# Security-Aware Allocation of Replicated Data in Distributed Storage Systems

Sabrina De Capitani di Vimercati[a], Sara Foresti[b], Giovanni Livraga[c], Pierangela Samarati[d] and Mauro Tedesco

*Computer Science Department, Università degli Studi di Milano, via Celoria 18, Milano, Italy*

Keywords: Security-Aware Data Allocation, Distributed Storage, Data Management, Replica Management, Optimal Allocation.

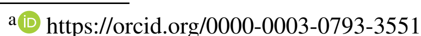
Abstract: Distributed storage systems offer scalable and cost-effective solutions for managing large data collections. A critical factor for the adoption of these systems is the allocation of data (possibly including replicas) to the storage nodes satisfying operational and security requirements, while ensuring economic effectiveness. Appropriate data and replica management provides significant benefits, ranging from enhanced fault tolerance and improved data availability, to reduced latency and optimized workload distribution. A suboptimal placement of data and replicas can instead lead to excessive costs, security risks, and performance bottlenecks. This paper proposes a novel model for permitting data owners to specify in a friendly manner complex data and replica allocation constraints, and an approach for computing optimal (satisfying operational and security requirements and minimizing costs) data allocations in distributed storage environments. Our work aims to improve the reliability, security, and cost-effectiveness of distributed storage systems.

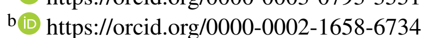
SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

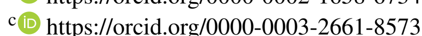
## 1 INTRODUCTION

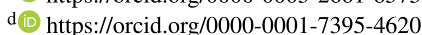
The availability of distributed storage systems, possibly based on cloud/fog/edge paradigms, has revolutionized how large and/or critical data collections can be stored and managed, offering scalable, fault-tolerant, and cost-effective solutions for different application scenarios (e.g., (Russo Russo et al., 2024)). To fully benefit from the features of these distributed systems, it is critical to ensure the storage of data in compliance with operational and security requirements, while maintaining economic effectiveness. An essential aspect of this challenge concerns the optimal allocation of data items (including their possible replicas) across the nodes. Careless or suboptimal data placement can intuitively lead to excessive costs, performance bottlenecks, and/or breaches of security or operational requirements. These requirements, often dictated by the data owner, may impose constraints such as grouping related data items (or data items that are frequently accessed together) on the same node to facilitate joint visibility, ensuring that sensitive collections are fragmented and distributed across separate nodes to avoid exposure, or replicating data among multiple storage nodes. In particular, replicating data can provide several benefits. A first benefit is represented by enhanced fault tolerance: replicas mitigate the risks of data loss due to failures or network outages. Data replication can also help improving data availability, as replicas permit access to data even when some nodes become unavailable, reducing downtime and ensuring service continuity. It can also be beneficial to increase system performance: for example, strategically placing replicas closer to end-users can reduce latency and improve data access speed, enhancing application responsiveness. Furthermore, it permits load balancing, as replicas can help distributing workload across storage nodes, preventing bottlenecks and optimizing resource utilization. When allocating data to the storage nodes, a careful management of the replica plays a critical role, as it directly influences system reliability, performance, and cost.

To address this problem, we propose an approach capable of computing optimal data allocations considering data replication, balancing two different objectives: minimizing economic cost on the one hand,


[a] https://orcid.org/0000-0003-0793-3551


[b] https://orcid.org/0000-0002-1658-6734


[c] https://orcid.org/0000-0003-2661-8573


[d] https://orcid.org/0000-0001-7395-4620



De Capitani di Vimercati, S., Foresti, S., Livraga, G., Samarati, P. and Tedesco, M.
Security-Aware Allocation of Replicated Data in Distributed Storage Systems.
DOI: 10.5220/0013484800003950
In *Proceedings of the 15th International Conference on Cloud Computing and Services Science (CLOSER 2025)*, pages 80-91
ISBN: 978-989-758-747-4; ISSN: 2184-5042
Copyright © 2025 by Paper published under CC license (CC BY-NC-ND 4.0)

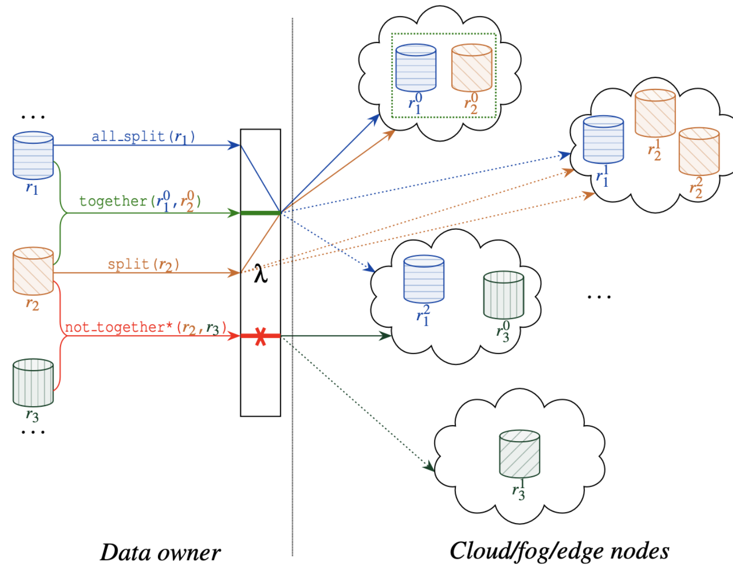



Figure 1: Reference scenario.

