



Popularity-based data placement in erasure-coded Edge Storage Systems

Cosimo Anglano¹ · Massimo Canonico^{1,3} · Rossano Gaeta² · Marco Guazzone^{1,3}

Received: 19 December 2024 / Revised: 9 July 2025 / Accepted: 3 August 2025
© The Author(s) 2025

Abstract

The explosive growth of smart devices and mobile users has led to an unprecedented demand for efficient data storage and processing. Edge computing has emerged as a promising solution to address these challenges by bringing computation and storage resources closer to data sources. Edge Storage Systems (ESSs) are a specific type of edge computing system designed to store and serve data efficiently at the network edge. While replication-based ESSs are a simple and effective approach, erasure-coded ESSs offer superior resilience, performance, and storage efficiency. However, effective data placement in erasure-coded ESSs is a complex problem, as it involves balancing factors such as data popularity, network latency, and resource utilization. In this paper, we propose a novel data placement approach for erasure-coded ESSs that addresses these challenges. Our approach consists of (1) a data placement strategy that leverages a virtual-space-based placement technique and incorporates data popularity into the placement decision, and (2) a dynamic data retrieval strategy that retrieves data from nodes with less load (in terms of data requests to be served). We evaluate our approach through extensive simulations, considering various network topologies, user mobility patterns, and data access patterns. Our results demonstrate that our proposed approach suitably addresses the data placement challenges arising from using erasure-coded ESSs, while allowing to exploit their advantages with respect to replication-based ESSs in terms of both space efficiency and data availability.

Keywords Edge storage systems · Erasure coding · Data placement · Data popularity

1 Introduction

The explosive growth of smart devices and mobile users that generate very large amounts of data to be processed in (near) real-time, has brought out the need of systems able to respond to storage and processing needs of these data. Edge computing [1] has risen in the past few years as solution to these needs. By placing computing, storage, and networking resources at the edge of the cloud [2], it reduces response times, improves processing efficiency, and alleviates network pressure.

Among edge systems, *Edge Storage Systems* (ESSs) [3–6] have been proposed as a way to satisfy the storage needs of smart devices and mobile users by bringing data closer to where these data are generated and/or consumed, so that these data can be accessed more efficiently than in traditional cloud storage systems. These latter systems, indeed, have been designed for scenarios where data are stored in a central location, so they fail to provide adequate

✉ Marco Guazzone
marco.guazzone@uniupo.it

Cosimo Anglano
cosimo.anglano@uniupo.it

Massimo Canonico
massimo.canonico@uniupo.it

Rossano Gaeta
rossano.gaeta@unito.it

¹ Dipartimento di Scienze e Innovazione Tecnologica, Università del Piemonte Orientale, Viale T. Michel 11, 15100 Alessandria, Italy

² Dipartimento di Informatica, Università di Torino, Corso Svizzera 185, 10141 Torino, Italy

³ Consorzio Nazionale Interuniversitario per le Telecomunicazioni (CNIT), Viale G.P. Usberti 181/A, 43124 Parma, Italy

performance [7, 8] for devices operating at the edge of the network.

Unlike cloud storage systems, ESSs exploit a set of computing and storage nodes (the *edge nodes*), placed at the edge of the network, to store and serve data items in such a way to keep data close (in terms of network distance) to the location from which they will be requested.

Edge computing systems in general, and ESSs in particular, however, exhibit peculiar characteristics – namely resource fluctuation and heterogeneity [1] – which need to be adequately taken into account to provide satisfactory levels of performance and availability. As a matter of fact, edge nodes are typically user-controlled, so they are usually shared among multiple applications, causing the performance delivered to an ESS to fluctuate over time. Furthermore, they can leave the edge system at any time, causing unavailability of the data they store. ESSs need therefore to provide suitable mechanisms enabling them to achieve adequate levels of performance and availability in face of node disconnections and performance fluctuations.

ESSs typically exploit one of two data storage mechanisms, namely *replication* and *erasure coding*.

In *Replication-based* ESSs (*R-ESSs*) [9, 10], several replicas of each data item are created and stored in the system. Performance is pursued by sending a request for a data item to the edge node which is closest to the user requesting it, thus reducing network latency, and by balancing the request load among all the nodes storing a replica of that data item. Availability is instead achieved by the presence of many equivalent replicas of the same data item, so the departure of an edge node from the system does not make that item unavailable.

In contrast, in *Erasure-Coded* ESSs (*EC-ESSs*) [4, 5, 11–14], each data item d_i is first split into k_i equally-sized blocks (the *systematic fragments*), and then $f_i = k_i \cdot n$ *coded fragments* (with $n \geq 1$) are generated by an erasure-coding algorithm [15] in such a way that any subset of k_i of them is sufficient to reconstruct d_i . The resulting coded fragments are stored on distinct edge nodes.

EC-ESSs offer several compelling benefits with respect to R-ESSs, namely:

- *enhanced resilience*: up to $f_i - k_i$ fragments of any data item d_i can become unavailable, e.g., because the corresponding storage nodes become unavailable (a common situation in edge computing scenarios), without compromising the availability of d_i ;

- *better performance*: the k_i coded fragments to reconstruct d_i can be retrieved in parallel, since they are stored on distinct edge nodes, and individual transfers of fragments last less than the entire data item because of their smaller size;
- *storage optimization*: less storage space is sufficient to achieve the same level of fault tolerance, which is crucial for resource-constrained edge devices.

In spite of that, achieving good performance in EC-ESS is not trivial because of the complexity of placing coded fragments on edge nodes in such a way to minimize latency, reduce network bandwidth usage, ensure data availability, and optimize resource utilization in face of data access patterns, workload characteristics, storage capacity constraints, and network conditions [5, 10, 16].

This problem is known in the literature as the *data placement* [3], and is known to be NP-hard [5]. This prevents the application of exact methods for realistic sizes of the system and of the dataset. Consequently, most approaches proposed in the literature [5, 9, 17–19] formulate it as an optimization problem, which is solved either optimally for small-scale systems, or by using specific approximate solutions or special-purpose heuristics for more realistic system sizes. These approaches, however, are characterized by two drawbacks [20]: (1) they suffer from poor stability and high computation costs, and (2) they do not take into account the *popularity* of data, which instead is known to be a crucial factor in the design of data placement algorithms.

1.1 Our contribution

In this paper, we focus on the data placement problem for EC-ESS, and we propose a data placement approach for EC-ESSs which addresses the two issues mentioned above by (a) taking into account data popularity when placing data, and (b) computing allocations in polynomial time without resorting to approximations or special-purpose heuristics. To the best of our knowledge, this work is the first one in which the EC-ESS data placement problem is solved by taking into account data popularity.

Our data placement approach combines data popularity and erasure-coding by coupling a virtual-space-based data placement strategy (along the lines of [5, 10, 16]), which is characterized by polynomial time complexity, with a dynamic data retrieval strategy which reduces data access times.

We evaluate the performance of our data placement approach via a thorough discrete-event simulation experimental campaign, whereby a set of realistic scenarios, characterized by different topologies of the edge system, user mobility patterns, and data request patterns, have been considered.

Two features differentiate our study from previous work, that is:

1. our simulation model considers a more comprehensive system, accounting for request processing time, network delays, request queuing, and the dynamic nature of user behavior, such as user mobility and request generation patterns;
2. we exploit a model of the request load which allows us to study performance in more realistic settings where spatial correlation of the requests is combined with temporal correlation among data item requests [21]. In particular, we use *multivariate spatiotemporal Hawkes processes* [22, 23] as a way to characterize mutual influence among events, whereby the occurrence of an event may increase the probability for subsequent events to occur. This mathematical tool allows us to simultaneously take into account user mobility (which creates spatial correlation)

and content topic similarity (which creates temporal correlation among data items requests).

Our findings indicate that, as we expected, the data placement and retrieval strategies we propose exhibit robust performance across all simulated scenarios. In particular, the adaptation of the virtual space placement algorithm from [10] to erasure-coded systems, in conjunction with the Join-the-Shortest-Queue data retrieval strategy [24], empowers EC-ESS to effectively accommodate dynamic user movements and request generation under realistic system conditions.

Hence, our proposed approach suitably addresses the data placement challenges arising from using erasure-coding, while allowing to exploit their advantages with respect to R-ESS in terms of both space efficiency and data availability.

In summary, the contributions of this paper are as follows:

1. we propose a polynomial time virtual-space-based data placement approach for EC-ESSs, which takes into account the popularity of data, the logical topology of the edge system, and the dynamic request patterns issued by a population of mobile users;
2. we study the performance of our method for a wide range of scenarios, which include several network topologies characterized by different parameters, different data access patterns, and a mobile user population;
3. we show that our data placement approach exhibits good performance in all the scenarios considered in the experimental study.

Table 1 summarizes the notation used throughout the rest of the paper.

The rest of the paper is organized as follows: Sect. 2 describes the system architecture, Sect. 3 defines the problem we tackle, Sect. 4 describes the data placement algorithm, Sect. 5 discusses the data retrieval strategy, Sect. 6 presents the performance evaluation we carried out. Finally, Sect. 7 discusses a selection of previous works related to ours while Sect. 8 draws conclusions and outlines future directions for further analysis.

