications.

#### 5.1.10 Total bandwidth consumption

Figure 23 indicates the comparison of bandwidth consumption for different file sizes. As shown, SDR algorithm can improve the performance especially in large file size since it finds hot spot files which are normally less than 20% of the whole data resource according to the 80/20 rule. In addition, it considers centrality and file fragmentation during replication process. We can see that the bandwidth consumption of BaRRS algorithm is lower by 18% compared with EFR strategy. This is because BaRRS takes into account the workflow feature analysis such as file sizes, task parallelism and task interdependencies during task scheduling and selects the suitable replica provider to respond to the data requester. From Fig. 23, we can be observed that MORM algorithm compared with DPRS achieves higher bandwidth consumption in average about 7%, specifically when the size of files is set 4 GB, the bandwidth consumption for DPRS and MORM is 3300 and 3500, respectively. It is because DPRS tries to place new replicas in central data center and so a lower number of data centers are involved to bring files to the requester data center for execution tasks.

#### 5.1.11 Efficiency

One of the main optimization metric is efficiency that is defined based on Eq. (32) [59].

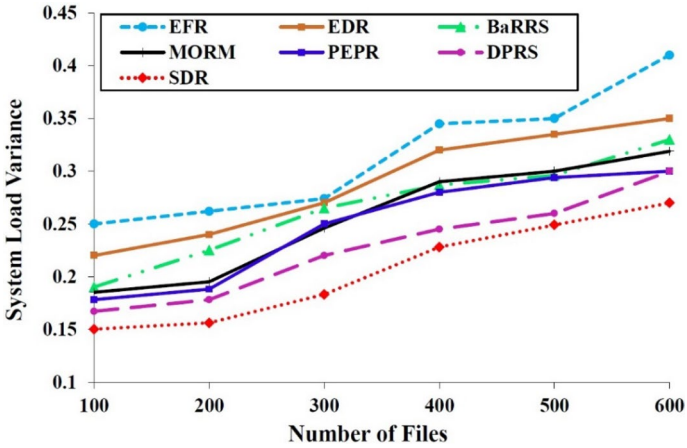


**Fig. 21** Load variance with different number of files

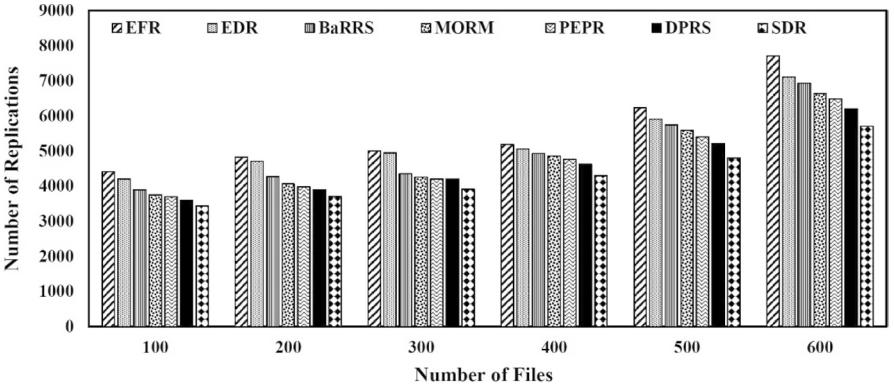


**Fig. 22** Number of replications based on varying number of files

$$Ef = \frac{Sl}{Ms} \tag{32}$$

where $Sl$ is determined by Eq. (33).

$$Sl = \underset{v_j \in V}{Min}\left(\sum_{t_i \in T} Ex_{i,j}\right) \tag{33}$$

where $Ex_{i,j}$ shows the execution time of the task $t_i$ on the data center $v_i$.$T$ and $V$ are set of tasks and set of data centers, respectively. $Ms$ indicates makespan that is introduced as the completion time of the last task.

