A dynamic decentralized strategy of replica placement on edge computing

Yingying Yin and Leilei Deng

Abstract
Smart phone and its applications are used more and more extensively in our daily life. Short delay of arriving data is important to these applications, especially to some time-sensitive ones. To reduce transmission latency and improve user experience, a dynamic decentralized data replica placement and management strategy which works in edge nodes is proposed in this article. It studies the location, access frequency, latency improvement, and cost spent on placing replicas on edge nodes to seek a balance between cost spent for storage and reduced latency. Specifically, dynamic and decentralized replica placement strategy algorithm has load guarantee for edge nodes to avoid overload; it dynamically create or delete data replicas on edge nodes according to the request frequency. Dynamic and decentralized replica placement strategy is decentralized to relieve transmission cost. Experiment results show that dynamic and decentralized replica placement strategy algorithm in edge computing environments can greatly reduce transmission latency, balance edge nodes load, and improve system performance. Dynamic and decentralized replica placement strategy also considers the cost spent for storage, and it pursues a balance between many factors.

Keywords
Replica placement strategy, edge computing, resource allocation, computation offloading, smart grid

Date received: 13 February 2022; accepted: 2 June 2022

Handling Editor: Peio Lopez Iturri

Introduction
In the past decades, computer has almost changed the lifestyle of human beings. Nowadays, many new technologies and devices are emerging and developing very fast, such as Internet of Things (IoT), smart devices, wireless network, and cloud computing. They have made a continuous impact to our daily life and work. Many new smart applications, such as smart city, smart campus, and smart transportation, are appearing every day. They often need to exchange data with cloud server instantly. These interactive activities bring much transmission and calculation load to network and data center. Although the network has got great improvements over the last decades, it can never meet the needs. Cloud centers and network can provide neither acceptable transfer rates nor response times for applications. To obtain good performance of applications, replication mechanism is always used to reduce file transfer time and bandwidth consumption. Many companies choose to buy storage services on edge nodes to place replica to keep better response times and transfer rates. So, it is necessary to pay more attention to the strategy which is used to decide when and where to place a replica; a good strategy could get a great improvement of latency with less cost.

In this article, we concentrate on a special case in which requests are growing exponentially, the data are
time-sensitive, and distances between some end users are not far. Several real-life cases are indicated from example 1 to example 3. Example 1 (Figure 1): Decades of students are having class in the same classroom; assume that they need to watch the same video as their teacher asked. They connect to the cloud center and ask for the same data at the same time, the data center should serve every user, process the same requests, and send them the same data. Example 2: Hundreds of people are in the same meeting in several buildings, they ask cloud center for the same video data continuously. Example 3: A short video is very popular in recent days; many end users ask cloud center for this video, and the computing task is very heavy if the cloud center serves every user. In these scenarios, end users need the same data, their requests are time-sensitive, and they asked data almost at the same time, so, it is really a hard and boring work to cloud center. Moreover, many requests need a large amount of data and they are latency-sensitive applications. As we know, data centers are often located near the core network; the applications and services will suffer terrible round-trip latency, when the user receives the data, maybe it is not useful. Even it is useful, the obtaining process is really a waste of resource both for the data center and the network.

So we span our sight to edge nodes which ones are in close proximity of end users. In this case, end users can get the data from edge nodes rather than data center, both the request latency and load of data center will be reduced. Of course, if we put data replicas on all edge nodes, we can get the shortest latency, but it is a waste of resource and almost unrealistic. However, as shown in Figure 1, if we place the replica on critical edge nodes (e.g. node B), we can achieve the goal of reducing the load of data center and reducing the latency at low cost. So, when and where to place a data replica needs to be studied; we expect to spend the least cost in exchange for the biggest gain.

Overall, the contributions of this article are as follows. (1) We proposed a dynamic and decentralized replica placement strategy (DDRP) which is also multi-objective optimized. This decentralized algorithm can make decision much faster than centralized algorithm because it does not need to do a complicate computation to get a global optimal solution; (2) we compared DDRP with other strategies with regard to several metrics; the results showed DDRP could reduce more latency with less cost. The rest of this article is organized as follows. Section “Related works” analyzes the related works. Section “Problem formulation” describes the problem formulation. Section “Strategy and algorithms” presents DDRP algorithm. Section “The simulation results and analysis” shows and analyzes the experimental results. Finally, section “Conclusion” concludes the article and gives some future work.

