Secure Distributed Storage on Cloud-Edge Infrastructures

Konstantinos Kontodimas*‡, Polyzois Soumplis*‡, Aristotelis Kretsis*‡, Panagiotis Kokkinos†‡ and Emmanouel Varvarigos*‡
*School of Electrical and Computer Engineering, National Technical University of Athens
†Department of Digital Systems, University of Peloponnese
‡Institute of Communication and Computer Systems, Athens, Greece
e-mail: {kontodimas, soumplis, akretsis, kokkinop, vmanos}@mail.ntua.gr

2021 IEEE 10th International Conference on Cloud Networking (CloudNet) | 978-1-6654-3538-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/CloudNet53349.2021.9657156

***Abstract*—Distributed storage systems place data in multiple cloud datacenters, leading to increased availability and flexibility. However, limitations are present when strict bandwidth and latency requirements are posed by the applications, as the data are stored into different, probably distant locations. The incorporation of edge resources in distributed storage services enables the placement of the data closer to their source, serving better the applications' demands. Erasure coding offers a way to increase the availability and longevity of data during hosting. In our work, we develop mechanisms that perform resource allocation and store data at edge and cloud resources taking advantage of their different characteristics, while also exploiting the erasure coding technique. Initially, we provide a mixed integer linear programming formulation of the considered problem. As the search space can be vast and the execution time prohibitively large for real size problems, we also propose a heuristic approach which makes use of the rollout policy to efficiently trade-off performance with execution time. A set of simulation experiments is performed to showcase the validity of the proposed methods.**

***Index Terms*—distributed storage, cloud, edge, erasure coding, resource allocation.**

I. Introduction

Digital transformation has a significant impact on the storage capacity requirements. According to IDC [1], a tenfold increase in the data generated worldwide is expected between 2017 and 2025. As cloud computing is becoming the cornerstone of our digital society, businesses prefer to store their data in the cloud rather than deploying their own privately held infrastructure, taking advantage of the offered scalability and increased availability. In this manner, businesses are alleviated from the burden of deploying complex and costly data redundancy and fault-tolerance mechanisms.

Distributed storage systems store data in multiple cloud datacenters and consolidate resources from multiple cloud providers, leading to increased flexibility as opposed to single storage services [2]. However, such storage systems struggle to address the heterogeneous and strict requirements of bandwidth-intensive and latency-sensitive applications, as the data are stored into different and probably distant locations. Edge computing enables the storage and the processing of the data close to the generating source (e.g. camera, or other sensor) [3]. The incorporation of edge resources in distributed storage services improves the way the demanding applications are served: data are stored and processed at the edge in order to minimize latency and network usage and if additional resources are required, then cloud is utilized. Overall, the increase in the number and density of edge resources is expected to change the way current storage services operate.

An important aspect in distributed storage systems is the maintance of data integrity against losses, ensuring privacy and security. Erasure coding [4] is the foremost technique that makes use of forward error correction codes and ensures the availability and longevity of data. Data are split into fragments and are encoded with an overhead that depends on the encoding algorithm. Whenever a part of the data fragments cannot be retrieved, the original data can still be recovered by retrieving only a subset of minimum size, which is defined by the encoding algorithm. After the successful retrieval of the encrypted fragments, the decryption operation is performed.

It is evident that the operation of a distributed storage service that jointly utilizes edge and cloud resources and employs erasure coding when splitting the data, sets a challenging task for the respective resource orchestration mechanism. In our work, we develop and evaluate such mechanisms. Initially, we provide a mixed linear programming formulation of the considered problem. As the search space can be vast and the execution time prohibitively large, we also propose a rollout heuristic approach, which efficiently trades-off performance with execution time. The developed mechanisms take under consideration the edge and cloud storage resource characteristics, i.e. cost, capacity, reliability, performance, availability, and jointly decide on the data fragmentation, encryption and selection of the set of hosting storage locations. The selection is made in a way that fulfills the applications' demands, optimizing also critical factors like latency, data availability and security. A set of simulation experiments are performed showcasing that the proposed methods can efficiently serve the performance and security requirements of the offered services.