In Fig. 24, it is obvious that when the number of tasks increases, the efficiency will be reduced for all methods, but the increment ratio of SDR algorithm is much lower than other replication algorithms especially compared with EFR algorithm in average is about 40%. For example when number of tasks is set to 600, the efficiency of SDR and EFR algorithms is 2.5 and 1, respectively. It means that the proposed

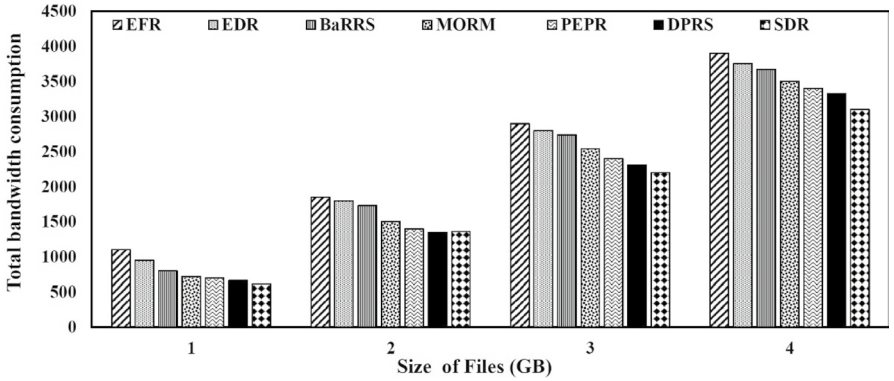


**Fig. 23** Total bandwidth consumption based on varying size of files

method (SDR) has a better performance by 2.5 times in comparison with EFR because it tries to find frequently accessed files and stores them in the best locations by designing a smart fuzzy system. Experimental results demonstrate that using fuzzy logic in the implementation of SDR algorithm can enhance the performance of replication process in terms of efficiency.

Also from Fig. 24, it can be interpreted that DPRS has better efficiency (in average 6.5%) compared with PEPR algorithm for a large number of tasks. Good performance in this metric highly depends on replica placement in appropriate data centers so that the execution time is reduced. In DPRS, files are placed in data center with a high request rate and thus execution time gets lower.

## 5.2 Evaluation of structural parameters

### 5.2.1 Number of rules

Another parameter that can be discussed for the evaluation of SDR algorithm is the size of population versus the number of rules in each individual. We can observe in Fig. 25 that when the population size is set to 60 and number of rules in each agent is set to 64, the heist fitness can be obtained.

### 5.2.2 Number of iterations

Figure 26 shows the response time for the best solution that is found by SDR algorithm in terms of different number of files and a different number of iterations. We can observe that for 30 files, the best solution is obtained in 200 iterations and achieves lower response time (about 70 s) compared with other the number of iterations.

By increasing the number of files to 50 and 60, the best solution in iteration 210 achieves approximately 124 and 153 s for response time, respectively. When

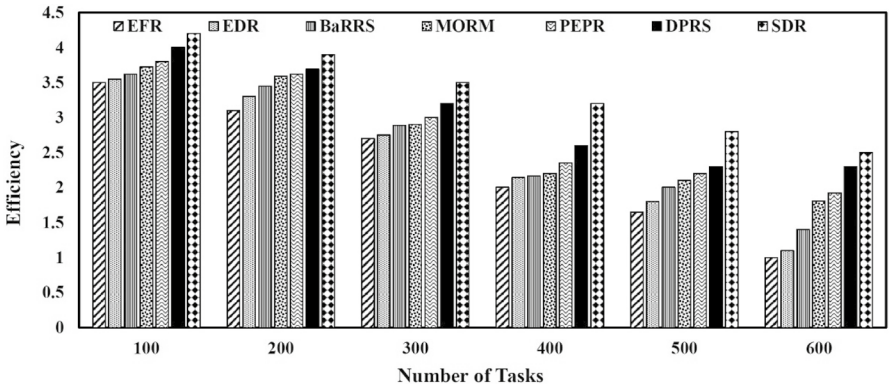


**Fig. 24** Efficiency based on different number of tasks

considering 70 as the number of files the best solution is obtained by SDR in iteration 220 and achieves the lowest response time (about 184 s).