and satisfying all requirements imposed by the data owner on the other hand. The contribution of this paper is two-fold. First, we identify and formally define possible requirements that a data owner may need to impose to regulate the overall allocation of data and replicas to the storage nodes. We then present an approach for computing optimal data allocations (i.e., allocations that respect all requirements while minimizing the overall economic cost) in a distributed storage environment. We underline that our approach is agnostic to (and does not focus on) specific storage/processing frameworks (e.g., distributed file systems such as HDFS or IPFS) and considers a more general distributed storage environment, interpreted as a collection of nodes with storage capabilities that can be connected and queried, making it applicable to a broader range of distributed architectures beyond traditional data processing ecosystems.

Our work aims at contributing to the body of literature on efficient and secure data outsourcing strategies in distributed systems, with a specific focus on the critical aspect of replica placement and management. Our approach seeks to improve the reliability, security, and cost-effectiveness of distributed storage systems, making them more accessible and appealing to data owners, across various application scenarios, fostering trust and enabling efficient resource usage in modern data outsourcing scenarios. The remainder of this paper is organized as follows. Section 2 introduces our modeling of nodes and data, and the notion of allocation along with two families of requirements to be considered when defining allocations. Section 3 discusses the first family of requirements, and characterizes the concepts of acceptable nodes that fit the characteristics and needs of the data items to be allocated. Section 4 presents the second family of requirements, which imposes restrictions on data and replica allocation. Section 5 illustrates the problem of computing optimal allocations, and an approach for finding them. Section 6 discusses related works. Finally, Section 7 concludes the paper.

## 2 MODELING

We consider a scenario (see Figure 1) characterized by a data owner who is willing to move their data to the premises of external storage providers (e.g., on cloud, edge, fog nodes). While interested in offloading of storage and management of (possibly replicated) data collections, the data owner needs to enforce restrictions on the nodes storing the data in terms of their characteristics (e.g., node location or in-storage encryption algorithm adopted), as well as in terms of which data items should be or cannot be allocated to the same node. In the following, we introduce our modeling of nodes and of data, together with their replicas, and of data allocations.

**Nodes Modeling.** Our approach is designed to be highly adaptable and not restricted to any specific distributed storage architecture. It is applicable to any architecture that consists of multiple nodes, each potentially having different characteristics (e.g., storage capacity, performance, connectivity, security/privacy features). The flexibility of the approach allows data, including replicas, to be allocated dynamically across nodes. This general applicability makes the proposed

| Nodes | Attributes | | | | | Price |
|---|---|---|---|---|---|---|
| | prov | type | loc | encr | avail | |
| $v_1$ | prov1 | cloud | EU | 3DES | VH | 40 |
| $v_2$ | prov1 | cloud | US | DES | H | 30 |
| $v_3$ | prov3 | cloud | EU | 3DES | M | 55 |
| $v_4$ | prov2 | edge | EU | 3DES | VH | 95 |
| $v_5$ | prov2 | edge | US | DES | L | 90 |
| $v_6$ | prov4 | edge | EU | AES | L | 85 |
| $v_7$ | prov1 | cloud | EU | 3DES | VH | 10 |
| $v_8$ | prov3 | cloud | EU | 3DES | M | 25 |
| $v_9$ | prov2 | edge | EU | 3DES | VH | 70 |
| $v_{10}$ | prov4 | edge | EU | AES | L | 100 |

Figure 2: Abstract representation of a set of nodes and price for their usages (USD/GB).

method suitable for diverse environments, including cloud computing platforms (where centralized data centers can host numerous virtualized storage nodes), fog computing (with distributed devices closer to the network edge for latency-sensitive applications), and edge computing (where data are processed and stored directly on edge devices such as IoT gateways).

In light of this generality, the notion of a “node” is inherently dependent on the architecture considered within the specific application scenario. For example, in cloud computing, a “node” might correspond to a virtual storage instance or even to a cloud plan, while in edge computing, it could represent an individual device or sensor. Regardless of its specific instantiation, a “node” within our framework is defined as the actual place where data items can be stored and is characterized by attributes (e.g., capacity, performance, accessibility, security features). This flexibility ensures that the proposed approach can accommodate various interpretations of a “node,” tailored to the requirements of the considered application and architecture. We then define our problem as the problem of determining which data item has to be allocated to which node of the storage architecture.

