IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 9 November 2023, accepted 12 January 2024, date of publication 2 February 2024, date of current version 12 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3361748

RESEARCH ARTICLE

Efficient Multi-Cloud Storage Using Online Dynamic Replication and Placement Algorithms for Online Social Networks

ALI Y. ALDAILAMY[1], ABDULLAH MUHAMMED[1], NOR ASILAH WATI ABDUL HAMID[1], (Senior Member, IEEE), ROHAYA LATIP[1], AND WAIDAH ISMAIL[2]

[1]Department of Communication Technology and Networking, Faculty of Computer Science, Institute for Mathematical Research, Universiti Putra Malaysia, Serdang, Selangor 43400, Malaysia
[2]Faculty of Science and Technology, Universiti Sains Islam Malaysia, Nilai, Negeri Sembilan 71800, Malaysia

Corresponding authors: Ali Y. Aldailamy (a.aldailamy7@gmail.com) and Abdullah Muhammed (abdullah@upm.edu.my)

This work was supported by the Malaysian Ministry of Higher Education under Grant FRGS/1/2021/ICT08/UPM/02/2.

ABSTRACT The provision of Storage as a Service (STaaS) in many geo-distributed datacenters by several Cloud Storage Providers (CSPs) has made online cloud storage a great choice for replicating and distributing objects that are accessed worldwide. Online Social Networks (OSN) such as Facebook and Twitter have billions of active users worldwide accessing shared objects. These users expect to access these objects within a tolerable time. To minimize users' access latency time of these objects, OSN service providers must host several replicas of objects in many datacenters. However, this replication process produces a higher monetary cost. This paper addresses crucial issues, including how many replicas are required to fulfil the expected workload of the object and the optimal datacenters to host these replicas to reduce latency time for users and monetary costs for OSN service providers. Two online algorithms are proposed to determine the suitable number of replicas for each object and the optimal placement of these replicas. The DTS algorithm establishes the replication and placement of objects using deterministic time slots, while the RTS algorithm is based on randomized time slots. Experimental results show the effectiveness of the proposed algorithms for producing latency time below certain thresholds and reducing the monetary cost.

INDEX TERMS Dynamic replication, latency and cost optimization, multi-cloud, online placement algorithm, online social network, storage as a service.

I. INTRODUCTION

According to the Digital report [1], in 2023, about 5.16 billion people now have regular access to the internet, whereas 4.76 billion are active social media users. More than 3.5 billion of those social media active users are from Facebook (2.96 billion active users every month) and Twitter (556 million active users monthly). With this massive number of users, Facebook generates about 4 petabytes of data daily [2], and Twitter receives every day about 762 million tweets [3]. This data is generated and shared by users around the world.

The associate editor coordinating the review of this manuscript and approving it for publication was Barbara Guidi.

These users also require good Quality of Service (QoS) from the Online Social Networks (OSN) service providers. One of the most important QoS is the latency time of accessing objects in the OSN. On the other hand, the main goal of moving data into the cloud is reducing the capital expenditures needed to build, develop, and maintain the required hardware infrastructures and avoiding the datacenters management complexities (IT maintenance cost). Therefore, OSN service providers are constantly struggling to reduce the expenses for hosting and managing these large-scale data without affecting the QoS requirements of their users.

Giant Cloud storage providers (CSPs) such as Google, Amazon, and Microsoft have provided storage as a service

(STaaS) based on a pay-as-you-go model [4], [5], [6]. Using STaaS is more cost-efficient than building a private storage infrastructure. Moreover, the STaaS of giant CSPs has geo-distributed datacenters around the world. Therefore, many enterprises have moved and replicated their data among these datacenters to increase availability and reduce access latency.

However, Milani and Navimipour in [7] and Wu et al. in [8] have stated that replicating objects among multiple datacenters increases the overhead of replication management and results in higher hosting costs. Using full static replication of OSN objects worldwide to ensure high availability and reduce access latency is exceptionally costly. For example, generating a static number of replicas and distributing them among selective datacenters that are scattered worldwide. On the other hand, the local replication of OSN objects can reduce the cost to a certain extent but increases latency time worldwide. For example, replicating the object in two datacenters of the same region where the object was created will direct all the worldwide requests to this region. Therefore, OSN service providers always consider the trade-off between monetary cost and access latency to produce the most affordable system.

The dynamic nature of objects in OSN has made it unfeasible to use static replication and placement strategies [9]. The popularity of the object in the OSN also has this dynamic nature. Hence, the popularity calculation process should be invoked regularly to adjust the replication and placement of the object in the OSN based on its current popularity. So, when the object is in hot popularity in the OSN, it should have many replicas to be capable of serving the expected workload. However, when this object becomes cold, its number of replicas should be minimized in order to save cost. This adjustment process should take into consideration the accepted latency time for the users of the OSN and the monetary cost for the OSN service providers.

Gill and Singh in [10] and Sun et al. in [11] used only the access rate to determine the suitable popularity of each object at each specific time slot. This popularity is then used to determine the appropriate number of replicas for the object. However, Facebook and Twitter OSNs have more criteria that are crucial in determining the popularity of an object. Besides the access rate, these criteria include Shares, Comments, and Likes numbers, which are called the engagement rate of the object in the OSN. They are also considered as the Put requests of the object.

On the other hand, users expect QoS from the OSN service provider, which includes unnoticeable latency time. Users of OSN expect to access the requested object and to see their engagement within a certain threshold. Liu et al. [12] stated that the typical latency time for data retrieval on the web is 100MS. However, in OSN, latency time up to 250MS of Put/Get requests is considered unnoticeable for the users [13]. Latency time of more than 250MS leads to users becoming frustrated, and users are likely to leave the OSN [14].

Fulfilling users' requirement of latency time and OSN service provider's requirement of monetary cost requires optimization in the replication and placement of objects to provide OSN that guarantees the expected latency of users and produces the lowest possible monetary cost for the OSN service provider. Therefore, a popularity-based approach is required to consider the object's popularity to dynamically determine the suitable number of replicas to produce the minimal size while fulfilling the expected workload. Furthermore, a region ranking-based placement that always places the object in the datacenters with the lowest latency time and cost and dynamically changes this placement according to the regions with the highest ranks of the object is also required. To focus on these points, this study produces the following key contributions:

- Proposing a region ranking-based strategy that optimizes the placement of the object replicas based on object access and engagement rates of the regions.
- Proposing two online algorithms to guarantee an unnoticeable latency time of less than 250ms for users of OSN and producing the lowest monetary cost for OSN service providers based on deterministic and randomized time slots.
- Besides providing the mathematical model and analysis, a comprehensive performance evaluation of the proposed algorithms using synthetical workloads based on a real Facebook dataset is provided by the CloudSim simulator.

The remainder of this paper is organized as follows. In section two, the studies related to this work are reviewed. Section three introduces the problem definition and mathematical models. The static replication and placement are presented in section four. In section five, the latency and cost optimization and the proposed online algorithms are presented. Section six presents the simulation experiments and performance evaluation of the proposed algorithms. Finally, we conclude this study in the last section.

II. RELATED WORK

One of the challenging issues for OSN service providers is delivering an acceptable access latency time for users while minimizing the cost of hosting objects in the cloud. In recent years, this topic has gained wide attention from researchers, who proposed several replication strategies to enhance object availability, placement strategies to host object replicas in the appropriate datacenters, and retrieval techniques to decrease access latency time. All these strategies can be classified into the following categories.

A. FULL REPLICATION

This replication concentrates on replicating the objects in all datacenters related to users or selective datacenters that are scattered around the world. The full replication of the objects of all friends in all user's datacenters, as proposed in [15] and [16] can reduce the latency time of users. However, it is not feasible to replicate hot and cold objects in all datacenters because of higher replication costs. Other researchers, such as [12] selected specific datacenters scattered around the

world to minimize inter-datacenter communications while obtaining minimal latency time. Another research is [17], in which a geo-partitioning strategy is proposed to reduce cost while preserving latency time below thresholds in OSNs. The replicas in the geo-distributed datacenters that satisfy this threshold are kept while the others are deleted. These strategies use static replication that replicates all objects, whether they are popular or unpopular, and produce expensive synchronization. Furthermore, using full replication for OSN objects to reduce latency, guarantee availability, and meet other requirements is incredibly expensive.

B. DYNAMIC REPLICATION

This replication focuses on deciding the suitable number of replicas for an object based on certain criteria, such as the access rate of the object. Khalajzadeh et al. [9] introduced a graph partitioning algorithm that divides OSN graphs into different partitions that connect friends. Although the proposed algorithm can reduce user access latency, it causes high storage and traffic costs. A two-layer Geo-cloud-based dynamic replication algorithm in large OSN such as Facebook was presented by Ye et al. [18]. The algorithm lowers the bandwidth consumption and the mean access time by considering site capacity. It also ranks objects based on popularity and replicates the most popular. A different study by Mansouri et al. [19] proposed a data replication technique based on the 80/20 principle that dynamically replicates objects based on the number of accesses to enhance data availability and latency time. Moreover, multi-objective replication management using the immune algorithm is designed by Long et al. [20] to optimize several factors, including latency time, by deciding the suitable number of replicas for each object. All these studies conducted dynamic replication based on the access rate only. Contrary to our research, the dynamicity used by all these studies to determine the replicas' number is solely based on the access rate. In OSN, the access rate alone is insufficient to predict the object's future access, as the engagement rate also plays an essential role.

C. COST-LATENCY TRADE-OFF

Full static replication of OSN objects around the world to ensure high availability and reduce latency is prohibitively expensive. The local replication of OSN objects can save costs but increase the access latency for global requests. To create a cost-effective system, OSN service providers always examine the trade-off between monetary cost and latency time. Matt et al. [22] proposed a storage service ranking function focusing firstly on cost and secondly on the latency of the download and upload requests. They used the erasure coding replication, which is not cost-effective when used with read-intensive objects, as illustrated in [23]. A placement that struggles to keep the latency time below a predetermined threshold while optimizing the cost by taking advantage of the dynamic set cover problem is proposed by Khalajzadeh et al. [24]. Another study by Han et al. [25] proposed an adaptive placement of objects in OSN hosted by multiple CSPs to improve performance. The approach makes decisions of intelligent migration of objects based on their traffic to provide acceptable latency time. Furthermore, to reduce transmission time and bandwidth cost, they presented a Lagrange relaxation-based data placement algorithm that takes into consideration each datacenter capacity and the load balance of the geo-distributed datacenters. By utilizing the erasure coding replication method with a modified Paxos, PANDO [26] focused on ensuring data consistency and reducing read/write latency time across geo-distributed datacenters. In general, these works can be orthogonal to our work; however, the Cost-latency trade-off of some works is considered to be offline as the workload of the object is known in advance, while other solutions are based on access rate only.