### 5.2.3 Effect of weighting parameters in CSO algorithm

*R1*, *R2* and *R3* are three weights that effect on updating velocity and position of agents (Eqs. (4) and Eq. (5)). These factors are user-defined, and also they are selected in interval (0, 1). In updating velocity of a loser agent for next iteration (Eqs. (20–23)) *R1*, *R2* and *R3* represent the impact of pervious iteration velocity of that agent, the impact of distance between winner agent and loser agent and impact of distance between the average distance of winner agent to loser agent, respectively.

If a high value is set for these weights (e.g., the sum of these three weights is placed in the interval (2,3)), then the search step will be large for finding global optimum. This result is not good because some appropriate solution (solution where after updating positions in several iterations find the global solution) may be ignored. If values for these parameters are considered low (e.g., the sum of these three weights are placed in the interval (0,1)), then the search step will be large and this is not appropriate because for a constant number of iterations all search space is not checked. So, the best solution may not be found.

Figure 27 explains the impact of different values of *R1*, *R2* and *R3* on response time. We can see that by increasing the values of *R1* and *R3* and with a low or moderate value of *R2* (0.1 to 0.4) the achieved response time value is lower than other conditions. In detail, there are six cases where the lowest response time is obtained. The values of *R1*, *R2* and *R3* along with the achieved response time for these six cases are presented in Table 6.

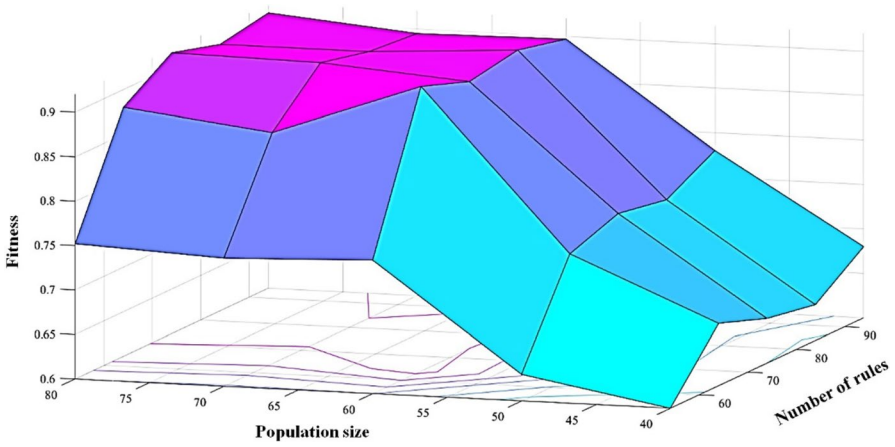


**Fig. 25** Number of rules versus population size

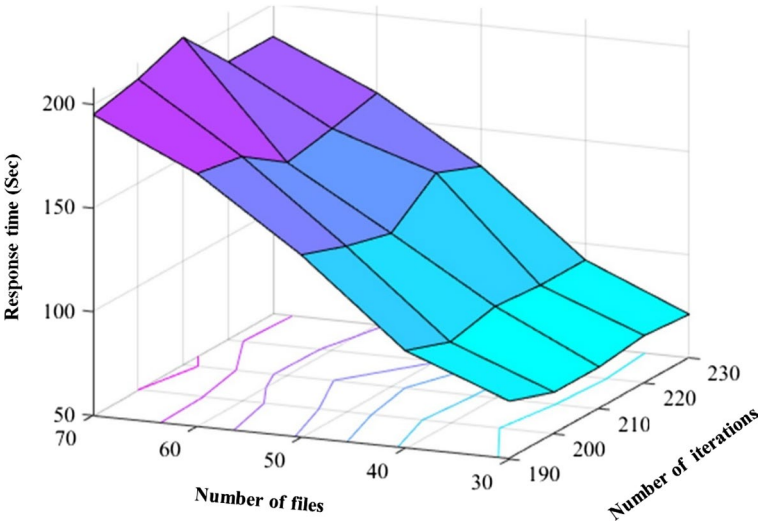


**Fig. 26** Response time in terms of number of iterations and number of files

We can conclude from Fig. 27 and Table 6 that the velocity of an individual and the average position of individuals in the population have a great impact on the quality of individual updation.