For generality and in line with previous works (e.g., (De Capitani di Vimercati et al., 2021a; De Capitani di Vimercati et al., 2021b)), we assume an abstract representation of nodes in terms of their attributes: each node $v$ is then represented as a vector having, for each attribute of interest, one element with the value assumed by it in $v$. Figure 2 illustrates an example of 10 nodes $v_1,\dots,v_{10}$ defined over attributes `prov` (the provider managing the node), `type` (the type of node), `loc` (the geographical location of the node), `encr` (the encryption scheme supported by the node), `avail` (the declared availability).

**Data Modeling.** A peculiarity of our scenario is the consideration of the possibility of replicating data. The computation of the data allocation to the nodes must then consider both the original data items, as well as their replicas. Each data item therefore has

| Resource $r$ | Size (GB) | Num. Replicas |
|---|---|---|
| clinical | 1000 | 1 |
| insurance | 500 | 2 |
| equipment | 250 | 0 |
| research | 300 | 1 |
| staff | 100 | 1 |
| admin | 200 | 1 |
| payroll | 100 | 1 |

Figure 3: Resources of our running example with their size and number of additional replicas.

an original version (i.e., the item itself), and 0 or more additional replicas. Our approach is not restricted to a specific type of data and, for generality and to encompass a variety of different data models (e.g., structured/semi-structured data, documents, files) in the reminder of this paper we refer to the data items to be outsourced with the term *resources*. A resource can then represent any single informative asset of the data owner, defined at a chosen granularity level (from entire documents collections to partitioned data), which may (if so desired and done by the owner) be replicated, and which needs to be outsourced and hence allocated to a node (or a set thereof).

**Example 2.1.** *We refer our examples to the outsourcing of the data assets of a hospital. The resources to be outsourced are the following.*

- `clinical`*: clinical information related to patient, such as their diagnosis, ongoing treatments, past medical history, and so on.*
- `insurance`*; information on patients' insurance and other fiscal data.*
- `equipment`*: information about medical devices and equipment maintained by the hospital for its activities.*
- `research`*: information related to internal and external research activities.*
- `staff`*: personal information of the staff working in the hospital.*
- `admin`*: administrative and management information.*
- `payroll`*: information related to the salary and working position of hospital staff.*

*Figure 3 illustrates these resources along with their size and the number of additional replicas needed for each of them by the hospital. Each replica of each resource is equal in size. For example, resource* `staff` *has 1 additional replica, meaning that the hospital maintains (and wishes to outsource) two copies of the resource, each of which is 100GB in size. Resource* `equipment`*, instead, has 0 additional replicas, meaning that only the original resource (250GB in size) is maintained and has to be outsourced.*

In our modeling, we use notation $r$ to refer to a resource in the set $R$ of resources to be outsourced, for which a certain number $n \geq 0$ of replicas can exist, without distinguishing whether reference is made to the original version of the resource or to one of its replicas. When explicit reference to a specific version of a resource (i.e., its original version *or* one of its replicas) is needed, we distinguish original resources from replicas with superscripts: $r^0$ denotes the original version of $r$, while $r^i$ denotes the $i^{th}$ replica of $r$. We denote the original version and the set of replicas existing for resource $r$ with symbol $\mathbb{R}(r)$. For example, with reference to the resources in Figure 3, consider resource `insurance`, for which 2 replicas are needed in addition to the original resource. When writing `insurance`, we refer to a generic version of the resource without explicit consideration of whether we are referring to the original version or to one replica. When writing $\texttt{insurance}^0$ we explicitly refer to the original version. Also, $\mathbb{R}(\texttt{insurance}) = \{\texttt{insurance}^0, \texttt{insurance}^1, \texttt{insurance}^2\}$, which includes the original version ($\texttt{insurance}^0$) and the first ($\texttt{insurance}^1$) and second ($\texttt{insurance}^2$) replica. For readability, we denote with $\mathcal{R}$ the *overall* set of original resources and their replicas $\mathcal{R} = \bigcup_{r \in R} \mathbb{R}(r)$. For example, with reference to the resources in Figure 3, $\mathcal{R}$ will include 14 resources: the 7 original resources, as well as one replica for each resource but `insurance` (for which 2 replicas are defined), and `equipment` (for which no replica is defined).

**Allocation Modeling.** Given a set $\mathcal{R}$ of resources and a set $\mathcal{V}$ of nodes, our goal is to determine an allocation $\lambda : \mathcal{R} \rightarrow \mathcal{V}$ that maps each resource (original or replica) to a node.

Given a set of nodes, it may be that not every node is suitable to store every resource. This can be due to two main factors. The first concerns the fit between the peculiarities of each node vs. the specific characteristics or requirements of different resources. For instance, certain resources may require high-performance nodes with fast processing capabilities, while others might need high-capacity nodes optimized for bulk storage. The second factor reflects the interplay between resource allocations, where the placement of one resource may influence the suitability of a node for storing another resource (e.g., due to capacity limitations, performance constraints, or operational dependencies).