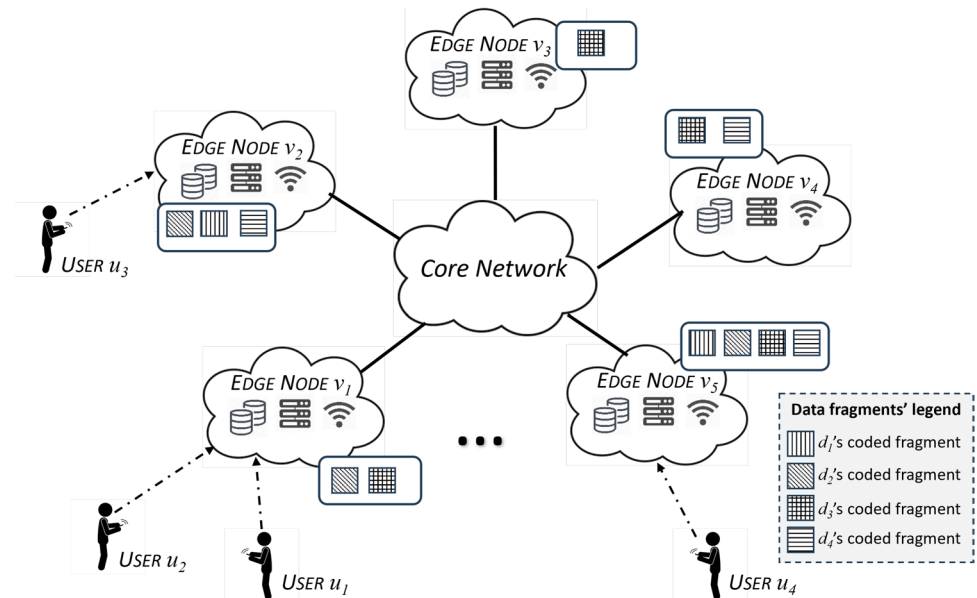
2 System architecture

In this work, we consider an EC-ESS, schematically depicted in Fig. 1. Our architecture is similar to what proposed by scientific literature discussing the use of edge storage systems for real-world applications. In

Table 1 Paper notation

Symbol	Meaning
D	Set of data items
W	Number of data items in D
d_i	Data item i
s_i	Size in bytes of data item d_i
k_i	Number of systematic fragments of data item d_i
f_i	Number of coded fragments of data item d_i ($f_i = k_i \cdot n$)
n	Redundancy factor
S	Size in bytes of a fragment
g	Grouping factor (expressed as number of coded fragments)
$G(d_i)$	Set of coded fragment groups of data item d_i
$c_{i,j}$	j^{th} coded fragment group of data item d_i
A_i	Set of edge nodes storing fragment groups of d_i
p_i	Popularity of data item d_i
V	Set of edge nodes
N	Number of edge nodes in V
v_j	Edge node j
$\delta(v_i)$	Storage capacity (expressed in fragments) of edge node v_i
$\gamma(v_i, g)$	Data access time to retrieve g fragments from the storage subsystem of edge node v_i
$(x(v_i), y(v_j))$	The virtual coordinates of edge node v_i
$(x'(c_{i,j}), y'(c_{i,j}))$	The virtual coordinates of coded fragment $c_{i,j}$

Fig. 1 Architecture of the EC-ESS. Each *edge node* v_j is equipped with computing, storage, and network resources. *Encoded data fragments* are represented by filled squares, and different *data items* d_i correspond to different filling patterns. Dotted arrows connect each user u_k to the corresponding *frontier node*. Edge nodes communicate with each other through the *core network*



particular, [25] considers multi-camera real-time vision applications, while the approach proposed in [26] is applied for applications related to IoT, smart grid and autonomous driving. Applications related to smart cities and social sensing are used in [27]. Furthermore, the approach proposed in [28] is focused on applying erasure coding in large-scale overlay networks. Finally, [29] focuses its research on application domains such as healthcare and traffic management.

In our study, a set of W data items $D = \{d_1, d_2, \dots, d_W\}$ are stored on a set of N edge nodes $V = \{v_1, v_2, \dots, v_N\}$.

Each data item d_i is associated with its size in bytes s_i and its *popularity* p_i (which measures how much a given piece of data is requested by the users in the system).¹

Each data item d_i is encoded by means of an erasure coding algorithm, which first splits d_i into k_i systematic fragments of equal size S , and then generates $f_i = k_i \cdot n$ *coded fragments* ($n \geq 1$) in such a way that any subset of k_i coded fragments are sufficient to reconstruct d_i . The factor n is named *redundancy factor* and determines how many additional fragments are created for each data item. For example, if $n = 1.5$ and $k_i = 10$, the erasure encoding algorithm generates $f_i = 15$ coded fragments out of 10 systematic fragments, and any subset of size 10 of these 15 coded fragments suffice to reconstruct d_i .

To maximize data availability, each fragment of data item d_i is distributed across distinct edge nodes. For instance, in Fig. 1, coded fragments of data item d_4 are stored on edge nodes v_2 , v_4 and v_5 . When data item d_i is requested, a recovery threshold of k_i is enforced,

meaning that any subset of k_i nodes can be contacted to retrieve the necessary coded fragments and reconstruct d_i . This, however, mandates that the number of edge nodes N in the EC-ESS exceeds the maximum number of fragments $f_{\max} = \max_{i=1}^W f_i$ for any data item. In certain cases, however, N may be less than f_{\max} .

To address scenarios where $N < f_{\max}$, we group the coded fragments of each data item d_i into sets of size g (the *grouping factor*). Each group, rather than a single fragment, is stored on a distinct edge node. This reduces the minimum required number of nodes N to $\lceil f_{\max}/g \rceil$ compared to f_{\max} . However, it comes at the cost of reduced reliability: a single node failure results in the loss of g fragments instead of one. We denote $G(d_i)$ as the set of coded fragment groups for data item d_i , and $A_i \subseteq V$ as the set of edge nodes storing these groups.

Each edge node v_i in the ESS is positioned within a two-dimensional space, represented by Cartesian coordinates (x_i, y_i) . Each node is characterized by its *storage capacity* $\delta(v_i)$, the maximum number of fragments it can store, and its *data access time* $\gamma(v_i, g)$, the time to retrieve a fragment group of size g from the node's storage. For simplicity, we assume a uniform data access time across all nodes: $\gamma(v_i, g) = \gamma(g), \forall v_i \in V$.

Edge nodes communicate via the *core network*. To simplify our analysis, we assume that edge nodes remain continuously connected to the system, ensuring that all f_i encoded fragments of data item d_i are always accessible.

¹ In this work, we assume p_i corresponds to the number of accesses to d_i over time. However, it is not difficult to replace this definition with alternative ones discussed in the literature [20].

We consider a population of users that moves across the virtual space according to a *user mobility process*. These users generate requests for data items following a *data request process*.

A user u_i sends a request to the nearest edge node v_j in Euclidean distance, termed as the *frontier node* for user u_i . For instance, in Fig. 1, the frontier node for user u_3 is the edge node v_2 . Upon receiving a request from user u_i , edge node v_j either:

1. sends the requested fragment groups directly to u_i if they are stored locally, or
2. forwards the request (through the core network) to other edge nodes that may store the required fragments.

Incoming requests at an edge node are queued if the node is currently busy, and processed in a first-come, first-served manner.

As already proposed in the literature [10, 16], we assume that the system we consider is deployed as a software-defined edge infrastructure, which typically consists of at least two planes, namely the control plane and the switch plane. Specifically, the control plane provides the control logic, including the virtual space for both edge nodes and data items as well as the data placement and data retrieval strategies, and generates forwarding rules for switches in the switch plane. The switch plane includes the switches of the core network and provides communication services among edge nodes according to the forwarding rules generated by the control plane.

3 Problem definition

As previously discussed, the coded fragment groups of data item d_i must be stored across a set $A_i \subseteq V$ of $\lceil f_i/g \rceil$ edge nodes. To minimize the *data retrieval time (DRT)* for d_i – the time elapsed from request to complete fragment group reception – we must carefully select the nodes in A_i .

DRT is influenced by two primary factors:

1. *Request network latency*: the time taken for a request to reach the nodes in A_i and for the retrieved fragments to return to the user;

2. *Request processing time*: the time spent by edge nodes processing requests and retrieving fragments.

To minimize DRT, both request network latency and request processing time must be minimized. To minimize network latency, popular data items should ideally be placed on edge nodes closer to their frequent requesters. However, this can lead to load imbalance, as popular items concentrate on a subset of nodes, increasing their processing load and queue lengths. Conversely, uniform data placement reduces load imbalance but increases network latency. Further complicating the issue, user mobility means that requests for a given item may be routed through different frontier nodes over time, making static data placement strategies less effective. Balancing these competing factors is crucial for optimal system performance.

To address this problem, we decompose it into two sub-problems:

1. *Data Placement*: determining the optimal placement of coded fragments for each data item d_i across a set A_i of $\lceil f_i/g \rceil$ edge nodes. We propose a data placement algorithm that leverages data popularity and network topology to minimize network latency by strategically allocating fragments to nodes with lower latency.
2. *Data Retrieval*: selecting a subset \bar{A}_i of $\lceil k_i/g \rceil$ nodes from A_i to fulfill a data request for d_i . Our data retrieval strategy dynamically chooses \bar{A}_i to balance the load across nodes.

In the rest of this section we will first discuss the data placement algorithm (Sect. 4), and then the data retrieval strategy (Sect. 5).

4 The data placement algorithm

Our data placement algorithm builds upon the approach presented in [10]. By mapping both edge nodes and data items to a virtual circular plane, we can assign fragment groups to edge nodes based on network latency and data popularity, aiming to minimize the request network latency. This technique, previously applied to replication-based ESSs in [10], has demonstrated promising results.

In our algorithm, edge nodes are mapped to the virtual space such that the Euclidean distance between

nodes correlates with their actual network latency. This mapping ensures that central nodes in the virtual space have shorter paths to other regions, particularly in dense networks.²

To minimize request network latency, we place more popular fragments closer to the center of the virtual plane. This is achieved by assigning fragment coordinates inversely proportional to their popularity. By assigning fragments to the nearest edge node in the virtual space, we effectively place popular fragments on central nodes with shorter paths.