II. Related Work

The storage resource allocation problem over a distributed infrastructure has attracted the research interest for many years. To address the difficulties of single cloud models, multi-cloud resource allocation schemes have been examined [2], [5]-[6]. In [2], Hadji proposed a solution based on commodity flows to minimize the storage cost and latency. Papaioannou

et al. [5] proposed Scalia, a cloud storage brokerage solution for data placement that targets to minimize the storage cost. Mansouri et al. [7] proposed an algorithm that minimizes the storage cost, guaranteeing also the expected data availability. The proposed mechanisms target the minimization of the storage cost while guaranteeing high data availability and privacy. Ma et al. [8] have proposed a mixed policy which is based on a combination of erasure and replication coding to minimize the latency, the storage and the network cost. In the same context Zhang et al. [9] proposed a sub-optimal heuristic approach for selecting the storage locations and the appropriate redundancy configuration, guaranteeing the availability of stored files. To minimize the cost while meeting the user's latency and availability requirements, Wu et al. [10] proposed a scheme that trades-off cost for latency. Liu et al. [6] proposed a heuristic (genetic) algorithm to minimize costs while providing Service Level guarantees.

Other works have proposed mechanisms that improve data availability through redundancy, minimizing also the required monetary cost. However, these works rely on replication coding, which requires more storage space, compared to erasure coding. In [5], [9] and [11], [12], the authors proposed mechanisms that make use of erasure coding solutions for improved data availability. In this direction, Wang et al. [13], [14] proposed various techniques that minimize the monetary cost while maximizing the availability. Wang et al. [15] proposed an adaptive model for data placement to minimize the monetary cost considering latency and availability constraints.

The above studies consider only cloud locations for data placement and the studied objectives such as data availability maximization, latency or cost minimization are examined separately. In our work, in addition to the cloud resources, we also consider the availability of edge devices, which are characterized by a dynamic nature compared to cloud resources and are also placed closer to the data generation points. These resources are utilized in a co-operative manner leveraging erasure coding to adapt to the user requirements and address simultaneously the different optimization criteria.

III. Distributed Storage Operations

The operations that are performed to store a file in a distributed storage infrastructure include: (i) Data processing operations, (ii) Store operations and (iii) Retrieve operations. Each one, relates to a file d and introduces a monetary cost and a latency. Both depend on the file's (i) size M_d (measured in Data Units - DU), (ii) hosting duration T_d (measured in Period Units - PU) and (iii) number of expected retrievals τ_d within the hosting duration. The number of fragments k each file d is split into and the number of fragments m used for redundancy depends on the erasure coding characteristics and the considered optimization criteria.

A. Store/Retrieve Data Processing Operation

When a file d needs to be hosted, it is initially transferred to an on-premise component (OPC) ω_d via a secure connection. At the OPC, the file is split into k fragments of size M_d/k. These fragments are encoded given a $(k+m,k)$ erasure coding scheme and a total of $k+m$ fragments are created, encrypted and stored in the infrastructure. The delay of splitting is denoted as $D_{\mathrm{spl}\omega_d}$ (in TUs/split), while the delay of encoding/encryption is $D_{\mathrm{enc}\omega_d}$ (in TUs) per DU. Hence, the overall latency for the store operation is:

$$g_{kmd} = \frac{k+m}{k} M_d D_{\mathrm{enc}\omega_d} + k D_{\mathrm{spl}\omega_d} \tag{1}$$

To retriecve a file, the decryption and decoding of the fragments is performed at the OPC ω_d, which performs the linear combination of a subset of k fragments among all the $k+m$ fragments. The decoded fragments are then merged into file d. The latency of decoding/decryption, denoted by $D_{\mathrm{dec}\omega_d}$ (in TUs) per DU, is proportional to M_d, while the latency of merging, denoted by $D_{\mathrm{mer}\omega_d}$ (in TUs/merge) per DU, is proportional to k. Therefore, the overall latency of the retrieve operation is derived as:

$$g'_{kmd} = M_d D_{\mathrm{dec}\omega_d} + k D_{\mathrm{mer}\omega_d} \tag{2}$$

As a different number of fragments and erasure codes of different capabilities can be selected, an interesting trade-off exists between the number of fragments and the associated overhead, which can be exploited by the storage allocation mechanism to further improve the performance of the infrastructure.

B. Store Operation

1) Store Operation Monetary Cost: Data fragments are stored in the available edge/cloud infrastructure at various locations. We denote by $\mathcal{N}_{\mathrm{s}}$ the selected locations that host these fragments, with $|\mathcal{N}_{\mathrm{s}}| = k+m$. Since each storage location $n \in \mathcal{N}_{\mathrm{s}}$ charges P_{sn} CUs for each DU, the monetary cost for storing the fragments of file d can be calculated as follows:

$$\phi_1(k,m,d) = \sum_{n \in \mathcal{N}_{\mathrm{s}}} \frac{M_d}{k} T_d P_{sn} \tag{3}$$

2) Store Operation Latency: The latency of the store operation depends on the: (i) processing latency, (ii) latency of placing the fragments into the storage locations, (iii) propagation latency, and (iv) transmission latency.