These two factors motivate two distinct families of requirements, which may be specified by data owners to regulate allocation. The first family of requirements addresses the suitability of individual nodes for specific resources: the enforcement of such constraints determines, for each resource, a set of *acceptable nodes*. The second family of requirements addresses the interplay among allocations: the enforcement of such requirements over acceptable nodes guarantees that the overall allocation fits the desiderata of the owner. We illustrate the first family of requirements along with the notion of acceptable node in Section 3. We then illustrate the second family of requirements in Section 4.

# 3 ACCEPTABLE NODES

The first kind of requirements that should be considered when computing an allocation of data to storage nodes considers the fit between the characteristics of the different nodes, and those of the resources to be allocated. This kind of requirements is specified for each resource $r$ independently, and defines the node characteristics that make a node suitable or unsuitable for $r$. For example, a resource requiring high reliability might be restricted to nodes with robust failover mechanisms, while a resource involving latency-sensitive operations might be allocated to nodes with low communication delay. By capturing these requirements, data owners can ensure that their resources can be allocated only to nodes that meet their needs and expectations. For clarity, in the remainder of this paper we will denote such requirements as *resource-level requirements*. There are several approaches that can be used to specify, and enforce, this kind of requirements (e.g., (De Capitani di Vimercati et al., 2021b; De Capitani di Vimercati et al., 2021a)). We aim at remaining general and do not restrict our approach to operate with any specific approach. For example, the proposal in (De Capitani di Vimercati et al., 2021b) provides a language for specifying complex resource-level requirements, building upon the concept of *base requirement*, denoted $c$. Given a set $\{w_1, \ldots, w_n\}$ of values in the domain of a node attribute (characteristic) `at`, a base requirement on `at` imposes that `at` can assume ($\texttt{at}(w_1, \ldots, w_n)$) or cannot assume ($\neg\texttt{at}(w_1, \ldots, w_n)$) such a set of values. For instance, a base requirement of the form $\texttt{prov}(\text{provA}, \text{provB}, \text{provC})$ states that, to be acceptable, a node must be managed by provider provA, provB, or provC. Starting from base requirements, the specification language in (De Capitani di Vimercati et al., 2021b) permits to express a variety of complex requirements, summarized in Figure 4(a). Those complex requirements can model alternatives among base requirements (ANY), sets of base requirements that must be jointly satisfied (ALL), conditional requirements (IF-THEN), prohibited char-

| |
|---|
| ANY$(c_1, \ldots, c_n)$ |
| ALL$(c_1, \ldots, c_n)$ |
| IF ALL$(c_1, \ldots, c_k)$ THEN ANY$(c_{k+1}, \ldots, c_n)$ |
| FORBIDDEN$(c_1, \ldots, c_n)$ |
| AT_LEAST$(m, (c_1, \ldots, c_n))$ |
| AT_MOST$(m, (c_1, \ldots, c_n))$ |

$(a)$

| |
|---|
| ANY$(\{$`prov`$(\text{prov2}),$`type`$(\text{cloud})\})$ |
| ALL$(\{$`loc`$(\text{EU}),$`avail`$(\text{VH})\})$ |
| IF$($`prov`$(\text{prov1}))$ THEN $($`type`$(\text{cloud}))$ |
| FORBIDDEN$(\{$`prov`$(\text{prov3}),$`type`$(\text{cloud})\})$ |

$(b)$

Figure 4: Complex requirements supported by the language in (De Capitani di Vimercati et al., 2021b) $(a)$ and an example of a set of requirements for resource `clinical` in Figure 3 $(b)$.

acteristic combinations (FORBIDDEN), and a minimum (AT_LEAST) or maximum (AT_MOST) number of base requirements to be satisfied.

Note that, while in principle each version (e.g., original and/or replica) of each resource may be assigned different requirements (and our approach fully supports this scenario), in our examples we assume that all versions of each resource are associated with the same set of requirements and, for readability, we associate resource-level requirements with the generic resource names. Figure 4(b) illustrates a set of 4 sample resource-level requirements for resource `clinical` in Figure 3. The first requirement demands that `clinical` is to be outsourced to a node that is managed by prov2 *or* is of type cloud. The second requirement demands that the node must be located in EU *and* guarantee a very high availability. The third requirement demands that if a node is managed by prov1, it must be of type cloud. The last requirement prohibits `clinical` to be stored at a node managed by prov3 and that is of type cloud.

The resource-level requirements associated with a resource $r$ restrict the possible nodes that can be considered for allocating the original version of $r$ and its replicas, based on the values the nodes assume for the attributes that characterize them. Regardless of the approach adopted for specifying and enforcing resource-level requirements, we call a node $v$ that satisfies all requirements specified for a resource $r$ as an *acceptable* node for $r$. For example, with reference to the nodes in Figure 2, the resources in Figure 3 and the requirements in Figure 4(b), the set of acceptable nodes for `clinical` includes $v_1$, $v_4$, $v_7$, and $v_9$. On the contrary, for example, node $v_2$ is not acceptable for `clinical` since its characteristics do not satisfy the second requirement of Figure 4(b) (it is not located in EU nor has a very high availability).