Our algorithm operates in two phases:

1. *Coordinate Assignment*: calculates coordinates for both fragments and edge nodes.
2. *Fragment Placement*: assigns fragments to edge nodes based on decreasing popularity, prioritizing central nodes.

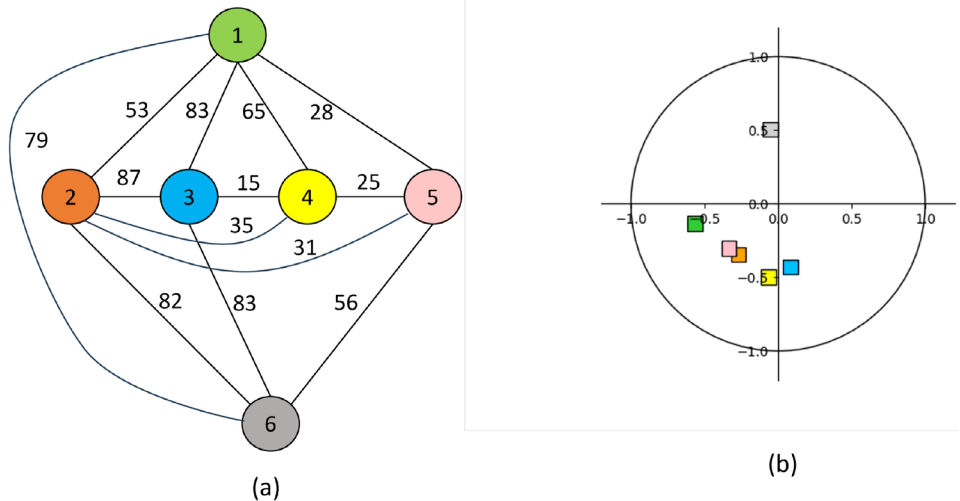
Compared to complex optimization-based approaches, our virtual space method offers a simpler and efficient solution with a time complexity of $O(W \cdot N)$.

The following sections provide detailed descriptions of these phases.

4.1 Virtual coordinates calculation

The computation of the coordinates of edge nodes and data items in the virtual space (their *virtual coordinates*) follows two different principles.

Fig. 2 (a) Network Topology: edge nodes and their latency-weighted connections. (b) Virtual Space Placement: visualization of edge nodes mapped to a virtual space. Color-coded for direct comparison with the network topology



We assign virtual coordinates to both edge nodes and data items based on distinct principles:

- **data items**: fragment groups of a data item d_i are placed closer to the center of the virtual plane as the popularity p_i increases;
- **edge nodes**: edge nodes are positioned in the virtual plane such that the Euclidean distance between any two nodes reflects their actual network distance.

4.1.1 The coordinates of edge nodes

We employ the M-position algorithm [10, 16, 31] to assign virtual coordinates to edge nodes. This algorithm leverages the network topology graph $G = (V, E)$, where the weight of each edge (v_i, v_j) represents the network latency between the corresponding edge nodes, and computes the virtual coordinates $(x(v_i), y(v_i))$ for each node v_i in $O(N^3)$ time. For a detailed explanation, please refer to the cited literature.

Figure 2 illustrates an example of edge node placement in the virtual space, where Fig. 2(a) shows the network topology and Fig. 2(b) shows the corresponding virtual coordinates.

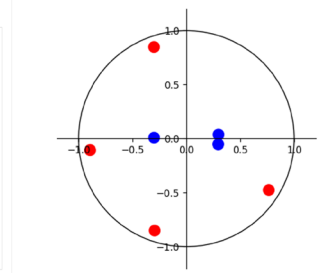
4.1.2 The coordinates of data items

Given data item d_i , we assign polar coordinates $(r(c_{i,j}), \theta(c_{i,j}))$ to each of its coded fragment groups $c_{i,j}$ ($j = 1, \dots, g$) as follows.

² The *density* of a network assesses the density of edges between nodes of the network, and is the quantitative relation of the total number of edges in the network to the maximum number of edges that the network can accommodate [30].

Fig. 3 (a) Table reporting the characteristics of data items d_1 and d_2 , where we use $H(i) = 0x\text{FFFFFFF}$ & SHA1(i)/($2^{32} - 1$) and where SHA1(i) is the SHA-1 hash value of i in binary form; **(b)** placement on the virtual plane of data items, where red and blue small circles correspond to fragment groups of d_1 and d_2 , respectively

Data item i	Num. of fragment groups	Popularity	H(i)	Root angle	Differential angle
1	4	100	0.7169	$\theta_{\alpha}(1) = 2\pi \cdot 0.7169$	<ul style="list-style-type: none"> • $\theta_{\beta}(1,1) = \pi/2$ • $\theta_{\beta}(1,2) = \pi$ • $\theta_{\beta}(1,3) = (3/4) \cdot \pi$ • $\theta_{\beta}(1,4) = 2 \cdot \pi$
2	3	700	0.9222	$\theta_{\alpha}(2) = 2\pi \cdot 0.9222$	<ul style="list-style-type: none"> • $\theta_{\beta}(2,1) = (2/3) \cdot \pi$ • $\theta_{\beta}(2,2) = (4/3) \cdot \pi$ • $\theta_{\beta}(2,3) = 2 \cdot \pi$



(a)

(b)

The radius $r(c_{i,j})$ is inversely proportional to the popularity of d_i , placing more popular items closer to the center of the virtual plane. Specifically, $r(c_{i,j}) = 1 - p_i/p_{\max}$, where p_i is the popularity of d_i and p_{\max} is the maximum popularity.

The angle $\theta(c_{i,j})$ is calculated as $(\theta_{\alpha}(i) + \theta_{\beta}(i,j))/2\pi$, where:

- *root angle* $\theta_{\alpha}(i)$: this angle, determined by the index i of the data item d_i , ensures that different data items are distributed throughout the circular space. We compute it as $\theta_{\alpha}(i) = 2\pi \cdot H(i)$, where $H(i)$ is a hash-based function that maps i to a value in the range $[0, 1]$;
- *differential angle* $\theta_{\beta}(i,j)$: this angle, specific to each fragment group $c_{i,j}$, disperses the fragment groups of a single data item across different directions. It is calculated as $\theta_{\beta}(i,j) = j \cdot 2\pi / \lceil f_i/g \rceil$.

This approach ensures that more popular data items are placed closer to the center of the virtual plane, while the fragment groups of a single data item are evenly distributed around the corresponding circle.

An example of calculation of data items coordinates is shown in Fig. 3, where we consider a simple scenario where two data items are placed on the virtual space in the hypothesis that their characteristics are shown in the table in Fig. 3(a), together with the various quantities computed by the algorithm discussed above, and that the maximum popularity value p_{\max} is 1000.

As can be seen in Fig. 3(b), the four fragment groups of d_1 (small red circles) are farther away from the center than the three of d_2 (small blue circles), since the former have a popularity smaller (100) than the latter (700).

Given the polar coordinates of a coded fragment group $c_{i,j}$, it is possible to convert them into the corresponding Cartesian coordinates $(x'(c_{i,j}), y'(c_{i,j}))$ by means of the usual transformation:

$$\begin{cases} x'(c_{i,j}) = r(c_{i,j}) \cdot \cos \theta(c_{i,j}), \\ y'(c_{i,j}) = r(c_{i,j}) \cdot \sin \theta(c_{i,j}). \end{cases} \quad (1)$$

4.2 Data placement

To allocate fragment groups to edge nodes, we employ a simple nearest-neighbor approach. Each fragment group $c_{i,j}$ is assigned to the closest edge node v_x in the virtual space, as defined by the following mapping function:

$$\begin{aligned} \text{map}(c_{i,j}, V) \\ = \arg \min_{v_k \in V} \sqrt{(x(v_k) - x'(c_{i,j}))^2 + (y(v_k) - y'(c_{i,j}))^2} \end{aligned} \quad (2)$$

where $x'()$ and $y'()$ denote the Cartesian coordinates of the coded fragment groups, computed by means of Eq. 1.

This strategy effectively places popular data items closer to central nodes, which tend to have lower latencies, while less popular items are placed on nodes farther from the center. This trade-off balances latency and load, leading to improved overall performance. The resulting data placement algorithm (Algorithm 1) has an asymptotic time complexity of $O(W \cdot N)$, where N and W represent the number of edge nodes and data items, respectively.³

³ Note that, in Algorithm 1, the loop over $G(d_i)$ does not affect the asymptotic time complexity of the algorithm as the size of $G(d_i)$ is the same for any data item d_i , it is usually much smaller than N and W , and thus it can be treated as a constant factor in the time complexity analysis.

Input: the set of data items $D = \{d_1, \dots, d_W\}$, the W sets of coded groups $\{G(d_i) | d_i \in D\}$, the set of virtual coordinates $\{(x'(c_{i,j}), y'(c_{i,j})) | c_{i,j} \in G(d_i) \wedge d_i \in V\}$ of data items, the set of edge nodes V , and the set of virtual coordinates $\{(x(v_k), y(v_k)) | v_k \in V\}$ of edge nodes.