To process each fragment g_{kmd} TUs are required, while the latency for placing the fragments into the storage locations requires D_{sn} TUs per DU. A transmission latency of $D_{\mathrm{bn}\omega_d}$ per DU and a propagation delay of $D_{\mathrm{ln}\omega_d}$ are also introduced, which depend on the network's link capacity and the distance between the OPC ω_d and the storage location respectively. Given the locations $\mathcal{N}_{\mathrm{s}}$ where data fragments are stored, the total latency for storing a file d is calculated as:

$$\phi_2(k,m,d) = g_{kmd} + \sum_{n \in \mathcal{N}_{\mathrm{s}}} \frac{M_d}{k} D_{\mathrm{bn}\omega_d} + \max_{n \in \mathcal{N}_{\mathrm{s}}} \left\{ \frac{M_d}{k} D_{sn} + D_{\mathrm{ln}\omega_d} \right\} \tag{4}$$

C. Retrieve Operation

1) Retrieve Operation Monetary Cost: The retrieval cost is related to the subset $\mathcal{N}_\mathrm{r} \subseteq \mathcal{N}_\mathrm{s}$ ($|\mathcal{N}_\mathrm{r}| = k$) of nodes from which the fragments are retrieved. Given that each storage location $n \in \mathcal{N}_\mathrm{r}$ charges according to the number of GET requests that are required to retrieve the whole fragment, each GET request retrieves ρ DUs and costs P_{rn} CUs. Therefore, the monetary cost for the retrieve operation is calculated as:

$$\phi_3(k,m,d) = \sum_{n \in \mathcal{N}_r} \frac{1}{\rho} \frac{M_d}{k} P_{rn} \tag{5}$$

2) Retrieve Operation Latency: Similarly to the store operation latency, the retrieve operation introduces latency that consists of the: (i) processing latency, (ii) latency for recovering the fragments from the storage locations, (iii) propagation latency, and (iv) transmission latency. Particularly, the latency for recovering the k fragments from the storage locations requires $D_{\mathrm{r}n}$ TUs per DU. Next, the transmission latency $D'_{\mathrm{b}n\omega_d}$ per DU and the propagation delay $D_{\mathrm{l}n\omega_d}$ depend on the network's characteristics and the distance between the storage locations and the OPC ω_d. Finally, a processing latency of g'_{kmd} (in TUs) is required at the OPC ω_d for decrypting and decoding the k fragments and then merging them into file d. Assuming that the storage allocation mechanism selects the locations of $\mathcal{N}_\mathrm{r}$ for retrieving the datasets, then the latency for the retrieve operation is calculated as:

$$\phi_4(k,m,d) = g'_{kmd} + \sum_{n \in \mathcal{N}_\mathrm{r}} \frac{M_d}{k} D'_{\mathrm{b}n\omega_d} + \max_{n \in \mathcal{N}_\mathrm{r}} \left\{ \frac{M_d}{k} D_{\mathrm{r}n} + D_{\mathrm{l}n\omega_d} \right\} \tag{6}$$

IV. Distributed Storage Resource Allocation

Given a distributed storage infrastructure, the storage resource allocation problem decides the way a set of files $\mathcal{D}$ is served. Each file $d \in \mathcal{D}$ is described by the tuple $(M_d, T_d, \tau_d, \omega_d, \mathcal{Q}_d, \mathcal{K}_d, \mathcal{M}_d)$, where T_d is the hosting duration (in PUs), τ_d is the number of future retrievals (offline problem), ω_d is the OPC where d is going to be processed at and $\mathcal{Q}_d$ is the set of QoS requirements. Each file is initially processed at ω_d and it is split into fragments whose number lies in the set $\mathcal{K}_d$. Next, a set of different erasure codes is used to encode the fragments. The choices of the number of fragments used for redundancy is denoted as $\mathcal{M}_d$. A total of $k+m$ ($k \in \mathcal{K}_d, m \in \mathcal{M}_d$) fragments is produced. The resource allocation mechanism has to decide the (i) number of fragments that each file is split into, (ii) the appropriate erasure code and (iii) the set of storage locations where the fragments are hosted on, so as to minimize the weighted cost that results from the different optimization criteria.