Related works

In recent years, many scholars have studied on replica placement. Many remarkable results have been achieved. Shao et al. proposed a collaboration-aware data replica placement for IoT workflows in collaborative edge and cloud environments. Chen et al. proposed a centralized data replica placement strategy for cloud environment. Naas et al. proposed a runtime IoT data replica placement strategy in Fog architectures and the iFogSim framework with a heuristic solution. Khanli et al. proposed PHFS which is a dynamic replication method; it considered how to decrease access latency in the multi-tier data grid. Cui et al. presented a data replica placement strategy which is based on genetic algorithm, and this strategy has better performance for scientific applications in cloud environment.

However, some of the existing strategies are centralized or static; although they can yield optimal or near-optimal replica placement strategy by the collection of all replicas and all edge nodes status, the transmission of these controlling data causes a network overhead; the response time is also increased by the complex computation of optimal result. Moreover, the centralized methods also need a single point to be the central replica controller; if it is failure, the whole system failed. Centralized strategy is not popular now because it will take a heavier load to cloud center. There are also some decentralized algorithms. Chen is one of the earliest researchers tries to develop dynamic replica placement strategy. He proposed a dissemination tree for replica placing and data synchronizing. Shen et al. proposed an algorithm which attempts to place replicas on traffic hubs on in a P2P system. Mansouri and Dastghaibyfard introduced a Hierarchical data replication strategy (HDRS); it improved the performance in cloud computing, but did not take any cost considerations. The proposed strategy (DDRP) considers the cloud center load, and it is designed as a decentralized one.
In recent years, many scholars have paid more attention to replica placement strategy in Hadoop Distributed File System (HDFS) because Hadoop is a general platform for distributed storage and has been widely used in many enterprises.\(^{21,22}\) Carlos Guerrero et al. presented a strategy focused on virtualized Hadoop for a coordinated and simultaneous management of file replicas and virtual machines. Yang Liu et al. proposed a dynamic data placement strategy for new replicas to search a best location according to their hotness. This method can adjust data replicas stored on each node in a heterogeneous Hadoop cluster and also can reduce the response time of big data applications dynamically.

Wei et al. presented a cost-based dynamic replication management method called CDRM. It put replicas with the consideration of capacity and the blocking probability of edge nodes. But it did not span their sight into the volume of data.

Atakan Aral proposed a decentralized replica placement algorithm (D-ReP) for edge computing, and this work is the most similar work to ours. D-ReP is based on magnitude and location of user demand and storage price in attempt to reduce data access latency. The authors also provide a replica discovery method with which nearby replicas can be found quickly. It provides best effort latency improvement and cost reduction, but it has not concerned edge nodes load and real-time performance guarantees.\(^{23}\)

All the approaches mentioned above aimed at reducing system latency, providing better service to users. However, most of these strategies ignored the cost need to spend or the load of edge nodes. In this article, we focus on the ratio of cost we spend and latency we reduced to guarantee the cost we spend has high performance.

**Problem formulation**

The aim of our policy is to reduce transmission latency, save storage cost, and balance edge nodes load. The strategy wants to get an optimal value for every factor, but these three factors are not irrelevant, they have conflicts. For example, if replicas are placed on many edge nodes, the transmission latency tends to decrease, but the storage cost and edge nodes load tends to increase. So, we adopt cost-effective ratio to balance the trade-off between conflicting factors. The cost-effective value is the ratio of latency improvements and the cost we spend for storage.

In a real network environment, the situation tends to be more complex. The centralized strategy can get an optimal solution or a close to optimal solution, but the data center must know all replicas status on all edge nodes; the maintenance of the global list will increase much load to network because data center and edge nodes need to exchange much status information. Furthermore, the calculation of the optimal solution is also time-consuming. So, in this article, we adopt decentralized strategy; although it cannot get a global optimal solution, by the leverage of greedy idea, we can get a local optimal solution in a short time and we just need local topology network structure, so that we can meet the real-time requests of time-sensitive applications.