Output: Returns an associative array *Allocated* that, for each fragment group of any data item $d_i \in D$, stores the identifier of the edge node where it has been placed.

```

1: Allocated = {}
2: for all  $d_i \in D$  do
3:   for all  $c_{i,j} \in G(d_i)$  do
4:     Allocated[ $c_{i,j}$ ] = -1           ▷ Initialize associative array Allocated
5:   end for
6: end for
7:  $U = \{\}$ 
8: for all  $v_i \in V$  do
9:    $U[v_i] = 0$  ▷  $U[v_i]$  is the allocated storage capacity of  $v_i$  expressed as coded fragment
   groups
10: end for
11:  $L = \text{sort}(D)$            ▷ List of data items sorted in decreasing priority order
12: while  $L \neq \emptyset$  do
13:    $d_i = \text{pop}(L)$            ▷ Get the first element of  $L$  and remove it from the list
14:    $C = V$            ▷  $C$  is the set of candidate edge nodes for the allocation of  $d_i$ 
15:   for all  $c_{i,j} \in G(d_i)$  do
16:     found = false
17:      $v_x = \text{none}$ 
18:     while  $\neg \text{found} \wedge C \neq \emptyset$  do           ▷ Search loop
19:        $v_x = \text{map}(c_{i,j}, C)$ 
20:        $C = C - \{v_x\}$ 
21:       if  $(g + U[v_x]) \leq \delta(v_x)$  then
22:         found = true
23:       end if
24:     end while
25:     if found then
26:       Allocated[ $c_{i,j}$ ] =  $v_x$ 
27:        $U[v_x] = U[v_x] + g$ 
28:     else
29:       exit("Error: system storage capacity exhausted")
30:     end if
31:   end for
32: end while
33: return Allocated

```

Algorithm 1 Data placement algorithm

The algorithm takes as input:

- the set of data items D , where each data item d_i is associated with a set of fragment groups, each with its virtual coordinates;
- the set of edge nodes V , each with its corresponding virtual coordinates.

The output is an allocation array *Allocated* that assigns each fragment group to an edge node.

The algorithm initializes the *Allocated* array (lines 1–6) and a capacity tracking array U (lines 8–10). It then sorts the data items in D by decreasing popularity (line 11) and iterates over them (lines 12–32).

Then, for each data item d_i , the algorithm:

1. initializes a candidate node set C to V (line 14);
2. for each fragment group $c_{i,j}$ of d_i :
 - a) finds the closest node v_x in C to $c_{i,j}$ (line 19) and remove it from C (line 20);
 - b) if v_x has sufficient capacity (line 21), assigns $c_{i,j}$ to v_x , updates v_x 's capacity (line 27);
 - c) if no suitable node is found, terminates with an error (line 29).

The algorithm has an asymptotic time complexity of $O(W \cdot N)$, where N and W are the number of edge nodes and data items, respectively.

In Fig. 4 we graphically depict the allocation of the fragments of the two data items (see Fig. 3) on the six edge nodes (see Fig. 2) of our example.

5 The data retrieval strategy

When a user requests data item d_i , the EC-ESS can recover d_i by retrieving only k_i of its f_i coded fragments. To do so, it needs to contact only $\lceil k_i/g \rceil$ edge nodes in the set $A_i \subseteq V$ (we call these the *sub-requests* for the data item). The data retrieval strategy is designed to select these nodes in a way that minimizes the total waiting time experienced by all users. This means that the complexity of the data retrieval strategy, measured as the number of messages that have to be sent to retrieve any data item, is $O(k_{max}/g)$, where k_{max} is the number of coded fragments of the largest data item d_{max} stored in the system.

The retrieval time of d_i is determined by the time it takes to receive the last of the $\lceil k_i/g \rceil$ coded fragment groups (the *request processing time*, see Sect. 3). This is because d_i can only be reconstructed after all $\lceil k_i/g \rceil$ groups have been received. The request processing time depends on both the *data access time* (i.e., the time to fetch a coded fragment group from a node's storage, see Sect. 2) and the *request waiting time* (i.e., the time a request spends in an edge node's queue).

Since all coded fragments are equivalent, the simplest data retrieval strategy is to randomly select $\lceil k_i/g \rceil$ nodes from A_i , and to send to each one of them the sub-requests for coded fragments of d_i . We call this the *Random retrieval policy (RRP)*. This strategy is trivial to implement, as each node only needs to maintain a mapping table specifying the set A_i for each data item d_i .

While RRP provides good load balancing for equally likely data item requests, it leads to unbalanced node loads when data item popularity is skewed. This imbalance arises due to the following factors:

1. *hot-spot nodes for popular data items*: popular data items, which attract most user requests, tend to be stored on edge nodes closer to the virtual space's center. These nodes, consequently, receive more requests and experience longer queues than nodes farther from the center, which store less popular items. This concentration of requests on a few nodes increases waiting and retrieval times for popular items;
2. *skewed popularity distribution*: a highly skewed popularity distribution, where a few data items dominate requests, exacerbates the hot-spot problem by concentrating requests on a limited number of edge nodes;
3. *co-location of popular and less popular data items*: in storage-constrained scenarios, popular and less popular items may be co-located on the same nodes. This can lead to delays in serving less popular items due to the higher demand for popular ones;
4. *hot-spot nodes for geographically clustered users*: when multiple users are located in the same area, their requests are initially directed to a common set of nearby edge nodes. This can create a bottleneck, as these nodes must process and forward a large number of requests.

To reduce the load imbalance caused by these factors, we replace RRP with the *Join-the-Shortest-Queue retrieval policy (JSQ-RP)* [24]. Under the JSQ-RP, the sub-requests for the coded fragment groups of d_i are sent to those nodes in A_i that – at the time of the request – exhibit the shortest number of requests waiting to be served. By routing sub-requests to nodes with less load, JSQ-RP tends to equalize the workload across them. This prevents any single node from becoming overloaded while others remain idle. Furthermore, when requests are assigned to nodes with shorter queues, they typically experience shorter wait times before being processed, improving the overall system responsiveness. Finally, as the load of nodes fluctuates, JSQ-RP can dynamically adjust to distribute the workload evenly.

JSQ-RP results in significant performance improvements with respect to the random selection of edge nodes, as demonstrated by our experimental results, discussed in Sect. 6.3.3.

6 Performance evaluation

To evaluate the performance of the data placement and retrieval strategies under realistic conditions, we conduct a comprehensive study based on discrete-event simulations in which we evaluate performance using the set of metrics defined in Sect. 6.1. In this study, we consider a set of scenarios – described in Sect. 6.2 – featuring different combinations of the parameters of the edge system, of the data

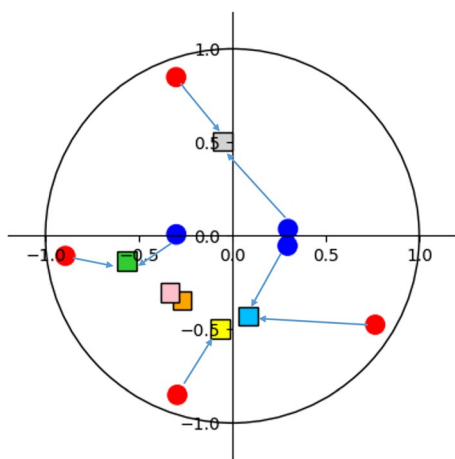


Fig. 4 Allocation of the fragments of the two data items of the example (Fig. 3) on the available edge nodes (Fig. 2). An arrow connects each fragment of a data item (a circle) to the edge node (a square) where it has been allocated

set, of the encoding scheme, and of the user mobility and request process.

6.1 Performance indices

For each simulation experiment, we compute the following performance metrics:

1. *Average Retrieval Time*: the average time from a user's request to the receipt of the last data fragment;
2. *Average Queueing Time*: the average time spent by data fragment requests in edge node queues;
3. *Average Network Time*: the average time spent by data fragment requests traversing the network.

For each metric, we calculate a 95% confidence interval with a relative error of $\pm 2.5\%$.

6.2 Simulation scenarios

In order to assess the performance yield by our data placement and data retrieval strategies, we consider a set of simulation scenarios, which cover a wide range of realistic systems.

These scenarios are generated by setting different values for the parameters that impact on performance. These parameters are classified into four groups, namely those relating to the edge system, to the data set, to the encoding scheme, and to the user mobility and request processes, respectively, and in particular:

- *edge system parameters*: number and storage capacity of nodes, core network topology;
- *data set parameters*: number, size, and popularity distribution of data items;
- *encoding scheme parameters*: redundancy and grouping factors;
- *user mobility and data request process parameters*: user mobility models and data request patterns.

Overall, we consider 180 distinct scenarios, obtained by setting these parameters for each one of these groups as detailed in the following.

6.2.1 Data set characteristics

We consider a dataset D of $W = 100$ data items, where each data item $d_i \in D$ has:

- *size k_i* : set as a random integer uniformly distributed between 10 and 100, which represents the number of its systematic fragments;

- *popularity p_i* : set as an integer number determined by the number of requests it receives during the simulation.

6.2.2 Encoding scheme characteristics

We set the coded fragment size at 1 MB and vary the redundancy factor n and grouping factor g . The redundancy factor n controls the number of edge nodes storing each data item and the required storage capacity. We consider values of n in the set $\{1, 1.5, 2, 2.5, 3\}$. The grouping factor g determines the minimum number of edge nodes needed to store the largest data item. We consider values of g in the set $\{1, 2, 4, 8, 16, 32\}$.