D. REDUNDANT REQUESTS

Another mechanism to reduce the latency time of requests is to send multiple requests to all the datacenters that host the replicas. CosTLO [27] generated two redundant requests to the nearest datacenters to reduce object transmission latency. The first request retrieves the object is the request used. Using a greedy algorithm to decide the number of redundant requests and their destinations based on historical datacenter latency performance, Cui et al. [28] proposed TAILCUTTER, a request scheduling mechanism to reduce object request latency while meeting the customers' monetary cost limitations. Other researchers, such as [29] proposed a redundant requests mechanism that issues multiple requests until the fastest response is received with the ability to cancel other requests. However, the cancellation of the running requests produces significant delays. In general, generating multiple redundant requests of objects to various datacenters hosting replicas incurs additional transmission costs.

E. SPLIT REQUESTS

Partitioning the object on multiple datacenters and retrieving the portions in parallel by splitting the request is another strategy used to improve access latency. GeoCol [30] optimized latency using the request split method to predict the request latency time and storage planning method to decide whether the object should be stored and which storage type. A request splitting mechanism that decides whether to split the request into two sub-requests based on the latency performance of the last request to the same datacenter proposed by Sun et al. [31]. Another research by Hajjat et al. in [32] proposed Dealer, which builds a global view by monitoring the latency performance and reduces the latency time by dynamically choosing the best combination of datacenters to serve user requests based on this historical view. Different request splitting mechanism by Hu and Niu [33] reduced the latency from the load balancing perspective by adjusting the block placement based

TABLE 1. Notation of symbols and their meaning.

Variable	Meaning	Variable	Meaning
A	*Set of objects in the OSN.*	D	*Set of available datacenters.*
a	*a^{th} object, $a \in [1-A]$.*	d	*d^{th} datacenter, $d \in [1-D]$.*
G	*Set of available regions.*	g	*g^{th} region, $g \in [1-G]$.*
t_s, t_i, t_p, T	*The first, i^{th}, and current time slots, $t_i \in [t_s - t_p]$, t_s, t_i, and $t_c \in [0-T]$.*	$RE^a_{t_i}(d)$	*The existence of object replica on d in t_i, $RE^a_{t_i}(d) \in [0,1]$.*
$RN^a_{t_i}$	*The replica number of a in t_i.*	$e^{-\sigma(t_c - t_s)}$	*The EDF from t_c to t_s.*
$AR^a_{t_i}$	*The access rate of a in t_i.*	$SZ^a_{t_i}$	*The size of a in t_i.*
$GR^a_{t_i}$	*The number of Get requests a received in t_i.*	$SR^a_{t_i}$	*The number of Share requests a received in t_i.*
$CR^a_{t_i}$	*The number of Comment requests a received in t_i.*	$LR^a_{t_i}$	*The number of Like requests a received in t_i.*
$ER^a_{t_i}$	*The engagement rate of a in t_i.*	$OP^a_{t_i}$	*The popularity of a in t_i.*
$GR^a_g(t_i, t_{i-1})$	*The number of Get requests a received in t_i from g.*	$LR^a_g(t_i, t_{i-1})$	*The number of Like requests a received in t_i from g.*
$CR^a_g(t_i, t_{i-1})$	*The number of Comment requests a received in t_i from g.*	$SR^a_g(t_i, t_{i-1})$	*The number of Share requests a received in t_i from g.*
$\beta_s, \beta_c, \beta_l$	*The weight variable of Share, Comment, and Like respectively.*	$PR^a_{t_i}$	*The number of Put requests a received in t_i.*
SC_{t_i}	*The total storage cost of objects A in the t_c.*	$USP(d)$	*The storage price of size unit in d.*
RC_{t_i}	*The total cost of Get and Put requests in t_i.*	$UGP(d)$	*The price of Get request in d.*
$UPP(d)$	*The price of Put requests in d.*	NC_{t_i}	*The total network cost in t_i.*
$DSC_{t_i}(d)$	*The total retrieved data size of Get requests through the network of d in t_i.*	$UNP(d)$	*The network price for size unit in d.*
L	*The latency of each request.*	RL^g_d	*The average latency from g to d.*
TPL_{t_i}	*The percentile latency time of all requests to A on D in t_i.*	P	*The required percentile of all requests.*
LN_{t_i}	*The number of the measured latencies L of all Get requests in t_i.*	$DC^a_{t_i}$	*The selected subset of the datacenters to host the replicas of a in t_i.*
$Gl^a_{t_i}$	*A list of the regions with highest ranks for a in t_i.*	$W^a_{t_i}$	*The window size of a in t_i used by RTS algorithm.*
TC_{t_i}	*The total cost of hosting A in t_i.*	$FDCL$	*The list of datacenters used by SDR algorithm.*
h_a	*The random time unit for a.*	UR_a	*The region of a owner.*
$TPS^a_{t_i}$	*The time slot size of a in t_i.*	TL	*The size limit of time slot.*

on recent request history using controlled migration to reduce the migration overhead. Nonetheless, dividing an object into multiple portions hosted in multiple datacenters requires reconstructing the object from multiple datacenters, which incurs significant delays. Moreover, this technique does not allow the utilization of the cheapest datacenter.

None of the studies reviewed above provides research that dynamically decides the placement of the replicas based on each region's access and engagement rates to optimize the access latency while reducing the monetary cost as much as possible. As a result, we present two online algorithms that employ dynamic replication and placement strategies to determine the number of replicas of the object based on its popularity in the OSN and the datacenters that should host these replicas based on the region's rank of the object at each time slot of its lifetime in the OSN. Furthermore, simulation-based experiments show that our suggested algorithms can preserve the access latency below the maximum expectation of the user without the requirement to have future access and engagement in advance.

III. MODELS

OSN, such as Facebook and Twitter, has a vast number of users scattered around the world, and large-scale data is generated daily. Placing the object in specific regions with static replication causes a higher latency time. For example, if a user has posted an object, it will be replicated and hosted in the same region as the user. As this object is accessible to all other users around the world, it may be accessed from another region more than the region of the object user. This can cause a higher latency time.

Furthermore, creating many replicas of all objects is not feasible as the overhead of maintaining a high number of replicas increases, and the cost of replication rises above what is reasonable. So, creating many replicas of all objects is not a good idea. Thus, it makes sense only to calculate

the popularity of each object based on its access and engagement rate at each time slot of its lifetime in the OSN. This popularity is used to determine the suitable number of replicas for the object in the OSN. Additionally, the popularity of the object in each region is used to decide the suitable datacenters that host these replicas.

In this study, we tackle the issue of dynamic replication and placement of OSN objects in geo-distributed datacenters with the guarantee of latency requirements for OSN's users and the minimization of monetary cost for OSN's service providers. The dynamic replication is handled based on the object workload, and the dynamic placement is conducted based on each object's popularity in each region. All these mechanisms should be conducted online without prior knowledge of the future workload for the object. For convenience, we list the symbols for the notation used in this paper in Table 1.

A. SYSTEM MODEL

In this section, the objects, the datacenters, and the replica existence matrix of each object in each datacenter are introduced using the following definitions:

Definition 1 (Datacenters): Datacenters are represented as a set of D, which are geographically distributed around the world. Every datacenter $d \in D$ is assigned to a region g, where $g \in G$, every subset of datacenters is located in a region g. For example, in Germany, Google Cloud has one datacenter, Amazon S3 also has one datacenter, and Microsoft Azure has two datacenters. Therefore, in Germany region, four datacenters belonging to different CSPs can be used to host objects.

Definition 2 (Objects): The system model represents the object as a set of A, where each object $a \in A$. Each object has size $SZ^a_{t_i}$, number of Get requests $GR^a_{t_i}$, Shares $SR^a_{t_i}$, Comments $CR^a_{t_i}$ and Likes $LR^a_{t_i}$ at each time slot t_i, where $t_i \in [t_s - t_c]$. In addition, at each time slot t_i, object a has a specific number of replicas $RN^a_{t_i}$ hosted in certain datacenters.

Definition 3 (Objects Replica Existence): The replicas' existence of the objects set A in the datacenters set D in time slot t_i is represented as a matrix of $(A \times D)_{t_i}$ as shown in Fig. 1. The replica existence value in the matrix for object a at time slot t_i in datacenter d is denoted by $RE^a_{t_i}(d) \in [0, 1]$. Where $RE^a_{t_i}(d) = 1$ indicates that object a has replica in datacenter d at time slot t_i. Whereas $RE^a_{t_i}(d) = 0$ indicates that object a has no replica in datacenter d at time slot t_i. The number of replicas $RN^a_{t_i}$ for object a in time slot t_i is the summation of the $RE^a_{t_i}(d)$ of all datacenters.

$$RN^a_{t_i} = \sum_{d=1}^{D} RE^a_{t_i}(d) \tag{1}$$

B. DYNAMIC REPLICATION MODEL

In order to determine the suitable number of replicas for the object a in the current time slot t_c, the object popularity is required to be calculated using the decaying function based on the four important tuples: numbers of Gets, Shares, Comments, and Likes. This dynamic replication model is presented using the following definitions.

	datacenter_1	datacenter_2	.....	datacenter_D
1	$RE^1_0(1)$	$RE^1_0(2)$	.....	$RE^1_0(D)$
2	$RE^2_0(1)$	$RE^2_0(2)$	.....	$RE^2_0(D)$
.	.....	.....	.....	.....
.	.....	.....	.....	.....
A	$RE^A_0(1)$	$RE^A_0(2)$	.....	$RE^A_0(D)$

FIGURE 1. Illustration of the replicas existence in the matrix of the datacenters and objects ($A \times D$).

Definition 4 (Exponential Decaying Function (EDF)): EDF means that the value shrinks by a constant factor for each time slot passed. It is defined over time slots with values in the $[0 - 1]$ range. EDF is used to calculate the exponential decay in the access and engagement rates of the object a in the current time slot t_c using the numbers of Gets, Shares, Comments, and Likes from the previous time slot t_{c-1} to the start time slot t_s.

$$EDF(t_c, t_s) = e^{-\sigma(t_c - t_s)} \tag{2}$$

where σ is used to decide how fast the decay happens with time slots. The bigger the σ, the faster the decay rate. If $(t_c - t_s) = 0$, then $EDF(t_c, t_s) = 1$. As the difference between the current time slot t_c and the first time slot t_s increases, this confirms that the EDF asymptotically approaches zero at infinity.