There are two kinds of nodes in our strategy. One is source data node which holds some data replicas, it is also called source node for the sake of brief; the other is edge node which request replicas from data source node, it is also called client node for short. A client node is a neighbor of a source node; when replicas are duplicated to it, it becomes a source node.

To a given source node, we assume that the below listed information in the current epoch can be known before:

1. Request number of each data replica from each client.
2. Transmission latency to each client node.
3. The unit storage cost of each client node.

**Strategy and algorithms**

*The basic idea of DDRP strategy*

The source node monitors requests it received from its client nodes. When the cost-effective value of placing a given replica on a client edge node is equal or bigger than the threshold, the source node sends a little virtual machine to the client node to test the system load. If the client node is not overload, the source node creates a replica on the client node, and then, the client node becomes a new source node. At the same time, the new source node notifies all of its neighbors about this replica so that its neighbors can get replicas from the new source node other than data center; by this way, both the transmission latency and the data center load are reduced. Every source node monitors its performance at a certain frequency. If it is overload, it will also duplicate data replicas to some edge nodes for offloading. If the requests of a given replica fade on a source node, the replica will be migrated to one of its client nodes.

In this strategy, the cost-effective threshold (CET) is the key to make the replica placement decision; if the application is sensitive to time, the value of CET should be small because more storage cost should be spent to reduce the latency, vice versa. The cost-effective value of the system can be changed according to the needs of the applications.

*The notation definition of DDRP*

Data set: \( K = \{ k_1, k_2, \ldots, k_m \} \). On a given data source node, the number of data objects is \( m \).

Source nodes set: \( S = \{ s_1, s_2, \ldots, s_n \} \); \( w \) is the number of source nodes.
Client edge nodes set of current source node: $E = \{e_1, e_2, \ldots, e_n\}$, $n$ is the number of client edge nodes.

The neighbor nodes set to a given client node: $N = \{n_1, n_2, \ldots, n_r\}$, $r$ is the number of neighbors of a given client node.

Requests set: $R = \{Rn(i, k) | k \in K, i \in N \text{ or } i \in E\}$; request number of data object $k$ from node $e$.

Size($k$): The size of data object $k$.

Bandwidth($i, j$): Bandwidth between node $i$ and node $j$.

Unitprice($i$): Storage cost of a unit on node $i$ in the current time.

$\text{Load}(i, j, k)$: Transmission latency for data object $k$ from node $i$ to node $j$

$$TL(i, j, k) = \frac{\text{Size}(k)}{\text{Bandwidth}(i, j)}$$

$\text{SC}(k, i)$: The storage cost for data object $k$ on node $i$ in this epoch

$$\text{SC}(k, i) = \text{Unitprice}(i) \times \text{Size}(k) \times \text{epoch}$$

Load($i$): Load evaluate function of edge node $i$.

The main factors that will affect edge node performance are CPU utilization ($\text{CU}(i)$), memory utilization ($\text{MU}(i)$), disk space usage ($\text{DU}(i)$), and I/O access rate ($\text{IOR}(i)$). The edge nodes loads can be obtained by equation (3)

$$\text{Load}(i) = w_1 \text{CU}(i) + w_2 \text{MU}(i) + w_3 \text{DU}(i) + w_4 \text{IOR}(i)$$

In the above equation, $\text{Load}(i) \in [0, 1]$ and $w_1, w_2, w_3, w_4$ represent the weight coefficients of the four metrics; they indicate the importance of each indicator on the edge node load and their sum is 1, namely

$$\sum_{i=1}^{4} w_i = 1$$

The analytic hierarchy process is used to calculate the weight coefficients.