A. Pre-processing Phase - Availability

In order to consider the availability of the file d, we make use of a pre-processing phase in which we calculate the combinations of $k+m$ storage nodes that can be used to host the fragments, $\forall k \in \mathcal{K}_d$ and $\forall m \in \mathcal{M}_d$. All fragments are stored over the infrastructure under the assumption that each storage node hosts at most one fragment of each file. Each node $n \in \mathcal{N}$ has an availability of A_n. There are $\Phi_{km} = \binom{|\mathcal{N}|}{k+m}$ combinations for choosing $k+m$ nodes, denoted as $\mathcal{J}_{1km}, \mathcal{J}_{2km}, \ldots, \mathcal{J}_{\Phi_{km}}$. Since $0 \sim m$ node failures can be tolerated, the availability is calculated by summing the probabilities of $\kappa \in [k, k+m]$ nodes being available. We denote the number of collections of κ available nodes as $\Theta = \binom{|\mathcal{J}_{ikm}|}{\kappa}$ and the j^{th} collection as S_j^Θ. The availability of a file when hosted by the storage nodes of $\mathcal{J}_{ikm}$ is derived as:

$$a_{ikm} = \sum_{\kappa=k}^{k+m} \sum_{j=1}^{\Theta} \Big[\prod_{n \in S_j^\Theta} A_n \prod_{n \in \mathcal{J}_{ikm} \setminus S_j^\Theta} (1 - A_n) \Big] \tag{7}$$

where $\mathcal{J}_{ikm} \setminus S_j^\Theta$ denotes the unavailable nodes. Next, we prune the combinations that do not meet the minimum availability requirement $A_\mathrm{req} \in \mathcal{Q}_d$. We denote the indices of the combinations that meet the requirement as $\mathcal{I}_{km} = \{i \,|\, a_{ikm} \geq A_\mathrm{req} \in \mathcal{Q}_d \text{ and } i = 1, 2, \ldots, \Phi_{km}\}$, $\forall k \in \mathcal{K}_d$ and $\forall m \in \mathcal{M}_d$. Hence, the availability objective is:

$$\phi_5(k,m,d) = \max_{i \in \mathcal{I}_{km}} \{a_{ikm} \,|\, k \in \mathcal{K}_d \wedge m \in \mathcal{M}_d\} \tag{8}$$

TABLE I: Definition of MILP variables.

Var.	Definition
v_{nd}	Binary variable that indicates whether a fragment of file $d \in \mathcal{D}$ is placed at location $n \in \mathcal{N}$.
y_{kmd}	Binary variable that indicates the splitting configuration $k \in \mathcal{K}_d$ and the erasure code $(k+m, m)$, $m \in \mathcal{M}_d$.
x_{nkmd}	Binary variable that indicates whether a fragment of file $d \in \mathcal{D}$ is placed at location $n \in \mathcal{N}$, the splitting configuration $k \in \mathcal{K}_d$ and the erasure code $(k+m, k)$, $m \in \mathcal{M}_d$, which is used for file d.
z_{nkmdt}	Binary variable that indicates whether the fragment of file $d \in \mathcal{D}$ which is placed at location $n \in \mathcal{D}$, is retrieved during the t^{th} ($t = 1, 2, \ldots, t_d$) retrieve operation, while the splitting configuration is $k \in \mathcal{K}_d$ and the erasure code is $(k+m, m)$, $m \in \mathcal{M}_d$.
ξ_{nd}	Binary variable that indicates the fragment and the storage node $n \in \mathcal{N}$ with the maximum propagation and placing delay in store operation, for file $d \in \mathcal{D}$.
ξ'_{ndt}	Binary variable that indicates the fragment and the storage node $n \in \mathcal{N}$ with the maximum propagation and recovery delay, in the t^{th} ($t = 1, 2, \ldots, t_d$) retrieve operation, for file $d \in \mathcal{D}$.
ψ_d	Integer variable, which denotes the maximum value of propagation and placing delay in store operation, for each file $d \in \mathcal{D}$.
ψ'_{dt}	Integer variable, which denotes the maximum value of propagation and recovery delay, in the t^{th} ($t = 1, 2, \ldots, t_d$) retrieve operation, for each file d.
ζ_{ikmd}	Binary variable that indicates the combination $i \in \mathcal{I}_{km}$ of storage locations, the splitting configuration $k \in \mathcal{K}_d$ and the erasure code $(k+m, k)$, $m \in \mathcal{M}_d$, used for each $d \in \mathcal{D}$.

B. MILP Formulation

In this section we present the MILP formulation of the distributed storage resource allocation mechanism.