## 6 Conclusion

Unlike most security-aware replication strategies that consider cryptographic processes to protect data privacy, we focus on the security issue with T-coloring, energy efficiency and time retrieval of the system. For that purpose, this work introduces a secure dynamic replication (SDR) placement strategy, which improves energy consumption and network bandwidth. SDR finds popular files based on the 80/20 idea by analyzing history. The popular file is divided into several fragments and placed across different locations by designing a fuzzy inference system with four inputs as data center centrality, storage usage, load and energy consumption. Suitable fuzzy rules are determined using CSO algorithm that increases accuracy. Moreover, the T-coloring approach is considered which is not giving any clue about fragment locations to a hacker. Simulation results with CloudSim demonstrate that SDR has better performance in comparison with current algorithms in terms of the average response time, storage usage, effective network usage, energy consumption, hit ratio, mean latency, load variance, number of replications, efficiency and bandwidth consumption. SDR decreases the mean response (over 28%) and, with preventing to store unused replicas, causes the minimum replication operations. In the future, we plan to predict the future requirements of data centers based on past access sequences

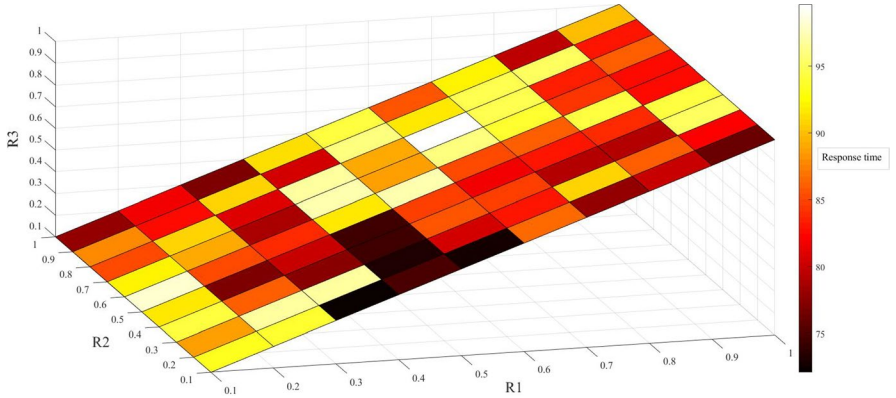


**Fig. 27** Response time for different values of *R1*, *R2* and *R3*

**Table 6** The appropriate values for *R1*, *R2* and *R3*

| R1 | R2 | R3 | Response time |
|---|---|---|---|
| 0.2 | 0.4 | 0.4 | 72.00 |
| 0.4 | 0.4 | 0.4 | 74.50 |
| 0.4 | 0.2 | 0.4 | 72.90 |
| 0.5 | 0.1 | 0.4 | 72.80 |
| 0.4 | 0.1 | 0.5 | 75.09 |
| 0.4 | 0.3 | 0.4 | 74.33 |

with data mining techniques. Furthermore, security levels, job priority as well as other criteria must be considered to the data replication problem so that service providers can satisfy most of service-level agreements through improving system utilization.