Cost-effective value

As mentioned above, cost-effective is a ratio, and it indicates the trade-off between latency improvement and storage cost to place a given replica $k_i$ on a given client edge node $e$. The cost-effective can be calculated by equation (2), as follows

$$CE(e, k_i) = \frac{\left[Rn(k_i, e) + \sum_{n_j \in E} Rn(k_i, n_j)\right] \times TL(s, e, k_i)}{\text{SC}(k_i, e)}$$

where $Rn(k_i, e)$ is the request frequency of data object $k_i$ on the given client node $e$, $Rn(k_i, n_j)$ is the request frequency of data object $k_i$ on neighbor nodes of the given client node $e$, $TL(s, e, k_i)$ is the transmission latency of data object $k_i$ from source node to the given edge node $e$, and $\text{SC}(k_i, e)$ is the storage cost of data object on the given client node $e$ in this epoch.

Discovery and retirement of replica

As we mentioned above, every source node in the system knows the data request number and storage cost of all its neighbor nodes, so we keep two tables on each source node: one is Neighbor Table (NT) which is used to store the neighbor nodes information, such as unit storage cost and latency; and the other is Replica Table (RT) which is used to store its replica information, such as replica ID, size, expire time, and stored node. Every client node will only keep one table, RT. When a data replica is created or deleted on node $e$, $e$ will send message to all neighbors to renew RT; when the storage cost or latency between $e$ and its neighbors changed, NT will be renewed, too.

So, when a client node $e$ requests for a given data, if the requested data is on one of its neighbors, for example, $s$, $e$ will send request to $s$ asking for the data by the help of RT, if the requested data is not listed on RT, the request will be sent to data center. There would be another case that the requested data are on one node $m$ which is on the path from $e$ to cloud center; under this circumstance, $m$ answers the request.

There are three cases that replica $k$ on source node $s$ will retire. The first case is $k$ has expired; the second case is that the requests of $s$ are fading; the third case is that the node $s$ is overload. For the first case, $k$ should be deleted because it is not useful; for the other cases, if the requests from client nodes $s$ are not fading, $k$ should be migrated from $s$ to one of its client nodes and then deleted from node $s$.

To guarantee the service quality, node $s$ will finish existing access action, then sends message to notify all its neighbors about the deletion of $k$ before the delete operation. Each neighbor will send a response message after it received the delete notification; when $s$ receives all the responses, the real delete operation will be processed. At the same time, its neighbors renew their RTs and turn their requests to other nodes.

An implement example of DDRP

We use an example to illustrate DDRP strategy. As shown in Figure 2. When node $C$ receives many requests of the data object $k$ and it is not overloaded, data center creates a replica on $C$, then $C$ notify its neighbors, namely node $A, B, E, F$ about the replica; after that, if any one of $A, B, E$ or $F$ has the same request, they send request to $C$. When $C$ serves so many neighbors that the value of Load($C$) is bigger than the threshold, $C$ duplicates $k$ to one of its
neighbor who has the max cost-effect value, assume it is \( E \), then \( E \) serves \( F \) and its own user directly. When the requests from users of \( C \) are fading to 0, \( C \) will also duplicate \( k \) to one of its neighbor who has the max cost-effect value, then, remove \( k \) out of its storage.

The simulation results and analysis

**Settings of experiments**

To evaluate the performance of our strategy comprehensively, we conducted decades of experiments with different settings to analyze and compare our strategy with four existing strategies. Some of them are single cloud without edge nodes. Some of them are Cloud-Edge systems. We used CloudSim to simulate the edge computing network and test our strategy in different scenarios (S1–S20), as shown in Table 1. The key parameters we have considered are listed in Table 2.

Since network latency and request number both cannot be predicted accurately, we set them to random integer values in a range. The epoch time is 10 in every scenario. First, we test our strategy with different number of edge nodes in S1–S10, and the results of LIR are shown in Figure 3(a); we can deduce that LIR do not have significant fluctuation with different edge nodes number. So, our strategy is stable, in the subsequent experiment, and we do not test our strategy with different number of edge nodes, just set its value as 50. Then, we test the impaction of CET, and the result is shown in Figure 3(b); we can deduce that when CET is increasing, the LIR is decreasing, that means, if you seek high-cost performance, you will spend less cost, so the LIR will decrease.