The examined objective includes the monetary cost for the store and retrieve operations, the latency and the average availability of the file. Constraints C1-C3 are used to calculate the latency when the fragments are stored at the different storage locations, while constraints C4-C6 are used for the calculation of the retrieval latency. Constraint C7 ensures that

the placement of the fragments has been performed based on the available storage capacity and constraint C8 ensures that all fragments are hosted by the infrastructure. Constraint C9 ensures that a selection of the number of fragments and the erasure code is performed. Constraints C10-C14 express that the minimum number of fragments is retrieved along with the respective storage locations. Finally, constraints C15-C16 spot the locations where the fragments for a specific file are placed and C17 is a weighting co-efficient related constraint.

min $[w_1\bar{\phi}_1, w_2\bar{\phi}_2, w_3\bar{\phi}_3, w_4\bar{\phi}_4, w_5\bar{\phi}_5]$

subject to:

C1 $\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_{\text{sn}}x_{nkmd}+ \sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{\text{ln}\omega_d}x_{nkmd}-\psi_d\leq 0, \quad n\in\mathcal{N}, d\in\mathcal{D}$

C2 $-\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_{\text{sn}}x_{nkmd}-\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{\text{ln}\omega_d}x_{nkmd}+ U_d\xi_{nd}+\psi_d\leq U_d, \quad n\in\mathcal{N}, d\in\mathcal{D}$

C3 $\sum_{n\in\mathcal{N}}\xi_{nd}=1, \quad d\in\mathcal{D}$

C4 $\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_{\text{rn}}z_{nkmdt}+\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{\text{ln}\omega_d}z_{nkmdt}- \psi'_{dt}\leq 0, \quad n\in\mathcal{N}, d\in\mathcal{D}, t=1,\ldots,t_d$

C5 $-\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}D_{\text{rn}}z_{nkmdt}-\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}D_{\text{ln}\omega_d}z_{nkmdt}+ U'_d\xi'_{ndt}+\psi'_{dt}\leq U'_d, \quad n\in\mathcal{N}, d\in\mathcal{D}, t=1,\ldots,t_d$

C6 $\sum_{n\in\mathcal{N}}\xi'_{ndt}=1, \quad d\in\mathcal{D}, t=1,\ldots,\tau_d$

C7 $\sum_{d\in\mathcal{D}}\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}\frac{M_d}{k}x_{nkmd}\leq C_n, \quad n\in\mathcal{N}$

C8 $\sum_{n\in\mathcal{N}}v_{nd}=\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}(k+m)y_{kmd}, \quad d\in\mathcal{D}$

C9 $\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}y_{kmd}=1, \quad d\in\mathcal{D}$

C10 $\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}z_{nkmdt}=\sum_{k\in\mathcal{K}_d}\sum_{m\in\mathcal{M}_d}ky_{kmd}, \quad d\in\mathcal{D}, t=1,\ldots,t_d$

C11 $z_{nkmdt}\leq x_{nkmd}, n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d, t=1,\ldots,t_d$

C12 $-x_{nkmd}+v_{nd}+y_{kmd}\leq 1, \quad n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$

C13 $x_{nkmd}-v_{nd}\leq 0, \quad n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$

C14 $x_{nkmd}-y_{kmd}\leq 0, \quad n\in\mathcal{N}, d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$

C15 $-\sum_{n\in\mathcal{N}}\theta(i,k,m,n)x_{nkmd}+U''_{ikm}\zeta_{ikmd}\leq U''_{ikm}- \sum_{n\in\mathcal{N}}\theta(i,k,m,n), \quad d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d, i\in\mathcal{I}_{km}$

C16 $\sum_{i\in\mathcal{I}_{km}}\zeta_{ikmd}=y_{kmd}, \quad d\in\mathcal{D}, k\in\mathcal{K}_d, m\in\mathcal{M}_d$

C17 $\sum_{i=1}^{5}w_i=1.0$

C. Heuristic Algorithm

For large instances, the optimal MILP mechanism requires a high execution time. For this reason we also developed a heuristic algorithm which relies on a sub-optimal best policy. It serves the demands sequentially by creating a partial trajectory of the different file fragmentation options. Then it executes a rollout based algorithm [16] that operates as if each partial trajectory was the only one and selects for each file the resource allocation with the minimum cost. The pseudocode of the proposed mechanism is presented in Algorithm 1.