Definition 5 (Exponential Decaying Access Rate): Access rate represented as AR^a_g is one of the factors that decide the object's popularity in the OSN. It is calculated using the number of Get requests issued from all regions to the object a from the time slot t_{c-1} to the first time slot t_s.

$$AR^a_{t_c} = \sum_{g=1}^{G} \sum_{t_i=t_{c-1}}^{t_s} GR^a_g(t_i, t_{i-1}) \times EDF(t_{c-1}, t_i) \tag{3}$$

Definition 6 (Exponential Decaying Engagement Rate): Shares, Comments, and Likes are the main parts of the engagement rate in the OSN. The engagement rate for object a at the current time slot t_c is denoted by $ER^a_{t_c}$. Similar to $AR^a_{t_c}$, $ER^a_{t_c}$ is calculated independently using the number of Shares SR^g_a, Comment CR^g_a, and Likes LR^g_a issued from all regions to the object a from the time slot t_{c-1} to the first time slot t_s using the EDF. However, according to Kim and Yung [35], Share, Comment, and Like have different weights in the determination of the object's popularity. The weight of one Share request is equal to two Comment requests. At the same time, each Comment request equals seven Like requests. Similarly, we assume that each Like is equal to two access requests. These weight variations between Share, Comment, and Like can be expressed as weight variables for each type of request. Hence, we use β_s, β_c, and β_l as weight variables for the Share, Comment, and Like requests,

respectively.

$$ER^a_{t_c} = \sum_{g=1}^{G} \sum_{t_i=t_{c-1}}^{t_s} (SR^a_g(t_i, t_{i-1}) \times \beta_s + CR^a_g(t_i, t_{i-1}) \times \beta_c + LR^a_g(t_i, t_{i-1}) \times \beta_l) \times EDF(t_{c-1}, t_i) \quad (4)$$

Definition 7 (Object Popularity): The popularity of object a in the current time slot t_c in the OSN is denoted by $OP^a_{t_c}$. The object popularity in the OSN is measured by the amount of access and engagement the object received in each time slot. However, the Get request is served from one replica, while the Put request is written to all object replicas. Therefore, the object popularity is determined from the result of dividing the access rate $AR^a_{t_c}$ of the object a in the current time slot by the number of replicas $RN^a_{t_{c-1}}$ from the previous time slot. This result is then added to the engagement rate $ER^a_{t_c}$ of the object a in the current time slot.

$$OP^a_{t_c} = \frac{AR^a_{t_c}}{RN^a_{t_{c-1}}} + ER^a_{t_c} \quad (5)$$

Definition 8 (Replication Number): Replication number $RN^a_{t_c}$ is the value calculated from the object's popularity value to decide the number of replicas for object a in the current time slot t_c. The popularity of the hot object in the OSN is always a large number, which is proportionally reduced to one digit using logarithms. The base-10 logarithm of a number is approximately equal to the number of digits in that number. Therefore, it is calculated by taking the base-10 logarithm using equation (6).

$$RN^a_{t_c} = log(\frac{AR^a_{t_c}}{RN^a_{t_{c-1}}} + ER^a_{t_c}) \quad (6)$$

C. COST MODEL

The cost model is concerned with calculating the object hosting cost in the STaaS. Each datacentre has a tuple of three costs for hosting the replica of the object, as follows.

Definition 9 (Storage Cost): The storage cost of all objects in the time slot t_i is denoted by SC_{t_i} and calculated using equation (7), which uses each replica size $SZ^a_{t_i}$ of the object a and the price of storage for each size unit in datacentre d represented by $USP(d)$.

$$SC_{t_i} = \sum_{a=1}^{A} \sum_{d=1|RE^a_{t_i}(d)=1}^{D} (SZ^a_{t_i} \times USP(d)) \quad (7)$$

Definition 10 (Request Cost): It is the cost of the Get and Put requests to all objects in time slot t_i, and it is denoted by RC_{t_i}. It is defined by equation (8) using the number of Get requests $GR^a_{t_i}$ and the number of the Put requests $PR^a_{t_i}$, the object a received in the time slot t_i multiplied by their corresponding price of requests unit in datacentre d.

$$RC_{t_i} = \sum_{a=1}^{A} \sum_{d=1|RE^a_{t_i}(d)=1}^{D} (GR^a_{t_i} \times UGP(d)) + (PR^a_{t_i} \times UPP(d)) \quad (8)$$

where $UGP(d)$ is the price of the Get request in the datacentre d and $UPP(d)$ is the price of the Put request in the datacentre d.

Definition 11 (Network Cost): It is the cost of the network consumption by the Get and Put requests to all objects. This cost is denoted by NC_{t_i} and estimated by equation (9) using the total retrieved data size $DSC_{t_i}(d)$ of the Get requests issued to the datacenter d in the time slot t_i and the price of the size unit price $UNP(d)$ in datacentre d.

$$NC_{t_i} = \sum_{d=1}^{D} (DSC_{t_i}(d) \times UNP(d)) \quad (9)$$

D. LATENCY MODEL

The latency model focuses on the calculations of each request latency time, the region's average latency time, and the percentile latency time of all requests in the time slot.

Definition 12 (Latency): Each user is assumed to obtain the requested object from the closest datacenter. In the cloud, [34], [36] illustrated that the latency time L of small-size objects is always determined by the latency of the network. Since the network's distance significantly decides the network latency, we used equation (10), which uses the distance between the user region and datacenter to estimate the latency time L in milliseconds (ms).

$$L(ms) = \begin{cases} 50, & \text{user and datacenter are in the same region.} \\ Dist(KM) \times 0.02 + 5, & \text{otherwise.} \end{cases} \quad (10)$$

Definition 13 (Region Average Latency): The average latency for all regions G from all datacenters D is calculated for the purpose of replica placement based on the latency limits. The average latency of regions is represented as a matrix of $G \times D$, which is denoted by RL^g_d and it is calculated using equation (11), where GR(g) is the number of requests issued from region g.

$$RL^g_d = \frac{\sum_{g=1}^{G} \sum_{d=1}^{D} L}{\sum_{g=1}^{G} GR(g)} \quad (11)$$

Definition 14 (Time Slot Percentile Latency): The percentile latency of all requests to all objects A in all datacenters D for time slot t_i is denoted by TPL_{t_i} and it is calculated using equation (12).

$$TPL_{t_i} = \frac{P}{100} \times (LN_{t_i} + 1) \quad (12)$$

where p is the required percentile of the requests (e.g., 90% or 99%) and LN_{t_i} is the number of the measured latencies L for all Get requests in time slot t_i.

IV. STATIC REPLICATION AND PLACEMENT

Static replication and placement are based on a static number of replicas hosted on fixed datacenters. The number of replicas and their placement are not adapted based on the changes in the object popularity in the OSN. In the following section, we present two static replication and placement algorithms used as benchmark algorithms for the online algorithms.

A. LOCAL REPLICATION AND PLACEMENT (LRP) ALGORITHM

Considering that the majority of Get and Put requests come from the same region as the object owner's region, we propose an algorithm that creates two replicas of each object and hosts them in the same region where the owner of the object is located. This algorithm hosts the replicas in different datacenters that belong to two different CSPs. The reason behind using two replicas hosted in two different CSPs is to provide another replica when there is a heavy workload on the object and to prevent vendor lock-in.

Let A_{new} be the new objects that must be replicated and hosted. The region of the object's owner is denoted by UR_a, and the set of datacenters that are located in the same region of the object owner are denoted by DC_a. These datacenters are selected based on the lowest cost produced in the region.

Algorithm 1 . LRP Algorithm

Input: Datacenters' detail (locations, regions,. . ..), and objects with their workload.

Output: The latency percentile (TPL_{t_i}) of all requests in each time slot, and the total cost (TC_{t_i}) of all objects A in each time slot.

$RN^a_{t_0} \Leftarrow 2$;
for $a = 1$ to A_{new} **do**
  $UR_a \Leftarrow Find\_User\_Region(a)$;
  $DC_a \Leftarrow Find\_Region\_Datacenters(UR_a)$;
  $Create\_Assign\_Replicas(DC_a)$;
end for
for $i = 0$ to T **do**
  $L_{t_i} \Leftarrow Calculate\_Each\_GET\_Latency(t_i)$;
  $TPL_{t_i} \Leftarrow Calculate\_Latency\_Percentile(L_{t_i})$;
  $TC_{t_i} \Leftarrow Calculate\_Total\_Cost(t_i)$;
  Return (TPL_{t_i}, TC_{t_i});
end for

The pseudocode of the LRP algorithm is presented in Algorithm 1. LRP sets the number of replicas for each object in the OSN to two replicas (line 1). In the first loop, it iterates through all the newly created objects A_{new} in the OSN (lines 2-6). Firstly, Using the function $Find\_User\_Region(a)$, it finds the owner region of the object a (line 3). Then, LRP finds the cheapest datacenters that should belong to different CSPs in UR_a using the function $Find\_Region\_Datacenters(UR_a)$(line 4). Next, it creates and assigns the replicas to the selected datacenter DC_a using the function $Create\_Assign\_Replicas(DC_a)$ (line 5).

TABLE 2. The details of datacenters selected for FDCL.

No.	Continent	CSP	Datacenter
1	South America	Azure10	US west Washington
2	South America	Amazon39	US East North Virginia
3	Europe	Google68	Germany
4	Europe	Amazon55	Italy
5	Asia	Azure36	Singapore
6	Asia	Google70	India
7	Asia	Amazon43	Hong Kong
8	Asia	Azure13	UAE
9	Africa	Amazon60	South Africa
10	Australia	Google73	Sydney
11	North America	Azure30	Brazil

In the second loop, LRP calculates the latency time of each Get request in the time slot t_i using the function $Calculate\_Each\_GET\_Latency(t_i)$ (line 8). Then, it calculates the required percentile for the latency times of all Get requests issued in the time slot t_i using the function $Calculate\_Latency\_percentile(L_{t_i})$ (line 9). Next, the total cost of hosting all objects in time slot t_i is calculated using the function $Calculate\_Total\_Cost(t_i)$ (line 10). Finally, LRP returns the latency time of the required percentile and total hosting cost in the time slot t_i (line 11).

B. SELECTIVE DISTRIBUTED REPLICATION (SDR) ALGORITHM

SDR is an algorithm that is also based on static replication and placement mechanisms. SDR is proposed in [12] and [17] for the optimal placement of objects in Facebook. It is designed to use several datacenters selected carefully to be scattered around the world so that any user in the world receives the requested object from the nearest datacenter. Therefore, it produces the lowest latency time for all users around the world.

We use the same design of this replication with a slight adaptation of the datacenters to cover most of the regions. Thus, the datacenters used should belong to all regions in Asia, north America, south America, Europe, Africa, and Australia. Therefore, eleven datacenters from the three giant CSPs are used for the SDR algorithm. The datacenters used by SDR are predefined in a list denoted by FDCL. Table 2 presents these datacenters and their details.

Let A_{new} be the set of the new objects in the OSN that need to be replicated and hosted in the FDCL datacenters. The pseudocode of this algorithm is shown in Algorithm 2. In the first loop, the SDR algorithm creates the replicas of each newly created object in the OSN using the function $Create\_Replicas(a)$ (line 2). Then, using the function $Assign\_Replicas(FDCL)$, it hosts each replica in one datacenter from the list FDCL (line 3). The second loop performs the same functions as the LRP algorithm.