6.2.3 Edge system characteristics

In all the scenarios, we fix the number of edge nodes to $N = 400$, and we set the characteristics of each node $v_i \in V$ as follows:

- *storage capacity $\delta(v_i)$* : it is set as a random integer uniformly distributed between 50 and 100, representing the number of coded fragments v_i can store;
- *data access time $\gamma(v_i, g)$* : it is computed as $\gamma(v_i, g) = (-1066944 + 3.057912 \cdot g \cdot S)/10^9$ (s), where S is the size in bytes of each coded fragment; this expression has been experimentally determined by fitting a linear regression model to a dataset generated through the *fio* tool [32]. Specifically, by means of the *fio* tool, we generated an I/O workloads on a real Intel DC S3610 solid state drive [33], considering different chunk sizes (specified through the “bs” parameter – i.e., the block size used for I/O operations – of the *fio* tool and ranging from 1 MB to 1 GB), different mixes of read and write operations (specified through the “rwmixread” parameter – i.e., the percentage of a mixed workload that should be reads – of the *fio* tool and chosen from the set $\{0\%, 25\%, 50\%, 75\%, 100\%\}$ of percentage of read operations, and different mixes of sequential and random I/O operations (specified through the “percentage_random” parameter – i.e., the percentage of random workload that should be used in a mix of sequential and random I/O operations – of the *fio* tool and chosen from the set $\{0\%, 25\%, 50\%, 75\%, 100\%\}$ of percentage of random I/O operations). From the results of the above experiments, we extracted the values of the average total latency metric (as reported by the *fio* tool as the “lat_ns” metric – i.e., the time elapsed, on average, from the submission of an I/O operation to its completion) and created a dataset as a collection of “(average data access time, data chunk size)” pairs, where:

- the “data chunk size” column represents the size (in bytes) of the data chunk transferred in an I/O operation;
- the “average data access time” column represents the time (in nanoseconds) it took to perform an I/O operation, on average, from its submission to the completion.

We then fit a polynomial regression model to the above dataset (where the “data chunk size” is used as the independent variable and the “average data access time” is used as the dependent variable), considering different polynomial degrees (ranging from 1 to 4). A model selection analysis based on the Akaike Information Criterion (AIC) [34] revealed that the linear model provides a good fit. Note that the values of $\gamma(v_i, g)$ depends only on g and S .

The characteristics of the core network are set by varying as discussed below its three main characteristics which impact on the performance of data placement and retrieval, namely topology, latency, and bandwidth,

Data placement is affected by the density Δ of the topology, which is defined as $\Delta = \frac{2 \cdot E}{N \cdot (N-1)}$ [35]. In particular, the larger Δ , the larger the number of links among nodes, and the better the connections towards nodes in the central region of the virtual space.

We consider three density values: $\Delta \in \{0.1, 0.5, 0.9\}$, from which we build a random graph by proceeding as follows. For each value of Δ , we determine the number of edges E in the network graph as $E = \Delta \cdot \frac{N \cdot (N-1)}{2}$, and we randomly generate a graph with N nodes and E edges. To ensure the generality of our results, for each value of Δ we generate 750 random graphs with the same number of nodes and density, and we repeat the simulation for each scenario resulting from the combination of the other parameters on all these graphs. This allows us to average the results over different network topologies.

Data retrieval is instead affected by the latency and bandwidth of each link (v_i, v_j) , which are set as follows:

- *latency*: it is set as a random value uniformly distributed between 180 and 220 (ms), corresponding to an average value of 200 (ms), based on real-world latency measurements [36];
- *bandwidth*: it is set as a random value uniformly distributed between 40 and 60 Mbps, corresponding to an average value of 50 Mbps, based on real-world bandwidth measurements [37].

6.2.4 User mobility and data request processes

The mobility process determines the trajectory of users in space over time, while the request process determines the data items requested by users.

Our study differentiates from previous work by employing a novel approach that combines user mobility with data request patterns. We use a *multivariate spatiotemporal (MST) Hawkes process* [22, 23] to model both spatial and temporal correlations in user behavior. A MST Hawkes process is a self-exciting point process that captures the influence of past events on future events. In our context, an event corresponds to a user request for a data item at a specific location and point in time.

We consider two MST Hawkes process models, both using the same mobility process but a different request process:

1. *Hawkes-Exp*: data popularity is modeled by an exponential distribution, leading to a uniform distribution of requests across data items.
2. *Hawkes-Zipf*: data popularity follows a Zipf distribution with parameter $\alpha = -0.6$ (a typical choice in the literature [38, 39]), reflecting a skewed distribution where a few popular items dominate.

To generate event traces, we adapt the algorithm proposed in [23]. Each event in the trace is a tuple (t_r, x_r, y_r, d_r) , where t_r is the timestamp, x_r and y_r represent the user’s location, and d_r is the requested data item.

Once these traces are generated, we analyze the requests they contain to determine the popularity of the various data items in the system.

6.3 Results

In this section we present the results obtained in our performance evaluation campaign. We start with Sect. 6.3.1, where we present the results concerning the comparison between RRP data retrieval strategy with respect to a baseline strategy adopted from what proposed in [10]. Then in Sect. 6.3.2, we present the results concerning the coupling of the data placement algorithm with the RRP data retrieval strategy. Next, in Sect. 6.3.3, we discuss the results obtained with the JSQ-RP data retrieval strategy. Sect. 6.3.4 draws general conclusions from the results of the individual data retrieval strategies.

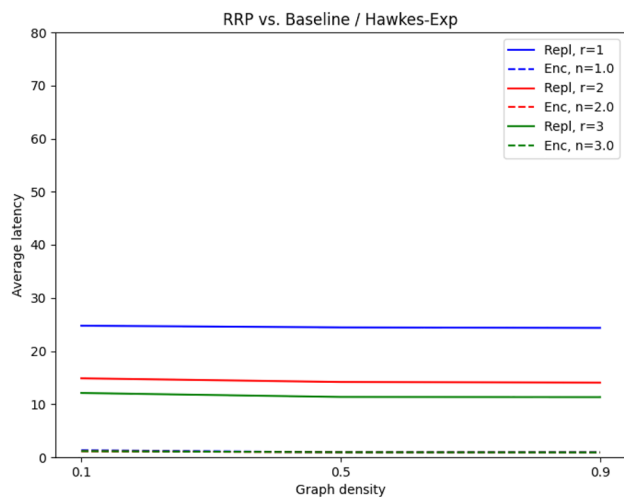


Fig. 5 RRP data retrieval strategy versus baseline strategy, *Hawkes-Exp* mobility and request model

6.3.1 Results for the RRP data retrieval strategy with respect to the baseline strategy

This section presents results concerning the use of the RRP data retrieval strategy with respect to a baseline strategy. In particular, since our work has been inspired by the popularity-aware data placement strategy proposed in [10] for a replication-based ESS, it follows that this strategy represents the natural baseline for a comparison. To this end, we considered scenarios where the number of replicas equals the redundancy factor to ensure a fair comparison wherein the same amount of storage resources is used to store a data item. Furthermore, the comparison against the baseline is performed by considering the simple random retrieval policy (RRP) for coded fragments.

Results for the average total latency under both the *Hawkes-Exp* (see Fig. 5) and *Hawkes-Zipf* (see Fig. 6) data popularity models, for $n = [1, 2, 3]$ and $r = [1, 2, 3]$, show that the proposed approach represents an improvement over the reference baseline.

6.3.2 Results for the RRP data retrieval strategy

This section presents results concerning the use of the RRP data retrieval strategy. We run experiments for both the *Hawkes-Exp* and the *Hawkes-Zipf* mobility and request model, and we conduct experiments across all values of n and g and Δ reported in Sect. 6.2. We show only the results for $n = [1, 2, 3]$, and we omit the results for $n = [1.5, 2.5]$ as they fall between those shown and would only add redundancy.

For each value of n , we report side-to-side the results corresponding to both mobility and request models. In

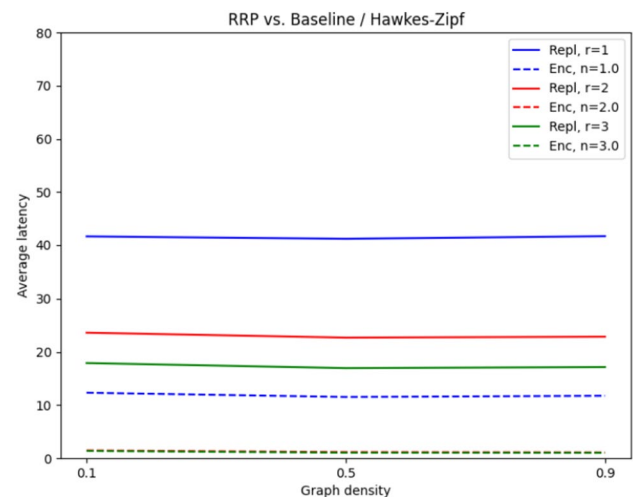


Fig. 6 RRP data retrieval strategy versus baseline strategy, *Hawkes-Zipf* mobility and request model

particular, the results for $n = 1$, $n = 2$, and $n = 3$ are shown in Figs. 7, 8, 9, 10 and 11, 12, respectively.

In each graph we report the value of the graph density Δ on the x-axis, and that of the Average Retrieval Time (see Sect. 6.1) on the y-axis. There are three groups of bars, each one corresponding to a specific value of Δ . In each of these groups there is a bar for each value of g (shown on top of the bar). Each bar reports the breakout of the Average Retrieval Time in the Average Queuing Time and the Average Network Time (see Sect. 6.1).