## Compliance with ethical standards

**Conflict of interest** The authors declare that they have no conflict of interest.

## Appendix

**Algorithm 12. Normalize_Parameters ()**

```
Input: Data node
Output: Normal_Data center //Normalized parameters of data center
int N_Data centers; //Number of data centers
Normal
{
    double load []= new double [N_Data centers]; //Create an empty array
    double  storage  []= new double [N_Data centers];
   double  energy  []= new double [N_Data centers];
     double  centrality []= new double [N_Data centers];
     for (i=1; i<= N_Data centers; i++)
     {
       Data nodes [i]. Load= load[i];
       Data nodes [i]. Storage Usage= storage[i];
      Data nodes [i]. Energy= energy[i];
      Data nodes [i]. Centrality= centrality[i];
     }
    [max_load, min_load, max_storage, min_storage, max_energy, min_energy,
max_centrality, min_centrality]= Find_max_min (load, storage, energy, centrality);
//Find minimum and maximum of  each array
     for (i=1; i<=N_Data centers; i++)
  {
    Normal_Data node[i]. Load= (load[i]- min_load)/(max_load-min_load);
    Normal_Data node[i]. Storage Usage= (storage[i]- min_storage)/(max_storage-
min_storage);
    Normal_Data node[i]. Energy= (energy[i]- min_energy)/(max_energy-
min_energy);
    Normal_Data node[i]. Centrality= (energy[i]- min_centrality)/(max_centrality-
min_centrality);
  }
return Normal_Data center;
}
```

## References


1. Wei J, Zeng X (2019) Optimal computing resource allocation algorithm in cloud computing based on hybrid differential parallel scheduling. Clust Comput 22:7577–7583
2. Singh Gill S, Ouyang X, Garraghan P (2020) Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres. J Supercomput 76:10050–10089
3. AliKhan A, Zakarya M, Khan R (2019) Energy-aware dynamic resource management in elastic cloud datacenters. Simul Model Pract Theory 92:82–99
4. Mansouri N, Javidi MM (2020) A review of data replication based on meta-heuristics approach in cloud computing and data grid. Soft Comput 24:14503–14530