V. DYNAMIC ONLINE REPLICATION AND PLACEMENT

The optimization of this research is concerned with determining the appropriate number of replicas for an object at each time slot so that the availability is adequate to fulfil

Algorithm 2 SDR Algorithm

Input: Datacenters' detail (locations, regions,....), full datacenters list (FDCL), and objects with their workload.

Output: The latency percentile (TPL_{t_i}) of all requests in each time slot, and the total cost (TC_{t_i}) of all objects A in each time slot.

for $a = 1$ to A_{new} **do**
  Create_Replicas(a);
  Assign_Replicas($FDCL$);
end for
for $i = 0$ to T **do**
  $L_{t_i} \Leftarrow Calculate\_Each\_GET\_Latency(t_i)$;
  $TPL_{t_i} \Leftarrow Calculate\_Latency\_Percentile(L_{t_i})$;
  $TC_{t_i} \Leftarrow Calculate\_Total\_Cost(t_i)$;
  Return (TPL_{t_i}, TC_{t_i});
end for

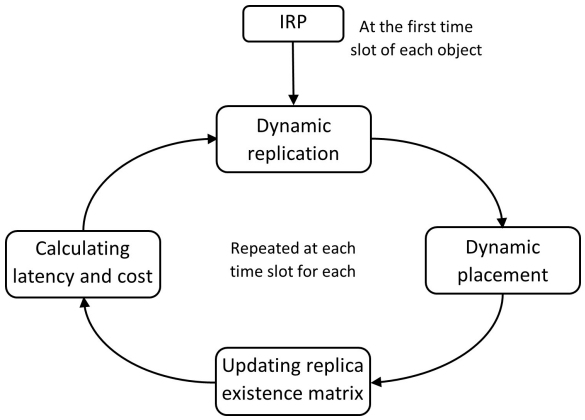


FIGURE 2. Dynamic object replication and placement procedure.

the expected workload and the optimal placement of these replicas to produce the lowest latency time for the user of the OSN and the minimum monetary cost for the OSN service providers. This optimization should be conducted online without prior knowledge of future workload. In this section, we present the optimization stages that trade off the latency time and monetary cost. These stages are then utilized by two online algorithms that use different sizes of time slots.

A. OPTIMIZATION STAGES

The optimizations of object availability, latency time of user requests, and monetary cost of OSN service providers are conducted through five stages, which include: 1) Initial replication and placement (IRP), 2) Dynamic replication, 3) Dynamic placement, 4) Updating object existence matrix, and 5) Calculating latency and cost. The first stage is conducted once at the beginning of the object's lifetime in the OSN, whereas the following four stages are repeated every time slot, as shown in Fig. 2.

1) INITIAL REPLICATION AND PLACEMENT (IRP)

When an object is created in the OSN, and as there is no known popularity of the object because there are no previous access and engagement rates in the first time slot, we generate two replicas of the object and host them in the most cost-effective datacenters of the same region where the object is created.

2) DYNAMIC REPLICATION

As the first time slot is passed and there is a previous workload of the object, the dynamic replication strategy is used to decide the popularity of the object in the OSN. The current object popularity is calculated using the access and engagement rates from the previous time slot t_{c-1} to the first time slot t_s using equations (2)-(5). Then, the number of replicas for the objects in the current time slot t_c is calculated using its current popularity in the OSN by equation (6).

3) DYNAMIC PLACEMENT

The placement stage of the object replicas is conducted based on each region rank for the object. The placement strategy first identifies the regions that issued requests to the object. The rank of each region of the object a in the current time slot t_c is denoted by $Rnk^a_{t_c}(g)$, and it is calculated using the number of Get, Share, Comment, and Like requests issued from the region g from the time slot t_{c-1} to the first time slot t_s as in equation (13)

$$\begin{aligned} &Rnk^a_{t_c}(g) \\ &= \sum_{t_i=t_{c-1}}^{t_s} (GR^a_g(t_i, t_{i-1}) + SR^a_g(t_i, t_{i-1}) \\ &\quad \times \beta_s + CR^a_g(t_i, t_{i-1}) \times \beta_c + LR^a_g(t_i, t_{i-1}) \times \beta_l) \\ &\quad \times EDF(t_{c-1}, t_i) \end{aligned} \tag{13}$$

The ranks of the regions are then sorted in descending order to get the regions with the highest ranks for the object a. A list $Gl^a_{t_c}$ of length $RN^a_{t_c}$ is then generated for the regions with the highest ranks, where $Gl^a_{t_c} \in G$. The placement strategy generates the required replicas according to the $RN^a_{t_c}$ and assigns them to the selected subset of the datacenters, which are denoted by $DC^a_{t_c}$, where $DC^a_{t_c} \in D$. These datacenters are selected in the regions $Gl^a_{t_c}$ based on the satisfaction of the latency limits and the lowest cost produced.

4) UPDATING REPLICA EXISTENCE MATRIX

After the number of replicas is decided and the object's replicas are hosted in the selected datacenters, the next stage is to update the replicas' existence matrix of the current time slot $(A \times D)_{t_c}$. This matrix is used as a reference for checking the object replica's availability and redirecting the Get request to obtain better latency performance.

5) CALCULATING LATENCY AND COST

In this stage, the latency of each request is calculated based on the distance between the region of the request and the datacenter hosting the replica using equation (10). Then, the required percentile latency of each time slot is calculated using equation (12). Next, the cost of hosting all the objects in

the selected datacenters is produced using equations (7)-(9). Finally, the average latency time and total cost of all time slots are produced to calculate the latency difference between all algorithms.

B. PROPOSED ONLINE ALGORITHMS

We propose two online algorithms that are based on dynamic replication and placement. They use the dynamic replication model to decide the appropriate number of replicas for an object at each time slot. The algorithms also optimize the latency time by conducting region ranking-based placement of the replicas at each time slot and the monetary cost by selecting the datacenters with the lowest costs.

1) DETERMINISTIC TIME SLOT (DTS) ALGORITHM

DTS Algorithm is based on time slots of fixed size. This algorithm focuses on determining the popularity $OP^a_{t_c}$ for object a in the current time slot t_c based on the decaying access and engagement rates from the previous time slot t_{c-1} to the first time slot t_s using the EDF according to equations (2)-(5). Then, $RN^a_{t_c}$ of the object a in the current time slot t_c is calculated using the popularity according to the equation (6). The placement of these replicas is then conducted using each region rank by equation (13).

Algorithm 3 DTS Algorithm

Input: Datacenters' detail (locations, regions,. . ..), regions-datacenters latency matrix, and objects and their workload.

Output: Number of replicas $RN^a_{t_i}$ for each object a in time slot t_i, replica existence matrix $(A \times D)_{t_i}$, the latency percentile (TPL_{t_i}) in each time slot, and the total cost (TC_{t_i}) of all objects A in each time slot.

1: $RN^a_{t_0} \Leftarrow 2$;
2: **for** $a = 1$ to A_{new} **do**
3: $IRP(a, RN^a_{t_0})$;
4: **end for**
5: **for** $i = 1$ to T **do**
6: **for** $a = 1$ to A **do**
7: $RN^a_{t_i} \Leftarrow Calculate\_Replicas\_Number(a, t_i)$;
8: $Gl^a_{t_i} \Leftarrow Find\_Max\_Region\_Rank(a, t_i, RN^a_{t_i})$;
9: $DC^a_{t_i} \Leftarrow Find\_Datacenter(Gl^a_{t_i})$;
10: $Create\_Assign\_Replicas(DC^a_{t_i})$;
11: $Update\_Replica\_Existence\_Matrix(a, DC^a_{t_i})$;
12: **end for**
13: $L_{t_i} \Leftarrow Calculate\_Each\_GET\_Latency(t_i)$;
14: $TPL_{t_i} \Leftarrow Calculate\_Latency\_Percentile(L_{t_i})$;
15: $TC_{t_i} \Leftarrow Calculate\_Total\_Cost(t_i)$;
16: **Return** (TPL_{t_i}, TC_{t_i});
17: **end for**

DTS algorithm, as shown in the pseudocode Algorithm 3, firstly loops through all the newly created objects in the OSN to conduct the IRP using the function $IRP(a, RN^a_{t_0})$ (Lines 2-4). In the nested loops, by using the function $Calculate\_Replicas\_Number(a, t_i)$, it calculates the number of replicas $RN^a_{t_i}$ for each object using its popularity in the OSN based on the equations (2)-(6) (line 7). Then, it calculates the ranks of the regions for the object a based on equation (13) and selects the regions with the highest ranks to $GL^a_{t_i}$ based on the number of replicas $RN^a_{t_i}$ using the function $Find\_Max\_Region\_Rank(a, t_i, RN^a_{t_i})$ (line 8). Based on the set of regions in $GL^a_{t_i}$, DTS determines the set of datacenters $DC^a_{t_i}$ with the lowest costs to host the replicas of the object a in time slot t_i using the function $Find\_Datacenter(Gl^a_{t_i})$ (line 9). Next, the function $Assign\_Replicas(DC^a_{t_i})$ assigns the replicas of the object to the set of datacenters in the $DC^a_{t_i}$ (line 10). After that, it updates the replica existence matrix $(A \times D)_{t_i}$ with the new placement of the replicas of the object a in the time slot t_i using the function $Update\_Replica\_Existence\_Matrix(a, DC^a_{t_i})$ (line 11). Since the placement is conducted, the DTS algorithm starts to measure the latency time of each Get request using the function $Calculate\_Each\_GET\_Latency(t_i)$ based on equation (10) (line 13). Using the function $Calculate\_Latency\_percentile(L_{t_i})$, it calculates the required percentile of latency time for all Get requests issued from all regions in the time slot t_i (line 14). Then, it calculates the total cost of all objects A in the time slot t_i using the function $Calculate\_Total\_Cost(t_i)$ (line 15). Finally, the DTS algorithm returns the latency time percentile and the total cost in the time slot t_i (line 16).

2) RANDOMIZED TIME SLOT (RTS) ALGORITHM

There is a strong relationship between the object's age and its popularity in the OSN [37]. The workload of the object is often heavy at the beginning of its lifetime. However, this workload is reduced as time passes until the object is cold. Therefore, we propose this algorithm that adapts the availability of the object and the placement of its replicas at a very early time of its lifetime in OSN. The RTS algorithm is based on the Receding Horizon Control [38] with the adaptation of random time units and exponential time slots. The timeline of the object is divided into an increasing time slot. The size of the time slot increases as the object becomes colder.