Algorithm 1 Rollout heuristic algorithm

Input: w, $\mathcal{N}$, $\mathcal{D}$, network data
Output: $nodes(d), erasure\_code(d), \forall d\in\mathcal{D}$ and $tcost$
for $d\in\mathcal{D}$ **do**
  $CM_d\leftarrow\{\}$
  for $k\in\mathcal{K}_d, m\in\mathcal{M}_d$ **do**
    $CS\leftarrow\{\}$
    for $n\in\mathcal{N}$ **do**
      if $C_n\geq M_d$ **then**
        $cost(n)\leftarrow calc\_node\_cost(n,k,m,d)$
        $CS\leftarrow CS\cup\{n\}$
      end if
    end for
    $\overline{CS}\leftarrow$ Sort CS according to $cost(n)$
    $\mathcal{N}^*\leftarrow\{\overline{CS}[1],\overline{CS}[2],\ldots,\overline{CS}[k+m]\}$
    $CM_d\leftarrow CM_d\cup\{(k,m,\mathcal{N}^*)\}$
  end for
end for
for $d\in\mathcal{D}$ **do**
  $fcost_{\min}\leftarrow\infty$
  for $(k,m,\mathcal{N}^*)\in CM_d$ **do**
    if $calc\_file\_cost(w,d,\mathcal{N}^*)<fcost_{\min}$ **then**
      $fcost_{\min}\leftarrow calc\_file\_cost(w,d,\mathcal{N}^*)$
      $CM_{\min}\leftarrow(k,m,\mathcal{N}^*)$
    end if
  end for
  $(k,m,\mathcal{N}^*)\leftarrow CM_{\min}$
  $nodes(d)\leftarrow\mathcal{N}^*$; $erasure\_code(d)\leftarrow(k+m,m)$
  Update $C_n,\forall n\in\mathcal{N}^*$; update $tcost$
end for

V. Permormance Evaluation

In this section, we present the performance evaluation of the proposed MILP and heuristic mechanisms through simulations. The MILP mechanism is written in Python and the Gurobi Optimizer [17] is used. The experiments are performed on a computer with an Intel Core i7-9700K processor running at 3,6 GHz and 32 GB of RAM. We assumed a cloud-edge storage infrastructure that consists of N_c cloud and N_e edge storage nodes. We examined the offline scenario in which all the storage demands are created in advance. Each file $d\in\mathcal{D}$ has a size of $M_d=5$ DUs, it is hosted for a period of $T_d=10$ PUs and it is retrieved $\tau_d=100$ times from the storage service. Furthermore, all files are split into

$k = 4$ fragments ($\mathcal{K}_d = \{4\}, \forall d \in \mathcal{D}$), which are encoded with an erasure code that introduces an overhead of $m = 2$ fragments ($\mathcal{M}_d = \{2\}, \forall d \in \mathcal{D}$). Finally, the minimum required availability for each file is set to 98%. The various monetary cost the delay components, as long as the average availability A_n and the storage capacity C_n of each node $n \in \mathcal{N}$ are presented in detail in Table II.

TABLE II: Simulation setup.

Var.	Value		Unit
M_d	5,	$\forall d \in \mathcal{D}$	DUs
$\mathcal{K}_d$	$\{4\}$,	$\forall d \in \mathcal{D}$	–
$\mathcal{M}_d$	$\{2\}$,	$\forall d \in \mathcal{D}$	–
T_d	10,	$\forall d \in \mathcal{D}$	–
τ_d	100,	$\forall d \in \mathcal{D}$	–
$\mathcal{Q}_d$	$\{A_{\text{req}} = 98.0\%\}$,	$\forall d \in \mathcal{D}$	%
N_c	6		–
N_e	6		–
ρ	$10e^{-6}$		DUs/GET
$D_{\text{spl}\omega_d}$	$\mathcal{U}(0.15, 0.195)$		TUs/split
$D_{\text{mer}\omega_d}$	$\mathcal{U}(0.12, 0.156)$		TUs/merge
$D_{\text{enc}\omega_d}$	$\mathcal{U}(300, 390)$		TUs/DU
$D_{\text{dec}\omega_d}$	$\mathcal{U}(150, 195)$		TUs/DU
	Cloud locations	**Edge devices**	
D_{sn}	$\mathcal{U}(300, 390)$	$\mathcal{U}(100, 130)$	TUs/DU
D_{rn}	$\mathcal{U}(150, 195)$	$\mathcal{U}(50, 65)$	TUs/DU
$D_{\text{ln}\omega_d}$	$\mathcal{U}(0.5, 0.65)$	$\mathcal{U}(0.05, 0.065)$	TUs
P_{sn}	$\mathcal{U}(0.25, 0.325)$	$\mathcal{U}(0.4, 0.52)$	CUs/(DU*PU)
P_{rn}	$\mathcal{U}(200e^{-8}, 260e^{-8})$	$\mathcal{U}(320e^{-8}, 416e^{-8})$	CUs/GET
A_n	$\mathcal{U}(99.9\%, 99.99\%)$	$\mathcal{U}(70.0\%, 75.0\%)$	%
C_n	∞	$\mathcal{U}(2500, 3250)$	DUs

In Figure 1, we examine the effect of the cloud and edge resources in the distribution of the fragments across the infrastructure for 100 storage demands and using the MILP mechanism. We examine six cases of weight assignments to the optimization objectives. In the first five cases, the effect of each optimization criteria in the distribution of the fragments is examined separately, while in the sixth case we assign balanced weights to all objectives so that all criteria are equally taken into account. As expected, when latency (for store or retrieve operations) is the primary optimization objective, the edge resources are preferred compared to the cloud resources. On the other hand, when the monetary storage cost or the availability are the primary objectives the cloud resources are preferred.