Given a set of nodes and a set of resources with resource-level requirements, the identification of the

| | Nodes | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Resources** | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ | $v_9$ | $v_{10}$ |
| `clinical` | ✓ | | | ✓ | | | ✓ | | ✓ | |
| `insurance` | ✓ | | | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| `equipment` | ✓ | | | ✓ | | | ✓ | | ✓ | |
| `research` | | | | ✓ | | | | | ✓ | |
| `staff` | ✓ | | | ✓ | | | ✓ | | ✓ | |
| `admin` | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| `payroll` | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figure 5: An example of acceptable nodes for the resources in Figure 3.

acceptable nodes for each resource can be automatically computed in different ways (e.g., leveraging a Boolean formulation of the problem as proposed in (De Capitani di Vimercati et al., 2021b)). Figure 5 reports an example of acceptable nodes, among those considered in our running example in Figure 2, for the resources in Figure 3.

# 4 REPLICA AND ALLOCATION CONSTRAINTS

Replica and allocation constraints are used to specify requirements that refer to the overall allocation of sets of resources, and of the replicas. They are enforced to determine which resource will be allocated to which node, among the acceptable ones for that resource. While the resource-level requirements illustrated in Section 3 specify, for each resource, a set of conditions that identify acceptable nodes, the main goal of the constraints introduced in this section is to force the joint/separate allocation of sets of resources (e.g., the replicas of a resource and its original version) to nodes not based on the characteristics of the nodes, bur rather on the resources themselves.

We introduce and define 8 kinds of replica and allocation constraints: `together`, `together`$^*$, `all_together`, `not_together`, `not_together`$^*$, `split`, `all_split`, and `alone`. Constraints `together`, `together`$^*$, and `all_together` force the joint allocation of resources on the same node. Constraints `not_together` and `not_together`$^*$ force fragmentation of resources across different nodes. Constraints `split`, `all_split` force different allocations for the original version and the replica of a resource. Constraint `alone` force the allocation of a resource to be different from any other resource. In the remainder of this section, we formally illustrate and discuss these constraints. For simplicity, we will denote with $r$ and $s$ two resources in $R$, with $\rho$ an instance of $r$ in $\mathbb{R}(r)$, and with $\sigma$ an instance of $s$ in $\mathbb{R}(s)$.

| | |
|---|---|
| $c_1$: | together(equipment$^0$,clinical$^0$) |
| $c_2$: | together$^*$(staff,research) |
| $c_3$: | all_together(staff,payroll) |
| $c_4$: | not_together(payroll$^0$,insurance$^0$) |
| $c_5$: | not_together(payroll$^0$,insurance$^1$) |
| $c_6$: | not_together(payroll$^0$,insurance$^2$) |
| $c_7$: | not_together$^*$(insurance,clinical) |
| $c_8$: | split(clinical) |
| $c_9$: | all_split(staff) |
| $c_{10}$: | all_split(insurance) |
| $c_{11}$: | alone(admin$^0$) |
| $c_{12}$: | alone(admin$^1$) |

Figure 6: An example of a set of replica and allocation constraints for the resources in Figure 3.

- together($\rho,\sigma$). A together constraint is defined over two specific versions $\rho \in \mathbb{R}(r)$ and $\sigma \in \mathbb{R}(s)$ of two resources $r$ and $s$, and demands that they are both allocated to the same node (e.g., to account for the fact that those resources are oftentimes accessed together, or combined).
  Formally, a together($\rho,\sigma$) constraint is satisfied by an allocation $\lambda$ iff $\lambda(\rho) = \lambda(\sigma)$. This constraint can be formulated by the owner to restrict the allocation of a pair of resources and, as such, demands the identification of two specific versions $\rho$ of $r$ and $\sigma$ of $s$ to be involved in the constraint. In other words, this constraint demands the specification of *which* versions of $r$ and $s$ are to be allocated together. For example, constraint $c_1$ in Figure 6 requires the original versions of equipment and of clinical to be allocated on the same node, modeling the fact that these two resources are often to be accessed together.

- together$^*$($r$,$s$). A together$^*$ constraint is defined over two generic resources $r$ and $s$, and demands that at least one version of $r$ be allocated to the same node of at least one version of $s$. In other words, it requires the allocation to include a node that stores at least one version of $r$ and one version of $s$. We expect this constraint to be formulated, similarly to the together constraint, over resources that are expected to be frequently accessed together. The main difference in semantics with the together constraint is that, in this case, it is not necessary to specify *which* versions of the two resources must be placed together.
  Formally, a together$^*$($r$,$s$) is satisfied by an allocation $\lambda$ iff $\exists\, \rho,\sigma$ s.t. $\rho \in \mathbb{R}(r), \sigma \in \mathbb{R}(s) : \lambda(\rho) = \lambda(\sigma)$. For example, constraint $c_2$ in Figure 6 requires that at least one version of staff and at least one version of research be allocated to the same node.