From these graphs, we can make the following considerations:

1. For a given n , the *Hawkes-Zipf* case consistently exhibits higher Average Retrieval Time than the *Hawkes-Exp* case. This is attributed to elevated Average Queuing Time values in the former. The skewed nature of the *Zipf* distribution concentrates a significant portion of requests on a few popular items. As a result, fewer edge nodes store coded fragments for these high-demand items, leading to congestion compared to the *Exp* distribution, especially for $n = 1$ and $g = 32$. However, as discussed below, as n increases and g decreases, the distribution of requests across more edge nodes mitigates this effect.
2. As n increases, the Average Retrieval Time decreases. A larger n implies a larger set of edge nodes storing fragments of a given data item d_i . This allows requests for the $[k_i/g]$ coded fragment groups to be distributed across more nodes, reducing queue lengths at individual edge nodes and, consequently, lowering the Average Queuing Time.
3. As g increases, the Average Retrieval Time also increases. Larger g values result in fewer coded

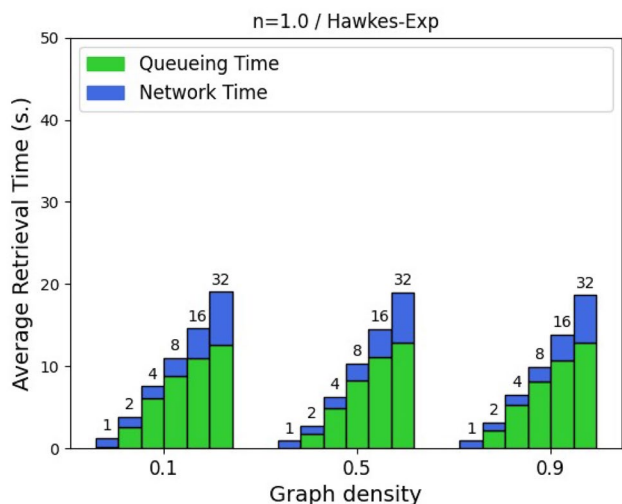


Fig. 7 RRP data retrieval strategy, $n = 1$, *Hawkes-Exp* mobility and request model

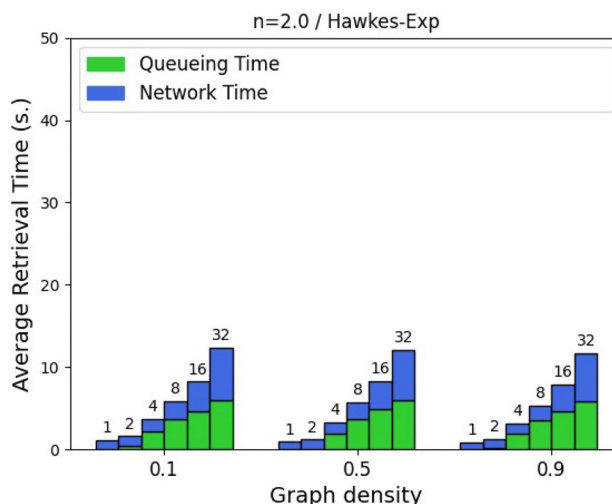


Fig. 9 RRP data retrieval strategy, $n = 2.0$, *Hawkes-Exp* mobility and request model

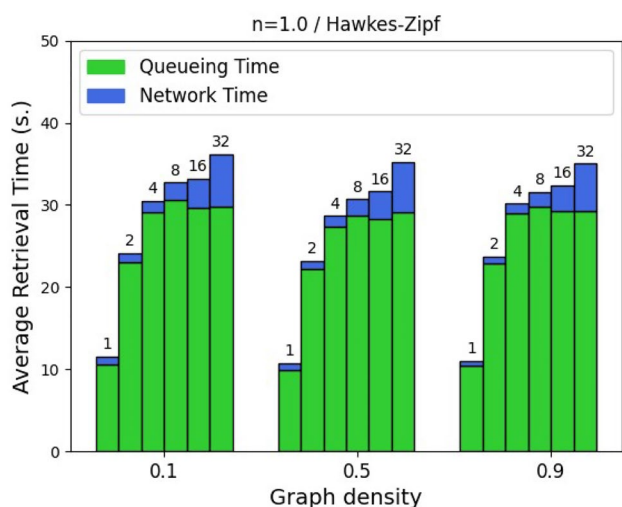


Fig. 8 RRP data retrieval strategy, $n = 1$, *Hawkes-Zipf* mobility and request model

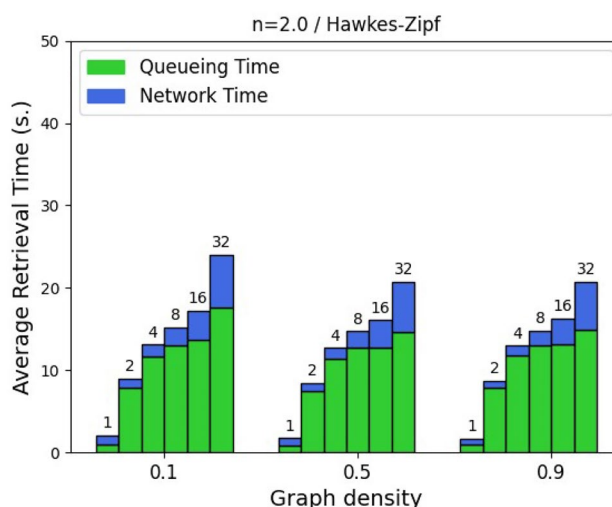


Fig. 10 RRP data retrieval strategy, $n = 2.0$, *Hawkes-Zipf* mobility and request model

fragment groups per data item d_i , limiting the number of edge servers that can store them. This restriction leads to increased queue lengths at edge nodes, consequently elevating the Average Queuing Time.

- The Average Retrieval Time is independent of Δ , as evidenced by the identical bar heights within each graph for a given g . This is because the Average Network Time, determined solely by link latency (as per [10, 16]), remains constant for a fixed g . However, the Average Retrieval Time is influenced by link bandwidth, which significantly impacts transfer time.

6.3.3 Results for the JSQ-RP data retrieval strategy

This section presents results for scenarios where JSQ-RP replaces RRP as the data retrieval strategy. As discussed in Sect. 6.3.2, results are independent of Δ , therefore, we only show results for $\Delta=0.5$ and omit those for $\Delta = [0.1, 0.9]$ to avoid redundancy.

Figures 13 and 14 depict results for the *Hawkes-Exp* and *Hawkes-Zipf* cases, respectively. The x-axis represents n , and the y-axis shows the Average Retrieval Time. For each n , the figures compare results for the RRP (label “Random”) and JSQ-RP (label “JSQ”) data retrieval strategies. Each strategy’s bar is further decomposed into Average Queuing Time and Average Network Time.

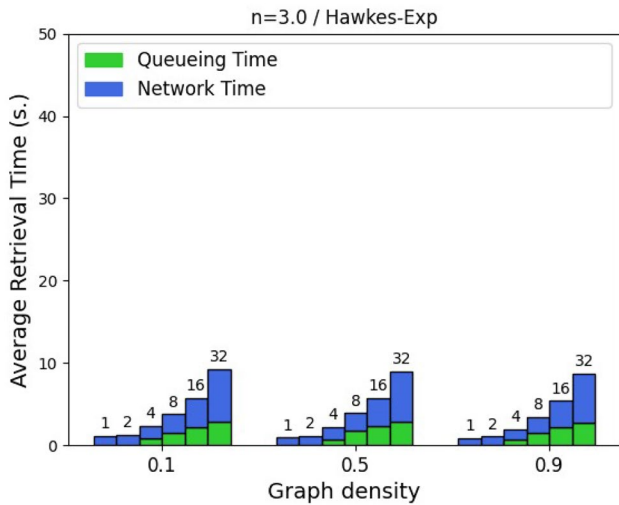


Fig. 11 RRP data retrieval strategy, $n = 3.0$, *Hawkes-Exp* mobility and request model

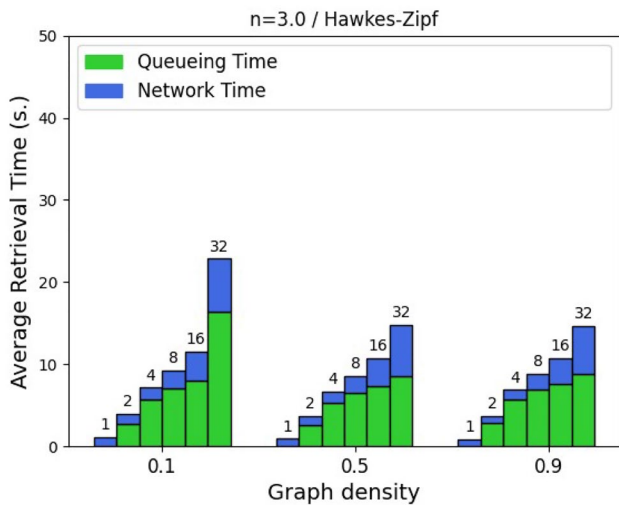


Fig. 12 RRP data retrieval strategy, $n = 3.0$, *Hawkes-Zipf* mobility and request model

As can be seen from this figure, *JSQ-RP* consistently reduces the Average Retrieval Time for all values of n and g by decreasing the Average Queuing Time. These figures highlight that the *JSQ-RP* strategy’s advantage over the RRP strategy is more pronounced for larger values of n . This is because larger n values allow for better distribution of requests for popular data items across more edge nodes.

The Zipf request model, with its highly skewed distribution, concentrates a significant portion of requests on a few popular items. Thus, larger n values lead to significant reductions in Average Queuing Time.