Next, in Figure 2, we present the monetary cost (in CUs) for different numbers of files. The white parts of the bars present the cost of the store operation, while the darker parts of the bars present the average cost per retrieval. As expected, the monetary cost increases linearly with the number of hosted files. When latency minimization is the desired objective, the total cost increases as more edge resources are preferred over cloud ones. On the other hand, when the objective is the minimization of the storage/retrieval monetary cost or the maximization of the expected availability of the file, cloud resources are preferred. This is because cloud resources can offer services of abundant capacity with high availability and low cost. When all the objectives are considered in the optimization process, cloud and edge resources are combined in a way that decreases the experienced latency by taking advantage of the edge locations that are close to where the data are needed. Cloud resources are used as a safety net to store part of the fragments and ensure the high availability of the fragments, while they also reduce the monetary cost requirements. Also in all examined cases, the performance of the heuristic mechanism is close to the optimal one, with a difference that increases along with the number of files.

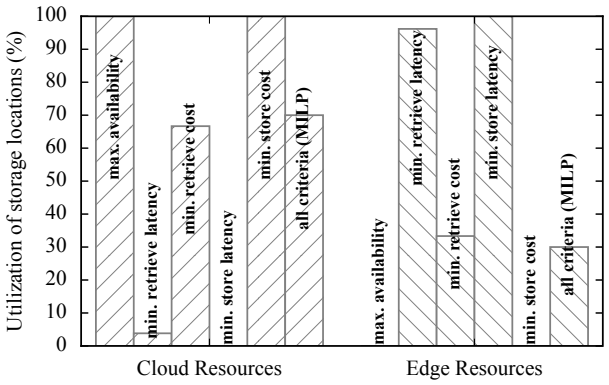

Fig. 1: The utilization of the edge and cloud resources for the different optimization criteria.

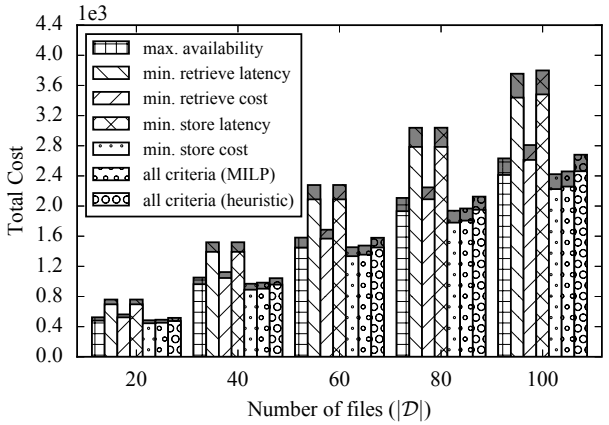

Fig. 2: The total monetary cost for the different optimization criteria for the MILP and heuristic mechanisms.

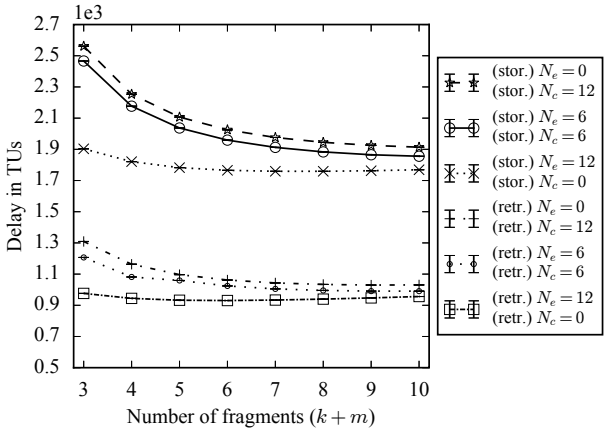

Fig. 3: The effect of the different fragmentation options in the experienced delay.

When all the objectives are equally taken into consideration, the performance of the heuristic mechanism in the achieved monetary cost is inferior to the optimal one, up to 11% when 100 files are hosted over the infrastructure. However, the execution time of the heuristic was 0.02 sec, compared to the 1816 sec that the MILP required for the same scenario.