- all_together($r$,$s$). An all_together constraint is defined over two generic resources $r$ and $s$, and demands that whenever a node stores a version of $r$, it also stores a version of $s$. In other words, it guarantees that $r$ is always accompanied by $s$. Note the difference in semantics with the together$^*$ constraint, which is also formulated over two *generic* versions of a pair of resources, but is satisfied when at least one pair of versions are allocated to the same node.
  Formally, an all_together($r$,$s$) is satisfied by an allocation $\lambda$ iff $\forall \rho \in \mathbb{R}(r), \exists \sigma \in \mathbb{R}(s) : \lambda(\rho) = \lambda(\sigma)$. For example, constraint $c_3$ in Figure 6 requires that, when a node stores a version of staff, it also stores at least one version of payroll. We note that when $|\mathbb{R}(r)| > |\mathbb{R}(s)|$, then the satisfaction of this constraint inevitably implies that more than one replica of $r$ be allocated to the same node (as clearly no more than $|\mathbb{R}(s)|$ nodes can be considered for satisfying this constraint).

- not_together($\rho,\sigma$). A not_together constraint is defined over two specific versions $\rho \in \mathbb{R}(r)$ and $\sigma \in \mathbb{R}(s)$ of two resources $r$ and $s$, and demands that they are not allocated to the same node.
  Formally, a not_together($\rho,\sigma$) constraint is satisfied by an allocation $\lambda$ iff $\lambda(\rho) \neq \lambda(\sigma)$. Like the together constraint, this constraint can be formulated by the owner to restrict the allocation of a pair of resources and, as such, demands the identification of two specific versions $\rho$ of $r$ and $\sigma$ of $s$ to be specified in the constraint. In other words, this constraint demands the specification of *which* versions of $r$ and $s$ should not be allocated together. This general constraint is expected to be formulated for performance reasons: for example, to ensure concurrent access to both resources reducing delays and bottlenecks that may be experienced if $\rho$ and $\sigma$ are large and the available nodes do not exhibit excellent performance. The constraint can also be used to guarantee service continuity in case of node failure: for example, if $\rho$ and $\sigma$ are allocated at different nodes, the unavailability of $\rho$ compromises only the functions that directly depend on $\rho$, while still permitting all the activities that rely on $\sigma$. For example, constraints $c_4$, $c_5$ and $c_6$ in Figure 6 demand that the original version of payroll is not allocated together with instances of insurance.

- not_together$^*$($r$,$s$). A not_together$^*$ constraint is defined over two generic resources $r$ and

$s$, and demands that no version of $s$ be allocated to a node that stores a version of $s$, and vice versa. In other words, it requires the allocation not to include a node that stores one version of $r$ and one version of $s$. Note the difference in semantics with the not_together constraint, which is formulated over two *specific* versions of a pair of resources. In particular, while the not_together constraint can be formulated for performance reasons, a not_together* constraint models *confidentiality constraints*, sets (pairs in this case) of resources that should *never* be visible together as their association is considered sensitive. Note that a not_together* constraint can also be formulated as a set of not_together constraints (one for each combination of a version of $r$ and a version of $s$).
Formally, a not_together*$(r,s)$ is satisfied by an allocation $\lambda$ iff $\forall\rho \in \mathbb{R}(r), \forall\sigma \in \mathbb{R}(s) : \lambda(\rho) \neq \lambda(\sigma)$. For example, constraint $c_7$ in Figure 6 prevents having, on the same node, versions of insurance and versions of clinical, accounting for the fact that these two resources should not be visible together.

- split$(r)$. A split constraint is formulated over a generic version of a resource $r$, and demands that all replicas of $r$ be not allocated to the node to which the original version $r^0$ is allocated. This constraint can be formulated for security reasons, to improve availability and resilience to node failures. Note that a split constraint can also be formulated as a set of not_together constraints of the form not_together$(r^0, r^i)$, for each $r^i \in \mathbb{R}(r)$ with $i \neq 0$.
Formally, a split$(r)$ constraint is satisfied by an allocation $\lambda$ iff $\forall\rho \in \mathbb{R}(r) \setminus \{r^0\} : \lambda(\rho) \neq \lambda(r^0)$. For example, constraint $c_8$ in Figure 6 requires all replicas of clinical to be allocated to a node different from that to which clinical$^0$ is allocated.

- all_split$(r)$. An all_split constraint is formulated over a generic version of a resource $r$, and extends the split constraint demanding that no two versions (original nor replicas) of $r$ be allocated to the same node. The satisfaction of this constraint can further increase availability and resilience to node failures (each version of a resource is allocated to a different node), while possibly requiring a large number of nodes to be included in the allocation. A all_split constraint can also be formulated as a set of not_together constraints (one for each combination of two versions of $r$).
Formally, an all_split constraint is satisfied by an allocation $\lambda$ iff $\forall\rho, \sigma \in \mathbb{R}(r), \rho \neq \sigma : \lambda(\rho) \neq \lambda(\sigma)$. For example, constraint $c_9$ ($c_{10}$, resp.) in Figure 6 requires all versions of staff (of insurance, resp.) to be spread across different nodes.