**Comparison of DDRP performance with PHFS, MORM, and HDRS**

We compared DDRP with PHFS, MORM, and HDRS with different request numbers and data size, and the results are depicted in Figure 4. In Figure 4, we compared the average response time of four strategies. Multi-objective Optimized Replication Management strategy (MORM’s) value is much higher than the other three. Since it is static, the replicas are generated according to average latency and the computing resource of cloud center. DDRP, Predictive Hierarchical Fast Spread (PHFS), and HDRS are dynamic; they create replicas based on request numbers, so their values are close. DDRP’s value is little lower than the other two.

**Comparison of DDRP performance with D-Rep**

We also compared DDRP with decentralized algorithm named D-Rep. D-Rep is a decentralized algorithm, it also considers cost and network latency as our strategy. We compared DDRP with D-Rep in different scenarios (S1–S20), as shown in Table 1. In our system, the storage and transmission cost is according to Amazon S3 price rules, the storage unit is 128 KB, the size of each data block is a random multiple times of 128 KB; there are 30 data blocks in every data center. To get more accuracy experiment results, we set the neighbor

| Scenario | Network latency range (ms) | Request numbers range | CET | NLT |
|----------|---------------------------|-----------------------|-----|-----|
| S1       | 0–50                      | 4001–6000             | 1.5 | 0.7 |
| S2       | 51–100                    | 4001–6000             | 1.5 | 0.7 |
| S3       | 101–150                   | 4001–6000             | 1.5 | 0.7 |
| S4       | 151–200                   | 4001–6000             | 1.5 | 0.7 |
| S5       | 201–300                   | 4001–6000             | 1.5 | 0.7 |
| S6       | 101–150                   | 0–1000                | 1.5 | 0.7 |
| S7       | 101–150                   | 1001–2000             | 1.5 | 0.7 |
| S8       | 101–150                   | 2001–4000             | 1.5 | 0.7 |
| S9       | 101–150                   | 4001–6000             | 1.5 | 0.7 |
| S10      | 101–150                   | 6001–10000            | 1.5 | 0.7 |
| S11      | 51–100                    | 4001–6000             | 0.5 | 0.7 |
| S12      | 51–100                    | 4001–6000             | 1   | 0.7 |
| S13      | 51–100                    | 4001–6000             | 1.5 | 0.7 |
| S14      | 51–100                    | 4001–6000             | 2   | 0.7 |
| S15      | 51–100                    | 4001–6000             | 2.5 | 0.7 |
| S16      | 51–100                    | 4001–6000             | 1.5 | 0.5 |
| S17      | 51–100                    | 4001–6000             | 1.5 | 0.6 |
| S18      | 51–100                    | 4001–6000             | 1.5 | 0.7 |
| S19      | 51–100                    | 4001–6000             | 1.5 | 0.8 |
| S20      | 51–100                    | 4001–6000             | 1.5 | 0.9 |

CET: cost-effective threshold; NLT: node load threshold.

| Parameters | Illustration |
|------------|--------------|
| Network latency | The average latency in an epoch |
| Edge node number | Edge node is the site to store replicas |
| Request number | The number of end user’s requests |
| NLT | Node load threshold |
| CET | Cost-effective threshold |
| LIR | Latency improvement rate |
number as 25, 50, 75, and 100, respectively; for each
node number, we run the simulated program 20 times
and calculate the average value.

Figure 5 is the comparison of LIR cost between
DDRP and D-Rep in S1–S5. These scenarios have dif-
ferent network delays, but other parameters are the
same. The result indicates that DDRP can achieve
lower LIR than D-Rep, especially when the network
delay is serious. That means, DDRP performed better
than D-Rep in bad network occasion, and it can spend
less cost while reducing more latency.

In S6–S10, the request number is different. Figure 6
is the comparison between different ranges of request
number; we can conclude that when the requests are
bursting out, DDRP has more LIR and less cost.

Algorithm 1. Algorithm for edge nodes.