Conversely, an increase in g results in a decrease in the number of edge nodes storing coded fragment groups for each data item. This limitation on parallel requests to

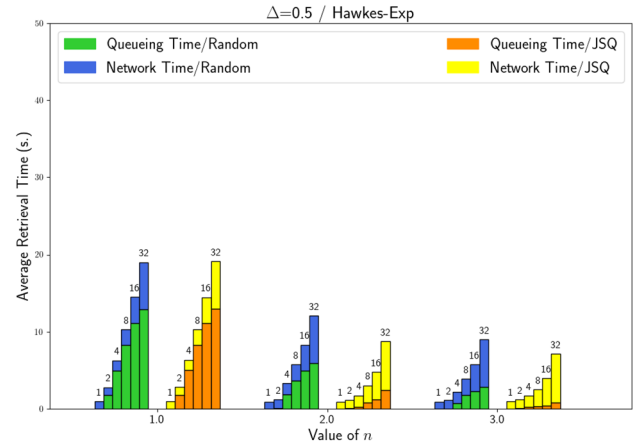


Fig. 13 RRP (label “Random”) vs *JSQ-RP* (label “*JSQ*”) data retrieval strategies, $\Delta = 0.5$, *Hawkes-Exp* mobility and request model

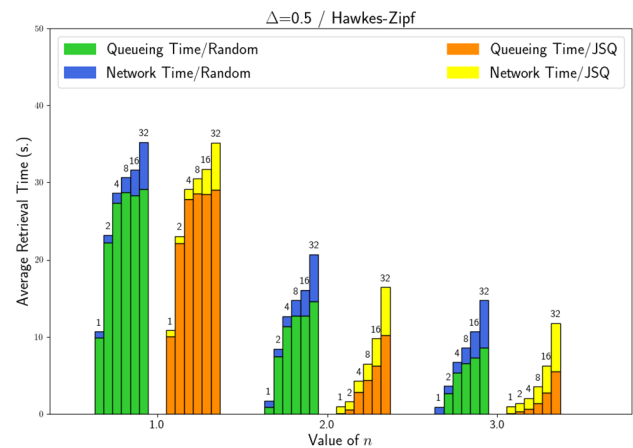


Fig. 14 RRP (label “Random”) vs *JSQ-RP* (label “*JSQ*”) data retrieval strategies, $\Delta = 0.5$, *Hawkes-Zipf* mobility and request model

distinct edge nodes consequently leads to increased queue lengths and a higher Average Queuing Time.

6.3.4 Discussion

The analysis of results in Sects. 6.3.2 and 6.3.3 highlights that minimizing the Average Queuing Time is crucial for achieving adequate performance. Two primary factors contribute to increased queuing time: (a) higher values of g and (b) skewed popularity distributions.

While the skewness of popularity distributions is inherent to the request process and beyond system design control, queuing time can be mitigated by increasing n . However, a large n can lead to excessive storage requirements, exceeding the capacity of available edge nodes. Thus, a careful balance between n and g is necessary for a given number of edge nodes N . While large n values reduce queuing, large g values exacerbate it. The *JSQ-RP* strategy offers a solution, significantly alleviating the queuing problem, especially

for larger g values, and facilitating a more favorable trade-off between n and g .

7 Related work

With the rapid and growing widespread of IoT devices at the edge of the network and the increasing demand for low-latency processing of data they generated, ESSs have emerged as an efficient and effective way to manage and access large amounts of data generated by such devices, thereby reducing the need for data transmission to centralized cloud infrastructures. However, a key challenge in ESSs is achieving the above goals and simultaneously ensuring data availability in face of the resource-constrained nature of edge devices (e.g., in terms of storage capacity) and the mobility of users. Several ESS approaches have been proposed to address the above challenge, mainly based on data replication or erasure coding techniques. Moreover, since data typically exhibits significant popularity skew (i.e., where a small fraction of data accounts for a large portion of the traffic), some of the existing approaches consider data popularity to deal with the effective data placement problem. In the rest of this section, we review some of them.

Data replication, whereby multiple copies of the same data are stored on different storage nodes, is one of the most widely-used techniques for addressing the above challenge [9, 10, 40–43]. For example, the authors of [40] propose D-ReP, a decentralized latency-aware and cost-efficient data placement strategy that dynamically manages data replicas by continuous monitoring data requests coming from edge nodes. In [9], authors propose DRIW_ITÖ, a data-replica placement stochastic optimization algorithm (similar to the particle swarm optimization) for the coordinated low-latency processing of data-intensive IoT workflows in collaborative edge and cloud computing environments. In [41], authors present MEAN, a deduplication-enabled ESS that places similar files together for better deduplication and maintain replicas of popular files for higher reliability, and propose an efficient heuristics based on similarity-aware hierarchical clustering. In [10], authors propose a data replica placement strategy in edge computing systems that considers the diverse popularity of data items, maps both data items and edge servers to a virtual plane, and places or retrieves data based on its virtual coordinate in the plane with the goal of reducing the pressure on overloaded edge servers and improving the overall performance. In [42], authors present a data placement strategy for Industrial IoT applications, based on data replication and taking into consideration data privacy. In [43], authors present BRPS, a data replica placement scheme based on the Deep Reinforcement Learning, designed to address the key challenges of data storage in the

cloud-edge environment (such as low latency, reliability and load balancing), while simultaneously taking into account the heterogeneity of device resources and the diverse data needs of user services. Although data replication ensures high data availability, it incurs significant storage overhead and network bandwidth as multiple copies of the same data are stored, which can be particularly challenging in resource-constrained environments typical of edge computing systems.

Therefore, encoding-based ESSs have been recently investigated as a more efficient approach than replication-based ESSs that cannot only provide fault-tolerance but also significantly reduce storage overhead [3, 4, 11]. Specifically, the work presented in [11] explores the use of erasure coding in edge computing systems to reduce storage capacity requirements, improve data transfer speed and provide fault tolerance. In [3], authors evaluate the performance of erasure coding on popular edge devices and coding libraries, and propose an acceleration method (consisting in parallelizing erasure coding on multi-core CPUs) to match the bandwidth of the underlying network. In [4], authors propose HCEFT, a secure and efficient hierarchical cloud-edge collaborative fault-tolerant storage architecture based on Software Defined Networks and erasure codes and its data writing optimization method. Although the above works provide a better understanding of erasure-coded storage systems for edge computing and can serve as references for future optimizations, the problem of effective data placement is completely neglected.

Also, the work in [5] proposes an encoding-based ESS where the data placement problem is modeled as an integer linear programming problem, which is solved either optimally for small-scale systems, or by using a specific approximate algorithm for large-scale systems. The main drawbacks of this approach are [20] that (1) the optimal algorithm suffers from poor stability and high computation costs, and (2) the work does not take into account the *popularity* of data, which instead is known to be a crucial factor in the design of data placement algorithms.

Finally, in [14], authors propose adaptive and scalable online caching schemes that are specifically designed for ESSs to achieve low latency and where the decisions of which data chunks to cache are taken in real time upon the arrival of requests and according on data popularity and network latency. This work is complementary to ours as it builds a caching layer on top of an encoding-based ESS (like the one we propose in this paper) to further lower data access latency especially in large geo-distributed ESS.

Overall, to the best of our knowledge, our work is the first data placement approach that combines data popularity and erasure-coding by pairing a polynomial-time virtual-space-based data placement strategy with a dynamic data

retrieval strategy that reduces data access times without overloading edge storage nodes. Moreover, our study differs from previous work by utilizing a request load model based on multivariate spatiotemporal Hawkes processes, enabling us to evaluate performance in more realistic scenarios with both spatial and temporal request correlations.

8 Conclusions and future developments

In this paper, we have presented a novel data placement approach for erasure-coded Edge Storage Systems (ESSs), which takes into account data popularity and system resource constraints. Our approach leverages a virtual-space-based placement technique, coupled with a Join-the-Shortest-Queue (JSQ) data retrieval strategy, to optimize data access and system performance.

Through extensive simulations, we have demonstrated that our proposed approach effectively mitigates queuing delays, especially in scenarios with high data popularity skew and high network load. By carefully balancing the number of coded fragments (i.e., n) and the number of groups of coded fragments (i.e., g), our approach enables efficient resource utilization and improved system performance. Hence, our proposed approach suitably addresses the data placement challenges arising from using erasure-coding, while allowing to exploit their advantages with respect to R-ESS in terms of both space efficiency and data availability

Our findings highlight the importance of considering data popularity and queueing dynamics in the design of ESSs. By adopting our proposed approach, edge systems can provide low-latency, high-throughput, and resilient data services, even in challenging network environments.

Future research directions include exploring more sophisticated data placement and data retrieval strategies that consider dynamic network conditions such as node failures and network fluctuations, user mobility patterns, and emerging storage technologies. In particular, we plan to explore data placement methods based on advanced techniques, such as deep learning or heuristic-based optimization. Additionally, investigating the impact of different erasure-coding schemes and fault-tolerance mechanisms on the performance of ESSs is an important area for further exploration. Moreover, for the sake of reproducibility, we plan to provide the source code of our approach after a complete code refactoring and polishing as well as fully functional testing coverage. Furthermore, we plan to extend the performance analysis by considering other performance indicators, and in particular network bandwidth occupation and storage space utilization, in addition to retrieval time. Finally, we plan to explore the suitability of our approach to various applicative

domains, such as data distribution optimization for distributed deep learning tasks, or edge storage deployment in various IoT-based scenarios.

Acknowledgements This research was partly supported by the INdAM-GNCS Project 2024.

Author Contributions All authors contributed equally to the manuscript.

Funding Open access funding provided by Università degli Studi del Piemonte Orientale Amedeo Avogadro within the CRUI-CARE Agreement. This research was partly supported by the INdAM-GNCS Project 2024.

Data Availability No datasets were generated or analysed during the current study.

Declarations

Ethical Approval Not applicable.