5. Mansouri N, Javidi MM (2018a) A hybrid data replication strategy with fuzzy-based deletion for heterogeneous cloud data centers. J Supercomput 74(10):5349–5372
6. Liang B, Dong X, Wang Y, Zhang X (2020) Memory-aware resource management algorithm for low-energy cloud data centers. Future Gener Comput Syst 113:329–342
7. Ardagna D, Panicucci B, Trubian M, Zhang L (2012) Energy-aware autonomic resource allocation in multitier virtualized environments. IEEE Trans Serv Comput 5(1):2–19
8. Kelefouras V, Djemame K (2018) Workflow simulation aware and multi-threading effective task scheduling for heterogeneous computing. In: 25th International Conference on High Performance Computing (HiPC)
9. Mansouri N (2016) QDR: a QoS-aware data replication algorithm for Data Grids considering security factors. Clust Comput 19(3):1071–1087
10. Kang S, Veeravalli B, Aung KMM (2014) ESPRESSO: an encryption as a service for cloud storage systems. In: AIMS 2014, Brno, Czech Republic, pp 15–28
11. Bhattacherjee S, Das R, Khatua S, Roy S (2020) Energy-efficient migration techniques for cloud environment: a step toward green computing. J Supercomput 76:5192–5220
12. Mansouri N (2014) Network and data location aware approach for simultaneous job scheduling and data replication in large-scale data grid environments. Front Comput Sci 8:391–408
13. Mansouri N, Ghafari R, Mohammad Hasani Zade B (2020) Cloud computing simulators: a comprehensive review. Simul Model Pract Theory 104:102144
14. Li C, Zhang J, Tang H (2019) Replica-aware task scheduling and load balanced cache placement for delay reduction in multi-cloud environment. J Supercomput 75:2805–2836
15. Mansouri N, Mohammad Hasani Zade B, Javidi MM (2020) A multi-objective optimized replication using fuzzy based self-defense algorithm for cloud computing. J Netw Comput Appl 171:102811
16. Long SQ, Zhao YL, Chen W (2014) MORM: a multi-objective optimized replication management strategy for cloud storage cluster. J Syst Architect 60:234–244
17. Boru D, Kliazovich D, Granelli F, Bouvry P, Zomaya AY (2015) Energy-efficient data replication in cloud computing datacenters. Clust Comput 18:385–402
18. Kliazovich D, Bouvry P, Khan SU (2012) GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. J Supercomput 62(3):1263–1283
19. Casas I, Taheri J, Ranjan R, Wang L, Zomaya AY (2017) A balanced scheduler with data reuse and replication for scientific workflows in cloud computing. Future Gener Comput Syst 74:1689–2178
20. Manjula S, Indra Devi M, Swathiya R (2016) Division of data in cloud environment for secure data storage. In: International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE)
21. Nivetha NK, Vijayakumar D (2016) Modeling fuzzy based replication strategy to improve data availability in cloud datacenter. In: International Conference on Computing Technologies and Intelligent Data Engineering
22. Tos U, Mokadem R, Hameurlain A, Ayav T, Bora S (2016) A performance and profit oriented data replication strategy for cloud systems. In: International Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress, pp 780–787
23. Mansouri N, Kuchaki Rafsanjani M, Javidi MM (2017) DPRS: a dynamic popularity aware replication strategy with parallel download scheme in cloud environments. Simul Model Pract Theory 77:177–196
24. Li B, Song SL, Bezakova I, Cameron KW (2013) EDR: an energy-aware runtime load distribution system for data-intensive applications in the cloud. In: IEEE International Conference on Cluster Computing, pp 1–8
25. Limam S, Mokadem R, Belalem G (2019) Data replication strategy with satisfaction of availability, performance and tenant budget requirements. Clust Comput 22:1–12
26. Mansouri N, Javidi MM (2018b) A new Prefetching-aware Data Replication to decrease access latency in cloud environment. J Syst Softw 144:197–215
27. Liang L, Xing L, Levitin G (2019) Optimizing dynamic survivability and security of replicated data in cloud systems under co-residence attacks. Reliab Eng Syst Saf 192:106265
28. Sun SY, Yao WB, Li XY (2018) DARS: a dynamic adaptive replica strategy under high load Cloud-P2P. Future Gener Comput Syst 78:31–40