- alone$(\rho)$. An alone constraint is formulated over a specific version of a resource $\rho \in \mathbb{R}(r)$ and requires it is not allocated to a node where other resources (be them original or replicas) as well as replicas of $r$ are also allocated. In other words, it requires $\rho$ to be allocated alone to a node. An alone constraint can also be formulated as a set of not_together constraints of the form not_together$(\rho, x)$, with $x$ any possible instance of any possible resource.
Formally, an alone$(\rho)$ constraint is satisfied by an allocation $\lambda$ iff $\forall\sigma \in \mathcal{R} \setminus \{\rho\} : \lambda(\rho) \neq \lambda(\sigma)$. For example, constraints $c_{11}$ and $c_{12}$ in Figure 6 require, respectively, the original version and the replica of admin to be allocated to nodes where no other resources are allocated.

# 5 OPTIMAL ALLOCATION

In this section, we illustrate our notion of optimal allocation of resources to nodes (Section 5.1), and illustrate a binary programming-based modeling that permits its computation (Section 5.2).

## 5.1 Problem Definition

We are interested in computing an allocation that satisfies *all* the requirements specified for the resources to be outsourced: in other words, we are interested in an allocation that satisfies all resource-level requirements (and hence for which each original resource and each replica $\rho$ is allocated at a node that is acceptable for $\rho$, Section 3), and which satisfies all replica and allocation constraints (Section 4). We define such an allocation as a *correct allocation*. Given a set $\mathcal{C}$ of resource-level requirements and of replica and allocation constraints, we denote the correctness of an allocation $\lambda$ w.r.t. $\mathcal{C}$ with notation $\lambda \models \mathcal{C}$.

In principle, given a set of resources with a set of resource-level requirements and replica and allocation constraints, there may exist different correct allocations, possibly characterized by different economic costs (as different nodes/providers may charge different costs for providing their services). We then aim at computing an *optimal* allocation of resources to nodes, meaning an allocation that, besides being

correct, also minimizes the overall economic cost of the allocation.

Given a set $\mathcal{R}$ of resources and a set $\mathcal{V}$ of nodes, the overall cost $cost(\lambda)$ of an allocation $\lambda : \mathcal{R} \rightarrow \mathcal{V}$ is therefore estimated as $cost(\lambda) = \sum_{\rho\in\mathcal{R}} cost(\rho, \lambda(\rho))$. The problem of determining an optimal allocation can therefore be formulated as follows.

**Problem 5.1** (Optimal allocation)**.** *Given a set $\mathcal{R}$ of resources associated, a set $\mathcal{C}$ of resource-level requirements and of replica and allocation constraints, and a set $\mathcal{V}$ of nodes, determine an allocation $\lambda : \mathcal{R}\rightarrow\mathcal{V}$ of resources to nodes such that* i) $\lambda \models \mathcal{C}$ *(i.e., the allocation is correct); and* ii) $\nexists\lambda' : \mathcal{R}\rightarrow\mathcal{V}, \lambda'\neq\lambda$, *such that $\lambda' \models \mathcal{C}$ and $cost(\lambda') < cost(\lambda)$.*

## 5.2 Computing an Optimal Allocation

To compute an optimal allocation, we interpret Problem 5.1 as a *binary programming problem*, which is formulated as follows: given a set of *binary variables*, a set of *constraints* over them, and an *objective function*, determine an assignment of values to variables that *i)* satisfies all the constraints; and *ii)* minimizes the value of the objective function. Our interpretation of Problem 5.1 therefore assumes: the objective function as the cost function of the allocation to be minimized; the constraints as the resource-level requirements and replica and allocation constraints, which are to be satisfied; and the binary variables as a set of variables, which we now illustrate, modeling the allocation (or non-allocation) of each resource $\rho \in \mathcal{R}$ to each node $v \in \mathcal{V}$. In particular, for each resource $\rho_i$ and each node $v_z$ we define a Boolean variable $a_{i,z}$ that has value 1 iff resource $\rho_i$ is allocated to node $v_z$, 0 otherwise (i.e., $a_{i,z} = 1 \Longleftrightarrow \lambda(\rho_i) = v_z$).

Having introduced our variables, we can now illustrate the constraints, as well as the objective function, needed for the computation of a solution to our binary programming interpretation of Problem 5.1. In the following formulation, we use variable $\bar{a}$ to model acceptable allocations (i.e., allocations that do not violate any resource-level requirement, see Section 3): given a resource $\rho_i$ and a node $v_z$, $\bar{a}_{i,z} = 1$ iff $v_z$ is an *acceptable* node for $\rho_i$.

**Constraints.** Our constraints need to guide assignment of values to our binary variables $a_{i,z}$, for all $\rho_i \in \mathcal{R}$ and $v_z \in \mathcal{V}$, such that the assignment: *i)* represents an allocation; *ii)* allocates each resource to one among its acceptable nodes; and *iii)* satisfies all the replica and allocation constraints.