According to EdgeRank Checker, a Facebook page analytic company, the average for receiving the majority of object's access and engagement happens within the first three hours [39] in the OSN. Hence, we can assume that the time unit can be in the range of [1-6] hours. In this algorithm, the time unit defines the number of hours h_a given for each object. h_a is determined by a random integer in the range of $1 \leqslant h_a \leqslant 6$. The window is represented by $w^a_{t_i}$, which contains an increasing number of time units. $w^a_{t_i}$ determines the time slot size $TPS^a_{t_i}$ in hours for each object a. However, to prevent the continuous increase of the time slot size, we define a time slot size limit TL.

$$h_a = Rand(1, 6) \tag{14}$$

$$wt^a_{t_i} = \begin{cases} 2^i, & \text{if } wt^a_{i-1} < TL \\ TL, & wt^a_{i-1} \geq TL \end{cases} \quad (15)$$

$$TPS^a_{t_i} = wt^a_{t_i} \times h_a \quad (16)$$

Algorithm 4 RTS Algorithm

Input: Datacenters' detail (locations, regions,. . ..), regions-datacenters latency matrix, and objects and their workload.

Output: Number of replicas $RN^a_{t_i}$ for each object a in time slot t_i, replica existence matrix $(A \times D)_{t_i}$, the latency percentile (TPL_{t_i}) in each time slot, and the total cost (TC_{t_i}) of all objects A in each time slot.

1: $RN^a_{t_0} \Leftarrow 2$;
2: **for** $a = 1$ to A_{new} **do**
3: $h_a \leftarrow Rand(1, 6)$;
4: $IRP(a, RN^a_{t_0})$;
5: **end for**
6: **for** $i = 1$ to T **do**
7: **for** $a = 1$ to A **do**
8: $w^a_{t_i} \Leftarrow Calculate\_Window(a, t_i)$;
9: $TPS^a_{t_i} \Leftarrow Calculate\_Time\_Period\_Size(a, w^a_{t_i}, t_i)$;
10: $RN^a_{t_i} \Leftarrow Calculate\_Replicas\_Number(a, t_i)$;
11: $Gl^a_{t_i} \Leftarrow Find\_Max\_Region\_Rank(a, t_i, RN^a_{t_i})$;
12: $DC^a_{t_i} \Leftarrow Find\_Datacenter(Gl^a_{t_i})$;
13: $Create\_Assign\_Replicas(DC^a_{t_i})$;
14: $Update\_Replica\_Existence\_Matrix(a, DC^a_{t_i})$;
15: **end for**
16: $L_{t_i} \Leftarrow Calculate\_Each\_GET\_Latency(t_i)$;
17: $TPL_{t_i} \Leftarrow Calculate\_Latency\_Percentile(L_{t_i})$;
18: $TC_{t_i} \Leftarrow Calculate\_Total\_Cost(t_i)$;
19: **Return** (TPL_{t_i}, TC_{t_i});
20: **end for**

The RTS algorithm is anticipated to outperform the DTS algorithm in terms of latency time and monetary cost optimizations. This is because the RTS algorithm starts the optimization of object availability and replica placement based on the first time slot whose size is in the range of [1-6]. This small time slot allows the RTS algorithm to adapt the number of replicas and the hosting datacenters based on the regions' ranks for the object so that the OSN can serve the expected workload of the object within the acceptable latency time.

The RTS algorithm's pseudocode is shown in Algorithm 4, where all the functions perform the same as in the DTS algorithm except the function of the new objects for calculating the random time units and exponential time slots. The function Rand(1,6) finds the random value of the time unit h_a for each new object in the OSN (Line 3). Then, by using the function $calculate\_window(a, t_i)$, it calculates the window of the time slot t_i for object a (line 8). Since the window of each object a is calculated, the RTS algorithm then decides the time slot size $TPS^a_{t_i}$ using the function $Calculate\_Time\_Period\_Size(a, w^a_{t_i}, t_i)$ (Line 9). The rest of the functions perform similar tasks to the functions in the DTS algorithm.

VI. EXPERIMENT AND PERFORMANCE EVALUATION

To evaluate the effectiveness of the proposed online algorithm, we conduct extensive experiments using the discrete event simulator CloudSim proposed in [40] and [41] with a synthesized workload generated based on real Facebook statistics. We first discuss the time complexity of all the algorithms. Then, the settings used in the implementation of the experiments are presented. Next, we introduce the evaluation methods. Finally, we compare and analyze all algorithms' latency and cost performance.

A. TIME COMPLEXITY

Algorithms 1 and 2 have the same structure and number of loops. Hence, they have the same time complexity. To calculate the time complexity, there are two loops in each algorithm. The first loop takes the time complexity of $O(A_{new})$. The time complexity of the second loop is $O(T)$. Some of the functions inside the second loop have also loops and produce considerable time complexity. These function include $Calculate\_Each\_GET\_Latency(t_i)$, which takes time complexity of $O(GR)$ and $Calculate\_Latency\_Percentile(L_{t_i})$ that runs for L^2 times. Therefore, each algorithm's total time complexity is $O(A_{new} + T(GR + L^2))$.

Similarly, the structure and number of loops of Algorithms 3 and 4 are the same. Consequently, they have the same time complexity. To determine the time complexity of each algorithm, we first need to calculate the time complexity of the first loop, which runs in $O(A_{new})$. The last two nested loops repeat for $T \times A$ times, which takes a time complexity of $O(TA)$. It is worth mentioning that the second loop in the nested loops iterates only through the active objects. Active objects are the objects that still receive Get and Put requests in the current time slot. Same as Algorithms 1 and 2, the functions $Calculate\_Each\_GET\_Latency(t_i)$ and $Calculate\_Latency\_Percentile(L_{t_i})$ inside the last two nested loops take $GR + L^2$. Hence, the total time complexity of each algorithm is $O(A_{new}) + O(TA) \times O(GR + L^2) = O(A_{new} + TA(GR + L^2))$.

B. EXPERIMENTS SETTINGS

We use the following setup for the specifications of the dataset, datacenters, the objects' workload, and the experiment's parameters.

1) DATASET

objects' long-term workload in OSN is classified as confidential data. So far, there is no OSN service provider has released such information. As a result, to evaluate our proposed algorithms, we use an experimental dataset based on real-world statistics of Facebook generated in [42].

The dataset was extracted from a real statistic of Facebook pages [43]. Similar to the model proposed in [44], the exponential distribution is used to construct the Get and Put

rates synthetically. As revealed in the study, it is always assumed that the number of GET requests in OSNs is significantly more than that of PUT requests. Therefore, the ratio of the Gets and Puts rates in the dataset is approximately 30:1. The dataset is a trace-driven of 1500 objects for the period of 120 days with a total size of 5GB.

2) DATACENTERS

we utilize all the available public datacenters provided by the three giant CSPs (Google, Amazon, and Microsoft Azure). In order to provide more alternatives for the placement strategy, we use 86 datacenters scattered in 20 regions around the world. We use the prices of each CSPs' datacenter as specified in December 2022.

3) PARAMETERS

During the experiments, we calculate the latency of each region for the duration of 120 days. The range of random time units used in the RTS algorithm is $[1-6]$, and one day for the DTS algorithm. In the implementation of the RTS algorithm, we set the $TL = 16$, which equals 48 hours. In the online algorithms, any object's minimum number of replicas is set to $RN^a_{t_c} = 2$ at any time slot.

C. PERFORMANCE EVALUATION

The main goal of the proposed online algorithms is to minimize the difference in the latency time produced between the expensive SDR algorithm and the traditional static LRP algorithm while producing possible cost savings. To examine the effectiveness of the online algorithms, we compare their performance to the performance of the SDR and LRP algorithms in terms of latency time and cost savings. The performance of the algorithms is evaluated in terms of three-fold, which include: 1) the latency time produced in each time slot by each algorithm using 90%ile, 95%ile, and 99%ile, which correspond respectively to 90%, 95%, and 99% of the requests, 2) the effect of the predefined latency limits on the percentile latency time produced by each algorithm in each time slot, and 3) the average percentage of cost savings using the various latency limits achieved by the online and LRP algorithms compared to the SDR.

For simplicity of the latency time results, we present the results of the simulation experiments using two different methods: 1) the latency time produced in each time slot by each algorithm using the various latency limits, and 2) the average percentage of latency time improvement of all time slots achieved by each algorithm compared to the traditional LRP algorithm. As there is a significant fluctuation in the latency time produced by the time slots, we produce the dotted line representing the linear trendline of the latency times delivered in time slots for each algorithm.

1) LATENCY PERFORMANCE

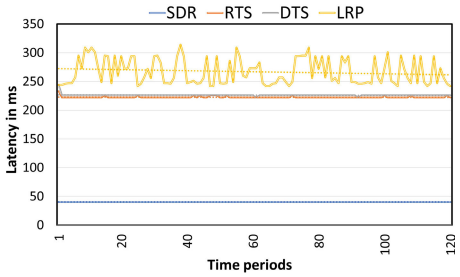
We first present the latency performance of the DTS, RTS, SDR, and LRP algorithms using the dataset without forcing any latency limits. Fig. 3. illustrates the percentile latency time produced by each algorithm in the 120 time slots. Figs. 3a and 3d depict the latency times produced by all algorithms using the 90%ile of all requests in each time slot. It can be seen that DTS and RTS produce latency times below the threshold of 250ms in all time slots. On average, the latency time improvement for all requests by SDR, RTS, and DTS are 85%, 17%, and 16%, respectively. Using the 95%ile as shown in Figs. 3b and 3d, the latency times produced by RTS reach the threshold, while the latency times of DTS exceed the threshold slightly. Because of this rise in the latency times produced, the average improvement is reduced to 84%, 16%, and 15% for the SDR, RTS, and DTS algorithms, respectively. At the same time, Figs. 3c and 3d present the 99%ile latency time produced by all algorithms. For the online algorithms, it is evident that the produced latency times in all time slots are higher than the 250ms threshold. This exceeding of the threshold is due to the fact that no latency limits are forced in the placement strategy. However, there are increases in the average improvements of SDR, RTS, and DTS to 87%, 26%, and 22%, respectively. This is because of the significant increase in the latency times produced by LRP.

For the online algorithms, we can see that the RTS algorithm slightly overcomes the DTS algorithms in almost all time slots using the various percentiles. This superiority of the RTS algorithm increases slightly as the latency limit increases. It is due to its early optimization of the placement for the replicas. It is also apparent that as the percentile increases, the produced latency times of the RTS, DTS, and LRP algorithms increase. Due to its static placement and the high number of replicas scattered around the world, the SDR algorithm delivers the lowest latency time using all the percentiles. In contrast, using the various percentiles, the LRP algorithm produces the worst latency times in all time slots. This happens because of the low number of replicas and the static placement strategy that hosts the replicas locally to the object's owner.