29. He L, Qian Z, Shang F (2020) A novel predicted replication strategy in cloud storage. J Supercomput 76:4838–4856
30. Xue L, Ni J, Li Y, Shen J (2017) Provable data transfer from provable data possession and deletion in cloud storage. Comput Standards Interfaces 54:46–54
31. Ramanan M, Vivekanandan P (2019) Efficient data integrity and data replication in cloud using stochastic diffusion method. Clust Comput 22:14999–15006
32. Antonio Parejo J, Ruiz-Corte's A, Lozano S, Fernandez P (2012) Metaheuristic optimization frameworks: a survey and benchmarking. Soft Comput 16(3):527–561
33. Mahdavi Jafari M, Khayati GR (2018) Prediction of hydroxyapatite crystallite size prepared by sol–gel route: gene expression programming approach. J Sol-Gel Sci Technol 86(1):112–125
34. Eberhart R, Kennedy J (1995) A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pp 39–43
35. Cheng R, Jin Y (2015) A competitive swarm optimizer for large scale optimization. IEEE Trans Cybern 45(2):191–204
36. Luo Y, Che X (2009) Chaos immune particle swarm optimization algorithm with hybrid discrete variables and its application to mechanical optimization. In: □ Third International Symposium on Intelligent Information Technology Application Workshops
37. Cheng R, Jin Y (2014) Demonstrator selection in a social learning particle swarm optimizer, In: IEEE Congress on Evolutionary Computation, pp 3103–3110
38. Wang H, Sun H, Li C, Rahnamayan S, Pan JS (2013) Diversity enhanced particle swarm optimization with neighborhood search. Inf Sci 223:119–135
39. Phan DH, Suzuki J, Carroll R (2012) Evolutionary multi objective optimization for green clouds. In: Annual Conference Companion on Genetic and Evolutionary Computation, pp 19–26
40. Jiang G (2009) Power and performance management of virtualized computing environments via look ahead control. Clust Comput 12(1):1–15
41. Moran MJ, Shapiro HN (1995) Fundamentals of engineering thermodynamics. Wiley, Hoboken
42. Lub L, Chena D, Rend XL, Ming Zhang Q, Cheng Y (2016) Vital nodes identification in complex networks. Phys Rep 650:1–63
43. Hale WK (1980) Frequency assignment: theory and applications. Proc IEEE 68(12):1497–1514
44. Wylie JJ, Bakkaloglu M, Pandurangan V, Bigrigg MW, Oguz S, Tew K, Williams C, Ganger GR, Khosla PK (2001) Selecting the right data distribution scheme for a survivable storage system, Carnegie Mellon University, Technical Report. CMU-CS-01-120
45. Saadat N, Rahmani AM (2012) PDDRA: a new pre-fetching based dynamic data replication algorithm in data grids. Future Gener Comput Syst 28:666–681
46. Jeffrey D, Sanjay G, MapReduce: simplified data processing on large clusters. In: Proceedings of the Conference on Operating System Design and Implementation, pp 137–150
47. Ghemawat S, Gobioff H, Leung ST (2003) The Google file system. ACM SIGOPS Oper Syst Rev 37(5):29–43
48. Shvachko K, Hairong K, Radia S, Chansler R (2010) The Hadoop distributed file system. In: Proceedings of the 26th Symposium on Mass Storage Systems and Technologies, pp 1–10
49. Jararweh Y, Alshara Z, Jarrah M, Kharbutli M, Alsaleh MN (2013) TeachCloud: a cloud computing educational toolkit. Int J Cloud Comput. 2(2):237–257
50. Gupta SKS, Robin Gilbert R, Banerjee A, Abbasi Z, Mukherjeey T, Varsamopoulos G (2011) GDCSim: a tool for analyzing green data center design and resource management techniques. In: International Green Computing Conference and Workshops
51. Nunez A, Vazquez-Poletti JL, Caminero AC, Castane GG, Carretero J, Llorente IM (2012) iCanCloud: a flexible and scalable cloud infrastructure simulator. J Grid Comput 10(1):185–209
52. Fittkau F, Frey S, Hasselbring W (2012) Cloud user-centric enhancements of the simulator CloudSim to improve cloud deployment option analysis. In: Proceedings of the 1st European Conference on Service-Oriented and Cloud Computing
53. Garg S, Buyya R (2011) Networkcloudsim: modeling parallel applications in cloud simulations. In: Proceedings of the 4th IEEE/ACM International Conference on Utility and Cloud Computing, pp 105–113
54. Lim S, Sharma B, Nam G, Kim E, Das C (2009) MDCSim: a multi-tier data center simulation, platform. In: Proceedings of IEEE International Conference on Cluster Computing and Workshops
55. Kecskemeti G (2015) DISSECT-CF: a simulator to foster energy-aware scheduling in infrastructure clouds. Simul Model Pract Theory 58:188–218

56. Teixeira T, Calheiros RN, Gomes DG (2014) CloudReports: an extensible simulation tool for energy-aware cloud computing environments. Cloud Computi, pp 127–142
57. Barroso LA, Clidaras J, Holzle U (2013) The datacenter as a computer: an introduction to the design of warehouse-scale machines, 2nd ed. Morgan and Claypool Publishers
58. Cameron DG, Carvajal-schiaffino R, Paul Millar A, Nicholson C, Stockinger K, Zini F (2003) UK Grid Simulation with OptorSim, UK e-Science All Hands Meeting
59. Wen Y, Xu H, Yang J (2011) A heuristic-based hybrid genetic-variable neighborhood search algorithm for task scheduling in heterogeneous multiprocessor system. Inf Sci 181:567–581

**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.