1 Input: data object set K, current source node s, edge node
set E, neighbor nodes set N, RT and NT of s, edge node load
threshold \( \alpha \), cost-effective threshold cf
2 Output: operations to be executed
3 Construct a cost-effective function \( CE(k, a) \)
4 Construct an edge node load evaluate function \( Load(a) \)
5 //place replica
6 for (i=1; i<=count(K), i++)
7 {while((k \in K) \&\&(e, k, not \in NT))
8 {if \( CE(k, e) \geq cf \) \&\& \( Load(e) < \alpha \)
9 Duplicate (k, e);
10 Add (e, k, RT);
11 Break;
12 }
13 }
14 //duplicate replica when s is overload
15 Do while (\( Load(s) \geq \alpha \))
16 for (i=1;i<=count(E), i++)
17 if ((e, k, not \in NT))
18 {if \( CE(e, k) = \max(CE(E, k)) \)
19 Duplicate (k, e);
20 Add (e, k, RT);
21 }
22 //delete replica when the requests fade
23 for (i=1;i<=count(E), i++)
24 if (\( CE(s, k) < \text{low}_c \))
25 {Notify (e, k);
26 if (received(e, E);
27 Remove (k, s);
28 Break;
29 }
30 }
In S11–S15, the network status is normal and the request number is normal too. We compared DDRP and D-Rep with different CETs, and the results are shown in Figure 7. When the CET value is between 0.5 and 2.5, LIR of DDRP is more than D-Rep, and cost of DDRP is much less than D-Rep, which means DDRP can reduce much latency with less cost.

In S16–S20, the network load is a little heavier, but it is stable. Figure 8 indicates the comparison results of DDRP and D-Rep with different node load thresholds; when the node load is less than 0.7, the LIR of DDRP is less than D-Rep, but the cost DDRP spent is much less than D-Rep, and, when the node load is equal or more than 0.7, the LIR of DDRP is much more than D-Rep, which means DDRP can be more effective when the edge node is overload.

From all above experimental results we can conclude that DDRP algorithm can reduce transmission latency, balance edge nodes load, and get higher cost-effective value; it has better performance in request bursting out case and edge nodes overload case.

**Conclusion**

The network is everywhere and it has been used by everything. The edge-cloud application is becoming the mainstream of future software. So, the replica placement strategy is concerned by more researchers since it is a good solution of long latency and low performance of the system. In this article, we have done a lot of study on replica placement strategies in edge-cloud environment and proposed DDRP in which replicas are placed on edge nodes based on cost-effective value; DDRP has also considered the load of edge nodes because heavy load will not reduce system latency but may increase it. The experimental results indicated that compared to D-Rep, our strategy...
can get more latency improvement with less cost and edge nodes will not overload. In the future, we will continue extending DDRP with real-time performance guarantees.

Besides the development of edge intelligence, the replica placement strategy can be interconnected together with the distributed intelligence for offloading intensive tasks generated by a large number of end users to idle edge servers. In the future, we intend to apply replica placement principle to place distributed machine learning model on edge nodes to reduce the computing tasks on cloud and take full use of computing capabilities on edge nodes.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by Jilin Provincial Department of Science and Technology (Grant No. 202101011853C). The manuscript was also improved thanks to China Scholarship Council (CSC; Grant No. 2018(10038)) and Dr Liqiang Wang who is the director of big data laboratory in Computer Science Department of University of Central Florida.

ORCID iD

Yingying Yin https://orcid.org/0000-0001-8820-8030

References

1. Taleb T, Dutta S, Ksentini A, et al. Mobile edge computing potential in making cities smarter. *IEEE Commun Mag* 2017; 55(3): 38–43.

2. Qi B, Kang L and Banerjee S. A vehicle-based edge computing platform for transit and human mobility analytics. In: *Proceedings of the ACM/IEEE symposium on edge computing*, 2017, pp.1–14, https://cse.buffalo.edu/faculty/tkosar/cse710_spring20/qi-sec17.pdf

3. Shi W, Cao J, Zhang Q, et al. Edge computing: vision and challenges. *IEEE Intern Things J* 2016; 3(5): 637–646.

4. Saxena A. Report stating the concept of cloud computing testing and its challenges. *Int J Recent Innov Trend Comput Commun* 2020; 8(1): 11–14.