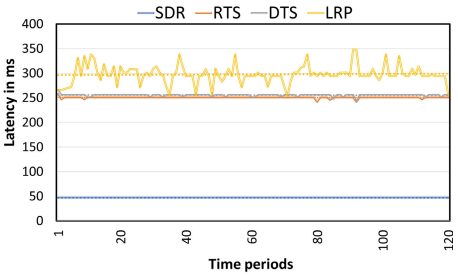
2) LATENCY LIMITS EFFECTS

We extend our experiments to investigate the effect of latency limits for the replica placement on the latency times produced by each algorithm. Therefore, we run each algorithm on the dataset using various latency limits, which include 50ms, 150ms, and 250ms.

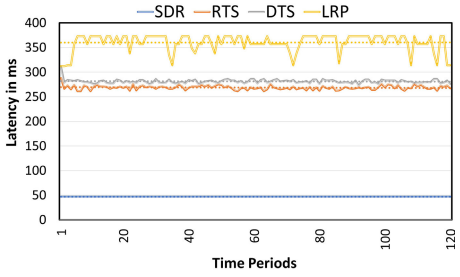
Fig. 4 shows the latency time performance of all algorithms using the latency limit of 50ms. The percentile latency times of all algorithms in each time slot are shown in Figs. 4a, 4b, and 4c. While Fig. 6d presents the average latency time improvement of the SDR, RTS, and DTS algorithms compared to the traditional LRP algorithm. The 90%ile latency times produced by each algorithm of all requests in each time slot are illustrated in Figs. 4a and 4d. It can be seen that our online algorithms delivered latency times below the 250ms threshold. The average improvement of latency time for the SDR, RTS and DTS algorithms compared to the LRP algorithm are 85%, 56%, and 35%, respectively. Figs. 4b



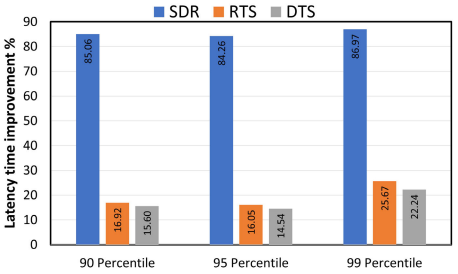
(a) The 90%ile latency time in each time slot of all algorithms.



(b) The 95%ile latency time in each time slot of all algorithms.

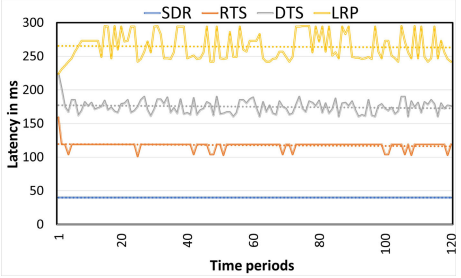


(c) The 99%ile latency time in each time slot of all algorithms.

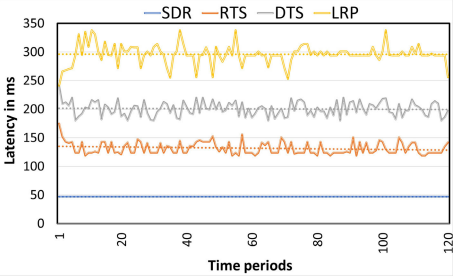


(d) The average percentage of latency time improvement by each algorithm compared to the LRP.

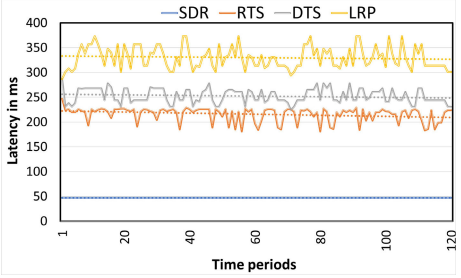
FIGURE 3. Latency time produced by all algorithms without latency limits for the 120 time slots.



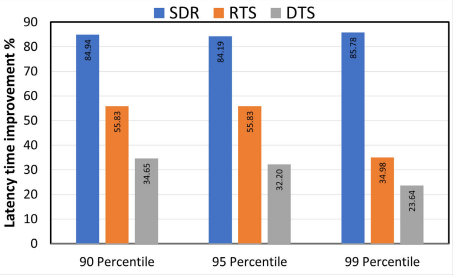
(a) The 90%ile latency time in each time slot of all algorithms.



(b) The 95%ile latency time in each time slot of all algorithms.



(c) The 99%ile latency time in each time slot of all algorithms.



(d) The average percentage of latency time improvement by each algorithm compared to the LRP.

FIGURE 4. Latency time produced by all algorithms for the 120 time slots with latency limits of 50ms.

and 4d present the latency times using the 95%ile. Although there is an increase in the latency times produced by all algorithms, the latency times of the RTS and DTS algorithms are still below the 250ms threshold. The average latency time improvement achieved by the SDR, RTS, and DTS algorithms are 84%, 56%, and 32%, respectively. When the percentile of the requests is increased to 99% as shown in Figs. 4c and 4d, there is a significant increase in the latency times produced by the LRP and online algorithms, leading the DTS algorithm to reach the threshold. In contrast, the RTS algorithm preserves

the produced latency times in all time slots below the 250ms threshold. Moreover, this rise in the produced latency times increases the average latency time improvement of the SDR algorithm to 86%, while reducing the improvement to 35% and 24% for the RTS and DTS algorithms, respectively.

Except for the SDR algorithm, the RTS algorithm significantly delivers the best latency times in all time slots compared to the DTS and LRP algorithms using the various percentiles. This is a result of the exponential time slots. In other words, time slots become more extensive as time passes, so the RTS algorithm can collect more information about the workload in the future, leading to further optimization. Due to the latency limits applied to the placement strategy, online algorithms, especially the RTS algorithm, become more competitive to the SDR algorithm which produces the best latency times and the most expensive replication.

Fig. 5 illustrates the results obtained by all algorithms under the latency limit of 150ms in each time slot. Figs. 5a, 5b, and 5c show each time slot latency times obtained by each algorithm using the various percentiles. The average improvement of the SDR, RTS, and DTS algorithms compared to the LRP algorithm is presented in Fig. 5d. Using 90%ile, as shown in Figs. 5a and 5d, it is apparent that the SDR, RTS, and DTS algorithms produce latency times below the 250ms threshold with an average improvement of 85%, 51%, and 20%, respectively. Figs. 5b and 5d depict the results delivered using 95%ile. As the request percentile increases, it can be seen that all algorithms produce an increase in the latency times. However, the latency times produced by the SDR, RTS, and DTS algorithms are still under the 250ms threshold with a respective average improvement of 84%, 51%, and 19% compared to the latency time produced by the LRP algorithm. When the percentile of the requests increases to 99, as illustrated in Figs. 5c and 5d, the RTS algorithm preserves the produced latency times below the 250ms threshold. In contrast, the DTS algorithm slightly exceeds this threshold. The average improvement of time latency obtained by the SDR, RTS, and DTS algorithms is 86%, 24%, and 21%, respectively.

As the latency limit is increased to 150ms, the online algorithms become less competitive with the optimal SDR algorithm. We can also observe that the RTS algorithm produces the second-best latency time using various percentiles. This is because of its premature optimization for the placement of the replicas according to the region ranks.

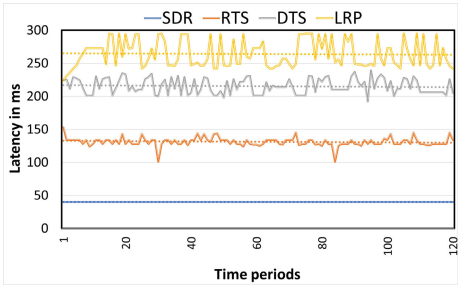
Fig. 6 depicts the performance of all algorithms using the latency limit of 250ms in the 120 time slots. The latency times of each percentile produced by all algorithms are demonstrated in Figs. 6a, 6b, and 6c. The latency time and the average improvement using the 90%ile and 95%ile are shown in Figs. 6a, 6b and 6d. As the trendlines demonstrate, the RTS and DTS algorithms produce nearly the same latency time with a slight superiority of the RTS algorithm at the beginning of the time slots. However, they produce almost the same access latency at the end of the time slots. The average improvement of the 90%ile achieved by the SDR, RTS, and DTS algorithms are 85%, 21%, and 18%, respectively. These values become 84%, 22%, and 18% with the 95%ile. The 99%ile of the produced latency times and the average improvement are presented in Figs. 6c and 6d. As illustrated by the trendlines, the latency times delivered by the DTS algorithm slightly exceed the threshold, while the produced latency times of the RTS algorithm reach the threshold only. The average improvements are increased to 86%, 25%, and 19% by the SDR, RTS, and DTS algorithms, respectively.

As the latency limit for the placement strategy is increased to 250ms, there is a significant rise in the produced latency times of the online algorithms. It is apparent that the latency times produced by online algorithms overlap in many time slots. This demonstrates that in some time slots, the RTS algorithm overcomes the DTS algorithm and vice versa in other time slots. The increase of the average improvement as the percentile increases is because of the increased latency times produced by the LRP algorithm. On the contrary, the RTS and DTS algorithms almost achieve steady latency times with 95%ile and 99%ile.

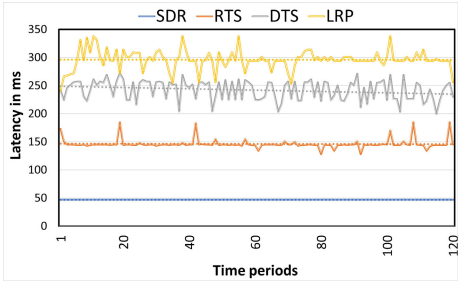
It is clear that the SDR and LRP algorithms produce the same latency time using various latency limits. This is because they are based on static replication and placement strategies. Moreover, the SDR algorithm is not influenced by the latency limits or percentiles because it has many replicas placed almost locally to each user worldwide. The results also reveal that the performance of the RTS and DTS algorithms is more effective with low latency limits when compared to their performance with high latency limits. Regardless of the superiority of latency times achieved by the SDR algorithm, which has expensive replication, it is also apparent that the RTS algorithm produces the most effective latency time compared to the DTS and LRP. Furthermore, it can be observed that the RTS algorithm never exceeds the threshold of 250ms, which is the maximum unnoticeable latency of OSN users. Whereas the produced latency times of the DTS algorithm reach or exceed this threshold using the 99%ile through all various latency limits. For online algorithms, As the latency limits increase, the fluctuation of the produced latency times increases between time slots within the same algorithm. The rationale is that more datacenters become qualified to host the object replicas as the latency limit rises.