Finally, in Figure 3 we examine the effect of the different fragmentation options to the experienced latency, by taking all the optimization criteria under consideration. In all cases, the delay decreases as the number of fragments of the file increases. When a file is retrieved, only a subset of the hosted fragments is required and the resources with the best characteristics, with respect to the QoS requirements, are selected. However, when cloud and edge resources are available, the delay decreases with a higher pace by taking advantage of the subset of fragments that is placed on the edge resources. This is an advantage that a distributed infrastructure offers when the resource allocation is performed in a way that is able to profit from network coding, addressing the dynamic nature of edge locations. By distributing the fragments into a way that combines the advantages of the different storage location and by requiring a subset of the stored files to be retrieved, we are able to address the heterogeneous QoS requirements combining each time the characteristics of the storage locations and increasing the value of the whole infrastructure. Comparing the store and retrieve operations, the experienced delay of the store operation is higher and decreases as more locations are available. This happens because the achieved latency for data placement is slower compared to the data recovery latency.

VI. Conclusion

The amount of data created, consumed, and stored is expected to dramatically increase the following years. This fact makes the distributed storage infrastructures and services, critical components for the targeted digital transformation of our society. Edge computing is expected to improve the performance of these systems, while erasure coding techniques increase the security and the availability of data. In our work, we proposed resource allocation mechanisms that take into account these two aspects, by allocating data fragments resulting from the erasure coding operation, jointly to edge and cloud resources based on various constraints and objectives. The developed mechanisms can trade-off performance for execution time. The performed simulation results illustrate the advantages in monetary cost and latency that these mechanisms offer over edge-cloud infrastructures when different criteria need to be optimized simultaneously.

Acknowledgment

The work presented in this paper is supported by the European Union's Horizon 2020 research and innovation program under grant agreement No. 101017168 in the context of the SERRANO project. This is also partly supported by the project "Strengthening Human Resources Research Potential via Doctorate Research" (MIS-5000432), implemented by the Greek State Scholarships Foundation (IKY).

References

[1] IDC and Seagate, "Data age 2025: The evolution of data to life-critical," 2017.

[2] M. Hadji, "Scalable and cost-efficient algorithms for reliable and distributed cloud storage," in *International Conference on Cloud Computing and Services Science*, pp. 15–37, Springer, 2015.

[3] J. Li *et al.*, "Erasure coding for cloud storage systems: A survey," *Tsinghua Science and Technology*, 2013.

[4] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE internet of things journal*, pp. 637–646, 2016.

[5] T. G. Papaioannou *et al.*, "Scalia: An adaptive scheme for efficient multi-cloud storage," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1–10, IEEE, 2012.

[6] G. Liu *et al.*, "An economical and slo-guaranteed cloud storage service across multiple cloud service providers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 9, pp. 2440–2453, 2017.

[7] Y. Mansouri *et al.*, "Brokering algorithms for optimizing the availability and cost of cloud storage services," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*.

[8] Y. Ma *et al.*, "An ensemble of replication and erasure codes for cloud file systems," in *2013 Proceedings IEEE INFOCOM*, pp. 1276–1284, IEEE, 2013.

[9] Q. Zhang *et al.*, "Charm: A cost-efficient multi-cloud data hosting scheme with high availability," *IEEE Transactions on Cloud computing*, pp. 372–386, 2015.

[10] Z. Wu *et al.*, "Spanstore: Cost-effective geo-replicated storage spanning multiple cloud services," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pp. 292–308, 2013.

[11] S. Mu *et al.*, "μlibcloud: Providing high available and uniform accessing to multiple cloud storages," in *2012 ACM/IEEE 13th International Conference on Grid Computing*, pp. 201–208, IEEE, 2012.

[12] H. Abu-Libdeh *et al.*, "Racs: a case for cloud storage diversity," in *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 229–240, 2010.

[13] P. Wang *et al.*, "An ant colony algorithm-based approach for cost-effective data hosting with high availability in multi-cloud environments," in *2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC)*, pp. 1–6, IEEE, 2018.

[14] P. Wang *et al.*, "Optimizing data placement for cost effective and high available multi-cloud storage," *Computing and Informatics*, vol. 39, no. 1-2, pp. 51–82, 2020.

[15] P. Wang *et al.*, "An adaptive data placement architecture in multicloud environments," *Scientific Programming*, vol. 2020, 2020.

[16] D. P. Bertsekas *et al.*, "Rollout algorithms for combinatorial optimization," *Journal of Heuristics*, no. 3, 1997.

[17] "Gurobi optimizer." https://www.gurobi.com/.