Competing interests The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. *IEEE Internet Things J.* **3**(5), 637–646 (2016). <https://doi.org/10.1109/JIOT.2016.2579198>
2. Kong, L., Tan, J., Huang, J., Chen, G., Wang, S., Jin, X., Zeng, P., Khan, M., Das, S.K.: Edge-computing-driven internet of things: A survey. *ACM Comput. Surv.* (2022). <https://doi.org/10.1145/3555308>
3. Liang, L., He, H., Zhao, J., Liu, C., Luo, Q., Chu, X.: An erasure-coded storage system for edge computing. *IEEE Access* **8**, 96271–96283 (2020). <https://doi.org/10.1109/ACCESS.2020.2995973>
4. Chen, J., Wang, Y., Ye, M., Jiang, Q.: A secure cloud-edge collaborative fault-tolerant storage scheme and its data writing optimization. *IEEE Access* (2023). <https://doi.org/10.1109/access.2023.3291452>
5. Jin, H., Luo, R., He, Q., Wu, S., Zeng, Z., Xia, X.: Cost-effective data placement in edge storage systems with erasure code. *IEEE Trans. Serv. Comput.* **16**, 1039–1050 (2023). <https://doi.org/10.1109/tsc.2022.3152849>

6. Yadgar, G., Kolosov, O.: Data storage at the edge: Challenges and opportunities. *IEEE BITS the Information Theory Magazine*, 1–13 (2024) <https://doi.org/10.1109/MBITS.2024.3362289>
7. Makris, A., Kontopoulos, I., Psomakelis, E., Xyalis, S.N., Theodoropoulos, T., Tserpes, K.: Performance analysis of storage systems in edge computing infrastructures. *Applied Sciences* **12**(17), 8923 (2022). <https://doi.org/10.3390/app12178923>
8. Luo, R., Jin, H., He, Q., Wu, S., Xia, X.: Cost-effective edge server network design in mobile edge computing environment. *IEEE Transactions on Sustainable Computing* **7**(4), 839–850 (2022). <https://doi.org/10.1109/tsusc.2022.3178661>
9. Shao, Y., Li, C., Tang, H.: A data replica placement strategy for iot workflows in collaborative edge and cloud environments. *Comput. Netw.* **148**, 46–59 (2019). <https://doi.org/10.1016/j.comnet.2018.10.017>
10. Wei, X., Wang, Y.: Popularity-based data placement with load balancing in edge computing. *IEEE Transactions on Cloud Computing* **11**(1), 397–411 (2023). <https://doi.org/10.1109/TCC.2021.3096467>
11. Dussoye, S., Issack, Z., Chiniah, A.: Erasure code and edge computing for providing an optimal platform for storage of iot data. In: 2019 IEEE Global Conference on Internet of Things (GCIoT), pp. 1–4 (2019). <https://doi.org/10.1109/gciot47977.2019.9058414>
12. Ruty, G., Baccouch, H., Nguyen, V., Surcouf, A., Rougier, J.-L., Boukhatem, N.: Popularity-based full replica caching for erasure-coded distributed storage systems. *Clust. Comput.* **24**(4), 3173–3186 (2021). <https://doi.org/10.1007/s10586-021-03317-0>
13. Kontodimas, K., Soumplis, P., Kretsis, A., Kokkinos, P., Fehér, M., Lucani, D.E., Varvarigos, E.: Secure distributed storage orchestration on heterogeneous cloud-edge infrastructures. *IEEE Transactions on Cloud Computing* **11**(4), 3407–3425 (2023). <https://doi.org/10.1109/tcc.2023.3287653>
14. Liu, K., Peng, J., Wang, J., Huang, Z., Pan, J.: Adaptive and scalable caching with erasure codes in distributed cloud-edge storage systems. *IEEE Transactions on Cloud Computing* **11**(2), 1840–1853 (2023). <https://doi.org/10.1109/TCC.2022.3168662>
15. Li, J., Li, B.: Erasure coding for cloud storage systems: A survey. *Tsinghua Science and Technology* **18**(3), 259–272 (2013)
16. Xie, J., Qian, C., Guo, D., Li, X., Shi, S., Chen, H.: Efficient data placement and retrieval services in edge computing. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 1029–1039 (2019). <https://doi.org/10.1109/ICDCS.2019.00106>
17. Lin, B., Zhu, F., Zhang, J., Chen, J., Chen, X., Xiong, N.N., Mauri, J.L.: A time-driven data placement strategy for a scientific workflow combining edge computing and cloud computing. *IEEE Trans. Industr. Inf.* **15**(7), 4254–4265 (2019). <https://doi.org/10.1109/tii.2019.2905659>
18. Li, C., Bai, J., Tang, J.: Joint optimization of data placement and scheduling for improving user experience in edge computing. *Journal of Parallel and Distributed Computing* **125**, 93–105 (2019). <https://doi.org/10.1016/j.jpdc.2018.11.006>
19. Breitbach, M., Schäfer, D., Edinger, J., Becker, C.: Context-aware data and task placement in edge computing environments. In: 2019 IEEE International Conference on Pervasive Computing and Communications (PerCom), pp. 1–10 (2019). <https://doi.org/10.1109/percom.2019.8767386>
20. Hamdeni, C., Hamrouni, T., Charrada, F.B.: Data popularity measurements in distributed systems: Survey and design directions. *Journal of Network and Computer Applications* **72**, 150–161 (2016)
21. Naboulsi, D., Fiore, M., Ribot, S., Stanica, R.: Large-scale mobile traffic analysis: A survey. *IEEE Communications Surveys & Tutorials* **18**(1), 124–161 (2016)
22. Hawkes, A.G.: Spectra of some self-exciting and mutually exciting point processes. *Biometrika* **58**(1), 83–90 (1971)
23. Yuan, B., Li, H., Bertozzi, A.L., Brantingham, P.J., Porter, M.A.: Multivariate spatiotemporal Hawkes processes and network reconstruction. *SIAM Journal on Mathematics of Data Science* **1**(2), 356–382 (2019)
24. Winston, W.: Optimality of the shortest line discipline. *J. Appl. Probab.* **14**(1), 181–189 (1977)
25. Ravindran, A., George, A.: An edge datastore architecture for Latency-Critical distributed machine vision applications. In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. USENIX Association, Boston, MA (2018). <https://www.usenix.org/conference/hotedge18/presentation/ravindran>
26. Sonbol, K., Özkasap, Al-Oqily, I., Aloqaily, M.: Edgekv: Decentralized, scalable, and consistent storage for the edge. *Journal of Parallel and Distributed Computing* **144**, 28–40 (2020). <https://doi.org/10.1016/j.jpdc.2020.05.009>
27. Rashid, M.T., Zhang, D., Wang, D.: Edgestore: Towards an edge-based distributed storage system for emergency response. In: 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 2543–2550 (2019). <https://doi.org/10.1109/HPCC/SmartCity/DSS.2019.00356>
28. Saurabh, N., Kimovski, D., Gaetano, F., Prodan, R.: A two-stage multi-objective optimization of erasure coding in overlay networks. In: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID), pp. 150–159 (2017). <https://doi.org/10.1109/CCGRID.2017.79>
29. Lujic, I., De Maio, V., Brandic, I.: Efficient edge storage management based on near real-time forecasts. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC), pp. 21–30 (2017). <https://doi.org/10.1109/ICFEC.2017.9>
30. Bedru, H.D., Yu, S., Xiao, X., Zhang, D., Wan, L., Guo, H., Xia, F.: Big networks: A survey. *Computer Science Review* (2020). <https://doi.org/10.1016/j.cosrev.2020.100247>
31. Xie, J., Qian, C., Guo, D., Wang, M., Shi, S., Chen, H.: Efficient indexing mechanism for unstructured data sharing systems in edge computing. In: *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, pp. 820–828 (2019). <https://doi.org/10.1109/INFOCOM.2019.8737617>
32. Axboe, J.: fio – The Flexible I/O Tester. Last accessed: Dec 12, 2024. <https://git.kernel.dk/cgit/fio/>
33. Intel: Intel Solid State Drive DC S3610 Series Product Specification. Last accessed: Dec 12, 2024. <https://www.intel.com/content/dam/www/public/us/en/documents/product-specifications/ssd-dc-s3610-spec.pdf>
34. Kenneth P. Burnham, D.R.A.: *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach*, 2nd edn. Springer, New York, NY (2002). <https://doi.org/10.1007/b97636>
35. Bollobás, B.: *Random Graphs*, 2nd edn. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge (2001)
36. Zare, H., Cadambe, V.R., Urgaonkar, B., Alfares, N., Soni, P., Sharma, C., Merchant, A.A.: Legostore: a linearizable geo-distributed store combining replication and erasure coding. *Proc. VLDB Endow.* (2022). <https://doi.org/10.14778/3547305.3547323>
37. Caiazza, C., Bernardi, L., Bevilacqua, M., Cabras, A., Cicconetti, C., Luconi, V., Sciurti, G., Senore, E., Vallati, E., Vecchio, A.: Mecperf: An application-level tool for estimating the network performance in edge computing environments. In: *IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)* (2020)

38. Chen, B., Yang, C.: Caching policy for cache-enabled d2d communications by learning user preference. *IEEE Transactions on Communications*, 6601 (2018). <https://doi.org/10.1109/TCOMM.2018.2863364>
39. Zhao, X., Zhu, Q.: Mobility-aware and interest-predicted caching strategy based on iot data freshness in d2d networks. *IEEE Internet of Things Journal* (2021). <https://doi.org/10.1109/JIOT.2020.3033552>
40. Aral, A., Ovatman, T.: A decentralized replica placement algorithm for edge computing. *IEEE Trans. Netw. Serv. Manage.* **15**(2), 516–529 (2018). <https://doi.org/10.1109/TNSM.2017.2788945>
41. Xia, J., Cheng, G., Luo, L., Guo, D., Lv, P., Sun, B.: The doctrine of mean: Realizing deduplication storage at unreliable edge. *IEEE Trans. Parallel Distrib. Syst.* **34**(10), 2811–2826 (2023). <https://doi.org/10.1109/TPDS.2023.3305460>
42. Shen, Z., Liu, B., Dou, W.: An effective data placement strategy for iiot applications. *Concurrency and Computation: Practice and Experience* **36**(10), 7977 (2024). <https://doi.org/10.1002/cpe.7977>
43. Zheng, M., Du, X., Lu, Z., Duan, Q.: A balanced and reliable data replica placement scheme based on reinforcement learning in edge-cloud environments. *Futur. Gener. Comput. Syst.* **155**, 132–145 (2024). <https://doi.org/10.1016/j.future.2024.02.004>

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