3) COST PERFORMANCE

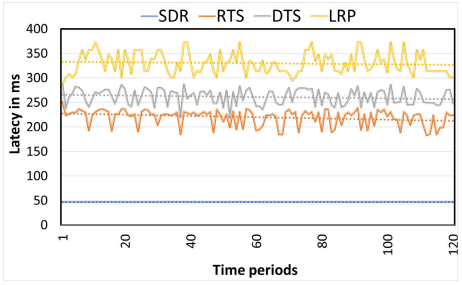
In order to investigate the cost performance of the online and LRP algorithms, we calculate the average percentage of cost savings achieved by each algorithm compared to the benchmark algorithm SDR. Table 3 shows the cost savings percentage of each algorithm compared to the SDR algorithm. The LRP algorithm produces steady cost savings of 10% using the various latency limits. This is because of the static replication and placement of the algorithm. For the online algorithms, we can see that using the latency limit of 50ms, the DTS and RTS algorithms cut the cost by almost 20% and 19%, respectively. When the latency limit



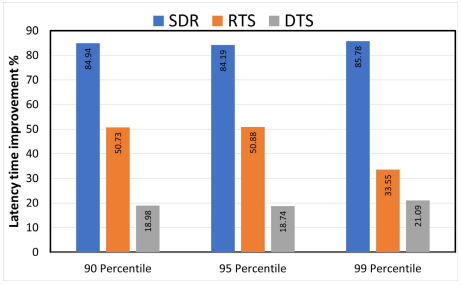
(a) The 90%ile latency time in each time slot of all algorithms.



(b) The 95%ile latency time in each time slot of all algorithms.

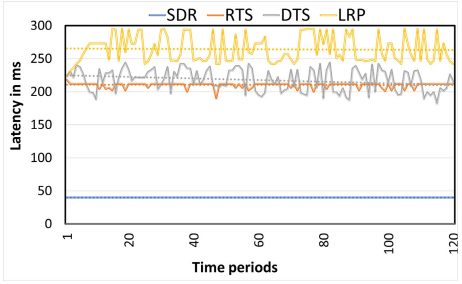


(c) The 99%ile latency time in each time slot of all algorithms.

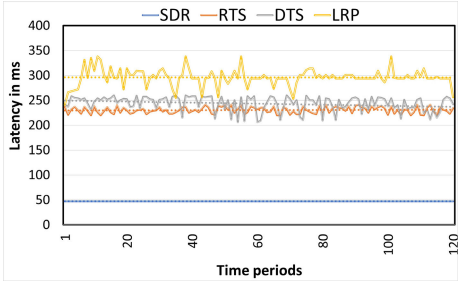


(d) The average cost savings percentage using various latency constraints of each algorithm compared to the SRLP.

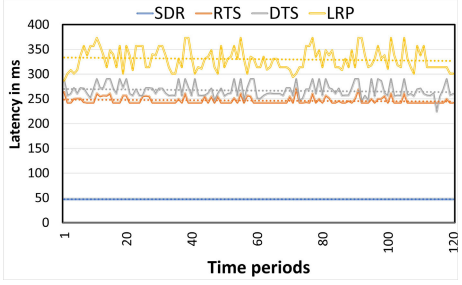
FIGURE 5. Latency time produced by all algorithms for the 120 time slots with latency limits of 150ms.



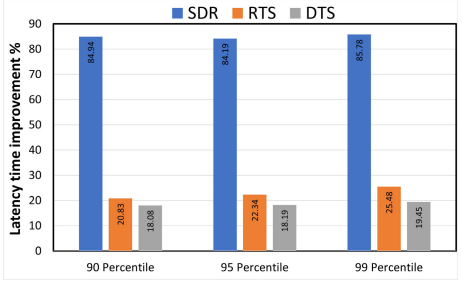
(a) The 90%ile latency time in each time slot of all algorithms.



(b) The 95%ile latency time in each time slot of all algorithms.



(c) The 99%ile latency time in each time slot of all algorithms.



(d) The average percentage of latency time improvement by each algorithm compared to the LRP.

FIGURE 6. Latency time produced by all algorithms for the 120 time slots with latency limits of 250ms.

is increased to 150ms, the DTS and RTS algorithms slightly increase the cost savings. Both reduce the cost by more than 20%. It is obvious that the difference in cost savings between the DTS and RTS algorithms is not that significant. However, when the latency limit is increased to 250ms, the RTS algorithm significantly overcomes the DTS algorithm. The DTS algorithm obtains a cost saving of almost 23%, while the RTS algorithm increases the cost saving to more than 28%. This is because the latency limit of 250ms offers more datacenters that can host the object replicas, allowing

TABLE 3. The average cost savings percentage achieved by each algorithm using the various latency limits compared to the SDR.

Latency limit	DTS	RTS	LRP
50ms	19.79%	18.66%	9.89%
150ms	20.22%	20.41%	9.89%
250ms	22.82%	28.33%	9.89%

the RTS algorithm to choose the datacenters with the lowest costs.

In general, from the latency time and cost savings results using various latency limits, it can be illustrated that the RTS algorithm produces better latency time using all required percentiles. Therefore, the latency results of the RTS algorithm are the most competitive to that of the SDR algorithm. Furthermore, latency times produced by the RTS algorithm never exceed the 250ms threshold. For the cost savings, it is obvious that the DTS algorithm slightly outperforms the RTS algorithm by 1% under the latency limit of 50ms. However, when the latency limit increases, the RTS algorithm's cost savings start to defeat that of the DTS algorithm. Therefore, we can conclude that the RTS algorithm produces better performance except for the cost savings under the latency limit of 50ms. However, this slight superiority of the DTS algorithm in cost savings can be sacrificed for the sake of the better latency time performance of the RTS algorithm in the latency limit of 50ms.

VII. CONCLUSION

Replicating the objects according to their popularity in the OSN and migrating their replicas based on the ranks of regions while utilizing the datacenters that produce the lowest possible costs in these regions are addressed in this paper. We use a dynamic popularity-based replication that determines the suitable number of replicas for object and region ranking-based placement that selects the optimal datacenters to host these replicas to trade off the latency time and monetary cost. The trade-off focuses on producing an acceptable latency time for the users of the OSN and the lowest monetary cost for the OSN service providers. Two online algorithms that use this dynamic replication and placement with deterministic and randomized time slots without previous knowledge of the future workload are proposed. The effectiveness of the proposed online algorithms is investigated using a Facebook-based synthetical workload. The results of the experiments show that RTS and DTS algorithms can satisfy the latency time up to 99%ile and 95% of the requests, respectively. For cost savings, RTS and DTS algorithms can deliver cost savings up to 28% and 23%, respectively, when compared to the SDR algorithm.

Two advantages of the proposed algorithms include the trade-off between the latency time of users and the hosting cost of the OSN service providers, and the proposed algorithms are online, which do not need prior knowledge of the future access and engagement rates. On the other hand, there are two disadvantages of the proposed algorithms. Firstly, the effectiveness of the proposed algorithms is not measured and examined using real CSPs. Secondly, the proposed algorithms do not consider the object replicas' consistency.

Some research tracks are possible to continue this work. 1) exploring different cost models, such as various storage classes and policies provided by different CSPs. 2) studying more factors that affect object popularity in OSNs. 3) measuring and examining the effectiveness of the proposed algorithms in real multiple CSPs.

ACKNOWLEDGMENT

The authors would like to acknowledge the facilities provided by Universiti Putra Malaysia for the execution, completion, and publication of this article.

ABBREVIATIONS

The following abbreviations are used in this manuscript:

STaaS	Storage as a service
CSP	Cloud storage service
OSN	Online social network
QoS	Quality of service
EDF	Exponential decaying function
SDR	Selective distributed replication
FDCL	Full datacenters list
SDR	Selective distributed replication
IRP	Initial replication and placement
DTS	Deterministic time slot
RTS	Randomized time slot

REFERENCES

[1] S. Kemp. (2023). *The Changing World of Digital in 2023*. We Are Social. Accessed: Feb. 16, 2023. [Online]. Available: https://wearesocial.com/uk/blog/2023/01/the-changing-world-of-digital-in-2023/

[2] M. Osman. (2022). *Wild and Interesting Facebook Statistics and Facts (2022)*. Accessed: Mar. 2, 2023. [Online]. Available: https://kinsta.com/blog/facebook-statistics/

[3] K. Smith. (2020). *60 Incredible and Interesting Twitter Stats and Statistics*. Accessed: Jan. 26, 2023. [Online]. Available: https://www.brandwatch.com/blog/twitter-stats-and-statistics/

[4] D. Yuan, X. Liu, and Y. Yang, "Dynamic on-the-fly minimum cost benchmarking for storing generated scientific datasets in the cloud," *IEEE Trans. Comput.*, vol. 64, no. 10, pp. 2781–2795, Oct. 2015, doi: 10.1109/TC.2015.2389801.

[5] M. D. Assunção, R. N. Calheiros, S. Bianchi, M. A. S. Netto, and R. Buyya, "Big data computing and clouds: Trends and future directions," *J. Parallel Distrib. Comput.*, vol. 79, pp. 3–15, May 2015, doi: 10.1016/j.jpdc.2014.08.003.

[6] W. Li, Y. Yang, and D. Yuan, "Ensuring cloud data reliability with minimum replication by proactive replica checking," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1494–1506, May 2016, doi: 10.1109/TC.2015.2451644.

[7] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *J. Netw. Comput. Appl.*, vol. 64, pp. 229–238, Apr. 2016, doi: 10.1016/j.jnca.2016.02.005.

[8] Z. Wu, M. Butkiewicz, D. Perkins, E. Katz-Bassett, and H. V. Madhyastha, "CSPAN: Cost-effective geo-replicated storage spanning multiple cloud services," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 545–546, Sep. 2013, doi: 10.1145/2534169.2491707.