5. Nukarapu DT, Tang B, Wang L, et al. Data replication in data intensive scientific applications with performance guarantee. *IEEE Trans Parall Distrib Syst* 2011; 22(8): 1299–1306.

6. Mohiuddin M, Saab W, Bluudze S, et al. Axo: detection and recovery for delay and crash faults in real-time control systems. *IEEE Trans Ind Inform* 2018; 14(7): 3065–3075.

7. Prabhakaran S, Neumann M and Wolf F. Efficient fault tolerance through dynamic node replacement. In: *Proceedings of the 2018 18th IEEE/ACM international symposium on cluster, cloud and grid computing*, Washington, DC, 1–4 May 2018, pp.163–172. New York: IEEE.

8. Rathi R and Gupta N. Game theoretic and non-game theoretic resource allocation approaches for D2D communication. *Ain Shams Eng J* 2021; 12: 2385–2393.

9. Shao Y, Li C and Tang H. A data replica placement strategy for IoT workflows in collaborative edge and cloud environments. *Comp Netw* 2019; 148: 46–59.

10. Chen CA, Won M, Stoleru R, et al. Energy-efficient fault-tolerant data storage and processing in mobile cloud. *IEEE Trans Cloud Comput* 2015; 3(1): 28–41.

11. Naas MI, Parveyd PR, Boukhobza J, et al. iFogStor: an IoT data placement strategy for fog infrastructure. In: *Proceedings of the IEEE 1st international conference on fog and edge computing (ICFEC)*, Madrid, 14–15 May 2017, pp.97–104. New York: IEEE.

12. Khanli LM, Isazadeh A and Shishavan TN. PHFS: a dynamic replication method, to decrease access latency in the multi-tier data grid. *Fut Gener Comput Syst* 2011; 27: 233–244.

13. Rochman Y, Levy H and Brosh E. Resource placement and assignment in distributed network topologies. In:
14. Lin JW, Chen CH and Chang JM. QoS-aware data replication for data-intensive applications in cloud computing systems. *IEEE Trans Cloud Comput* 2013; 1(1): 101–115.

15. Tabet K, Mokadem R, Laouar MR, et al. Data replication in cloud systems: a survey. *Int J Inform Syst Soc Change* 2017; 8(3): 17–33.

16. Liu G, Shen H and Chandler H. Selective data replication for online social networks with distributed data centers. In: *Proceedings of the 21st IEEE international conference on network protocols (ICNP)*, Göttingen, 7–10 October 2013, pp.1–10. New York: IEEE.

17. Pantazopoulos P, Karaliopoulos M and Stavrakakis I. Distributed placement of autonomic Internet services. *IEEE Trans Parall Distrib Syst* 2014; 25(7): 1702–1712.

18. Smaragdakis G, Laoutaris N, Oikonomou K, et al. Distributed server migration for scalable internet service deployment. *IEEE/ACM Trans Netw* 2014; 22(3): 917–930.

19. Chen Y, Katz RH and Kubiatowicz JD. Dynamic replica placement for scalable content delivery. In: *Proceedings of the international conference on peer-to-peer systems*, Cambridge, MA, 2002, pp.306–318, https://oceanstore.cs.berkeley.edu/publications/papers/pdf/dtree_iptps.pdf

20. Mansouri N and Dastghaibyfard GH. Enhanced dynamic hierarchical replication and weighted scheduling strategy in data grid. *J Parall Distrib Comput* 2013; 73(4): 534–543.

21. Medhat D, Yousef AH and Salama C. Cost-aware load balancing for multilingual record linkage using MapReduce. *Ain Shams Eng J* 2020; 11: 419–433.

22. Guerrero C, Lera I, Bermejo B, et al. Multi-objective optimization for virtual machine allocation and replica placement in virtualized Hadoop. *IEEE Trans Parall Distrib Syst* 2018; 29(11): 2568–2581.

23. Aral A and Ovatman T. A decentralized replica placement algorithm for edge computing. *IEEE Trans Netw Serv Manage* 2018; 15(2): 516–529.