[9] H. Khalajzadeh, D. Yuan, J. Grundy, and Y. Yang, "Cost-effective social network data placement and replication using graph-partitioning," in *Proc. IEEE Int. Conf. Cogn. Comput. (ICCC)*. IEEE, 2017, pp. 64–71.
[10] N. K. Gill and S. Singh, "A dynamic, cost-aware, optimized data replication strategy for heterogeneous cloud data centers," *Future Gener. Comput. Syst.*, vol. 65, pp. 10–32, Dec. 2016, doi: 10.1016/j.future.2016.05.016.
[11] D.-W. Sun, G.-R. Chang, S. Gao, L.-Z. Jin, and X.-W. Wang, "Modeling a dynamic data replication strategy to increase system availability in cloud computing environments," *J. Comput. Sci. Technol.*, vol. 27, no. 2, pp. 256–272, Mar. 2012, doi: 10.1007/s11390-012-1221-4.
[12] G. Liu, H. Shen, and H. Chandler, "Selective data replication for online social networks with distributed datacenters," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 8, pp. 2377–2393, Aug. 2016, doi: 10.1109/TPDS.2015.2485266.
[13] J. D. Brutlag, H. Hutchinson, and M. Stone, "User preference and search engine latency," in *Proc. JSM Quality Productiv. Res. Sect.*, Alexandria, VA, USA, 2008, pp. 1–13.
[14] R. Kuschnig, I. Kofler, and H. Hellwagner, "Improving Internet video streaming performance by parallel TCP-based request-response streams," in *Proc. 7th IEEE Consum. Commun. Netw. Conf.*, Jan. 2010, pp. 1–5, doi: 10.1109/CCNC.2010.5421815.
[15] L. Jiao, J. Li, T. Xu, W. Du, and X. Fu, "Optimizing cost for online social networks on geo-distributed clouds," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 99–112, Feb. 2016, doi: 10.1109/TNET.2014.2359365.
[16] D. A. Tran and T. Zhang, "S-PUT: An EA-based framework for socially aware data partitioning," *Comput. Netw.*, vol. 75, pp. 504–518, Dec. 2014, doi: 10.1016/j.comnet.2014.08.026.
[17] L. Jiao, T. Xu, J. Li, and X. Fu, "Latency-aware data partitioning for geo-replicated online social networks," in *Proc. Workshop Posters Demos Track*. Lisbon, Portugal: ACM, Dec. 2011, pp. 1–2.
[18] Z. Ye, S. Li, and J. Zhou, "A two-layer geo-cloud based dynamic replica creation strategy," *Appl. Math. Inf. Sci.*, vol. 8, no. 1, pp. 431–440, Jan. 2014, doi: 10.12785/amis/080154.
[19] N. Mansouri, M. K. Rafsanjani, and M. M. Javidi, "DPRS: A dynamic popularity aware replication strategy with parallel download scheme in cloud environments," *Simul. Model. Pract. Theory*, vol. 77, pp. 177–196, Sep. 2017, doi: 10.1016/j.simpat.2017.06.001.
[20] S.-Q. Long, Y.-L. Zhao, and W. Chen, "MORM: A multi-objective optimized replication management strategy for cloud storage cluster," *J. Syst. Archit.*, vol. 60, no. 2, pp. 234–244, 2014, doi: 10.1016/j.sysarc.2013.11.012.
[21] N. Mansouri, "Adaptive data replication strategy in cloud computing for performance improvement," *Frontiers Comput. Sci.*, vol. 10, no. 5, pp. 925–935, Oct. 2016, doi: 10.1007/s11704-016-5182-6.
[22] J. Matt, P. Waibel, and S. Schulte, "Cost- and latency-efficient redundant data storage in the cloud," in *Proc. IEEE 10th Conf. Service-Oriented Comput. Appl. (SOCA)*, Nov. 2017, pp. 164–172, doi: 10.1109/SOCA.2017.30.
[23] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, "CHARM: A cost-efficient multi-cloud data hosting scheme with high availability," *Int. J. Control Theory Appl.*, vol. 9, no. 27, pp. 461–468, 2016.
[24] H. Khalajzadeh, D. Yuan, B. B. Zhou, J. Grundy, and Y. Yang, "Cost effective dynamic data placement for efficient access of social networks," *J. Parallel Distrib. Comput.*, vol. 141, pp. 82–98, Jul. 2020, doi: 10.1016/j.jpdc.2020.03.013.
[25] S. Han, B. Kim, J. Han, K. Kim, and J. Song, "Adaptive data placement for improving performance of online social network services in a multicloud environment," *Sci. Program.*, vol. 2017, pp. 1–17, Aug. 2017.
[26] M. Uluyol, A. Huang, A. Goel, M. Chowdhury, and H. V. Madhyastha, "Near-optimal latency versus cost tradeoffs in geo-distributed storage," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement.*, 2020, pp. 157–180.
[27] Z. Wu, C. Yu, H. V. Madhyastha, and U. Riverside, "CosTLO: Cost-effective redundancy for lower latency variance on cloud storage services," in *Proc. 12th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2015, pp. 543–557.
[28] Y. Cui et al., "TailCutter: Wisely cutting tail latency in cloud CDNs under cost constraints," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1612–1628, Aug. 2019, doi: 10.1109/TNET.2019.2926142.
[29] K. Lee, R. Pedarsani, and K. Ramchandran, "On scheduling redundant requests with cancellation overheads," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 1279–1290, Apr. 2017.
[30] H. Wang, H. Shen, Z. Li, and S. Tian, "GeoCol: A geo-distributed cloud storage system with low cost and latency using reinforcement learning," in *Proc. IEEE 41st Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2021, pp. 149–159, doi: 10.1109/ICDCS51616.2021.00023.
[31] Y. Sun, Z. Zheng, C. E. Koksal, K.-H. Kim, and N. B. Shroff, "Provably delay efficient data retrieving in storage clouds," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 585–593, doi: 10.1109/INFOCOM.2015.7218426.
[32] M. Hajjat, P. N. Shankaranarayanan, D. Maltz, S. Rao, and K. Sripanidkulchai, "Dynamic request splitting for interactive cloud applications," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 12, pp. 2722–2737, Dec. 2013, doi: 10.1109/JSAC.2013.131212.
[33] Y. Hu and D. Niu, "Reducing access latency in erasure coded cloud storage with local block migration," in *Proc. IEEE INFOCOM 35th Annu. IEEE Int. Conf. Comput. Commun.*, Apr. 2016, pp. 1–9.
[34] A. Qureshi, "Power-demand routing in massive geo-distributed systems," Dept. Elect. Eng. Comput. Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, Tech. Rep., 2012. [Online]. Available: http://hdl.handle.net/1721.1/62430
[35] C. Kim and S.-U. Yang, "Like, comment, and share on facebook: How each behavior differs from the other," *Public Relations Rev.*, vol. 43, no. 2, pp. 441–449, Jun. 2013, doi: 10.1016/j.pubrev.2017.02.006.
[36] Y. Mansouri, A. N. Toosi, and R. Buyya, "Cost optimization for dynamic replication and migration of data in cloud data centers," *IEEE Trans. Cloud Comput.*, vol. 7, no. 3, pp. 705–718, Jul. 2019, doi: 10.1109/TCC.2017.2659728.
[37] S. Muralidhar, W. Lloyd, S. Roy, C. Hill, E. Lin, W. Liu, S. Pan, S. Shankar, V. Sivakumar, L. Tang, and S. Kumar, "f4: Facebook's warm BLOB storage system," in *Proc. 11th USENIX Symp. Operating Syst. Design Implement.*, 2014, pp. 383–398.
[38] D. Cheng, J. Rao, C. Jiang, and X. Zhou, "Resource and deadline-aware job scheduling in dynamic Hadoop clusters," in *Proc. IEEE Int. Parallel Distrib. Process. Symp.*, May 2015, pp. 956–965, doi: 10.1109/IPDPS.2015.36.
[39] J. Constine. (2012). *Study: Facebook Pages Shouldn't Post More Than 1x Every 3 Hours*. TechCrunch. Accessed: Jun. 21, 2022. [Online]. Available: https://techcrunch.com/2012/01/17/how-often-should-facebook-pages-post/
[40] R. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, and R. Buyya, "CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," *Softw., Pract. Exp.*, vol. 41, no. 1, pp. 23–50, Aug. 2011.
[41] T. Sturm, "Implementation of a simulation environment for cloud object storage infrastructures," Steinbuch Centre Comput., Karlsruhe Inst. Technol., Karlsruhe, Germany, Tech. Rep., 2013.
[42] A.Y. Aldailamy, A. Muhammed, R. Latip, N. A. W. A. Hamid, and W. Ismail, "Online dynamic replication and placement algorithms for cost optimization of online social networks in two-tier multi-cloud," *J. Netw. Comput. Appl.*, vol. 224, Jan. 2024, Art. no. 103827, doi: 10.1016/j.jnca.2024.103827.
[43] S. Moro, P. Rita, and B. Vala, "Predicting social media performance metrics and evaluation of the impact on brand building: A data mining approach," *J. Bus. Res.*, vol. 69, no. 9, pp. 3341–3351, Sep. 2016, doi: 10.1016/j.jbusres.2016.02.010.
[44] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny, "Workload analysis of a large-scale key-value store," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 40, no. 1, pp. 53–64, Jun. 2012, doi: 10.1145/2318857.2254766.

ALI Y. ALDAILAMY received the B.S. degree in information technology (IT) from the University of Modern Sciences (UMS), Yemen, in 2009, and the M.Sc. degree from the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), Malaysia, in 2017, where he is currently pursuing the Ph.D. degree with the Department of Computer networks. His research interests include networks, in-memory data stores, distributed systems, grids, and cloud computing.

ABDULLAH MUHAMMED received the bachelor's degree in computer science from Universiti Putra Malaysia, Malaysia, in 1998, the master's degree in computer science from Universiti Malaya, in 2004, and the Ph.D. degree in computer science from the University of Nottingham, U.K., in 2014. He is currently an Associate Professor with the Department of Communication Technology and Networks, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia. His research interests include grid/cloud computing, wireless sensor networks/IoT, heuristics, and optimization.

NOR ASILAH WATI ABDUL HAMID (Senior Member, IEEE) received the Ph.D. degree from The University of Adelaide, in 2008. She has been a Visiting Scholar with the High Performance Computing Laboratory, George Washington University, Washington, DC, USA, for two years. She is currently an Associate Professor with the Department of Communication Technology and Network, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Malaysia, and the Distributed and High-Performance Computing (DHPC) Group, working on high-performance distributed and parallel computing technologies and applications. She is also an Associate Researcher and a Coordinator of high-speed machines with the Institute for Mathematical Research (INSPEM), Universiti Putra Malaysia. Her research interests include parallel and distributed computing, cluster computing, distributed information systems, and other applications of high-performance computing.

ROHAYA LATIP received the bachelor's degree in computer science from Universiti Teknologi Malaysia, in 1999, and the M.Sc. degree in distributed systems and the Ph.D. degree in distributed database from Universiti Putra Malaysia (UPM). From 2011 to 2012, she was the Head of the HPC Section, UPM. She consulted the Campus Grid Project and the Wireless for Hostel in Campus UPM Project. She is currently an Associate Professor with the Faculty of Computer Science and Information Technology, UPM. She is also the Head of the Department of Communication Technology and Networks and a Co-Researcher with the Institute for Mathematical Research (INSPEM). Her research interests include big data, cloud and grid computing, network management, and distributed databases.

WAIDAH ISMAIL received the B.Sc. degree (Hons.) from the University of Liverpool, U.K., the master's degree from Universiti Teknologi MARA, Malaysia, and the Ph.D. degree in information system and computing from Brunel University, U.K. She is currently a Lecturer with Universiti Sains Islam Malaysia. She worked as a Lecturer for 14 years. Prior to this, she worked in the banking industry as a programmer, a system analyst, and a security analyst, for 11 years. Her research interests include optimization, classification, and grid/cloud computing.