- *Allocation.* Each resource must be allocated to exactly one node. We model this requirement with the following constraint:

$$\prod_{i=1}^{|\mathcal{R}|}\left(\sum_{z=1}^{|\mathcal{V}|} a_{i,z}\right) = 1$$

For this constraint to assume value 1, for each resource $\rho_i$, the innermost sum (over all $a_{i,z}$ over all nodes $v_z$) must be equal to 1. This requires that, for each resource $\rho_i$, there is one and only one node $v_z$ for which $a_{i,z} = 1$, hence implying that each resource is allocated to exactly one node.

- *Acceptable nodes.* Each resource must be allocated to an acceptable node. We model this requirement with the following constraint:

$$\prod_{i=1}^{|\mathcal{R}|}\left(\sum_{z=1}^{|\mathcal{V}|} \bar{a}_{i,z} \cdot a_{i,z}\right) = 1$$

This constraint extends the allocation constraint by considering for each resource, in the innermost sum, the product $\bar{a}_{i,z} \cdot a_{i,z}$. This product is equal to 1 iff $\rho_i$ is allocated to a node $v_z$ *and* $v_z$ is an acceptable node for $\rho_i$. By requiring the sum over all nodes of this product to be equal to 1, for each resource, this constraint then implies that all resources are allocated to acceptable nodes.

- `together` *requirement.* Given two specific versions $\rho_i \in \mathbb{R}(r)$ and $\rho_j \in \mathbb{R}(s)$ of two resources $r, s \in R$ involved in a `together` requirement, an allocation must place both resources on the same node. We model this requirement with the following constraint, defined for each `together` requirement:

$$\forall \mathtt{t} = \mathtt{together}(\rho_i, \rho_j) : \sum_{z=1}^{|\mathcal{V}|} (a_{i,z} \cdot a_{j,z}) = 1$$

Intuitively, given a node $v_z$ and two resources $\rho_i$ and $\rho_j$, we have that $(a_{i,z} \cdot a_{j,z})$ equals 1 iff $v_z$ stores both $\rho_i$ and $\rho_j$. Demanding the sum over all nodes to be equal to 1 implies that there is exactly one node that stores both $\rho_i$ and $\rho_j$.

- `together`$^*$ *requirement.* Given two generic resources $r$ and $s$ involved in a `together`$^*$ requirement, an allocation must place at least one version of $r$ and at least one version of $s$ on the same node. We model this requirement with the following constraint, defined for each `together`$^*$ requirement:

$$\forall \mathtt{t}^* = \mathtt{together}^*(r, s) :$$
$$\sum_{z=1}^{|\mathcal{V}|}\left(\sum_{\substack{\rho_i\in\mathbb{R}(r),\\ \rho_j\in\mathbb{R}(s)}} (a_{i,z} \cdot a_{j,z})\right) \geq 1$$

Given a node $v_z$, the sum over all versions $\rho_i$ and $\rho_j$ of $r$ and $s$, respectively, of $(a_{i,z} \cdot a_{j,z})$ is greater than or equal to 1 if $v_z$ stores at least one version $\rho_i$ of $r$ and $\rho_j$ of $s$.

- `all_together` *requirement.* Given two generic resources $r$ and $s$ involved in an `all_together` requirement, whenever a node $v_z$ stores a version of $r$, then it must also store *at least* one version of $s$. We model this requirement with the following constraint, defined for each `all_together` requirement:

$$\forall \texttt{at} = \texttt{all_together}(r,s):$$
$$\prod_{\rho_i \in \mathbb{R}(r)} \left( \sum_{z=1}^{|\mathcal{V}|} a_{i,z} \cdot \big( \sum_{\rho_j \in \mathbb{R}(s)} a_{j,z} \big) \right) \geq 1$$

For this constraint to hold, the parameter of the outermost product must be greater than or equal to 1 for all versions $\rho_i \in \mathbb{R}(r)$ of $r$. Consider a specific version $\rho_i$ of $r$ and a node $v_z$: $a_{i,z} \cdot (\sum_{\rho_j \in \mathbb{R}(s)} a_{j,z})$ is equal to or greater than 1 if $v_z$ stores $\rho_i$ and at least one version $\rho_j$ of $s$. The intermediate sum over all nodes ensures that there is at least one node for which the condition holds. The outermost product over all versions of $r$ then ensures that the overall condition is satisfied for all versions of $r$.

- `not_together` *requirement.* Given two specific versions $\rho_i \in \mathbb{R}(r)$ and $\rho_j \in \mathbb{R}(s)$ of two resources $r,s \in R$ involved in a `not_together` requirement, an allocation cannot place both resources on the same node. We model this requirement with the following constraint, defined for each `not_together` requirement:

$$\forall \texttt{nt} = \texttt{not_together}(\rho_i,\rho_j):$$
$$\sum_{z=1}^{|\mathcal{V}|} (a_{i,z} \cdot a_{j,z}) = 0$$

Given a node $v_z$, we have that $(a_{i,z} \cdot a_{j,z})$ equals 0 iff at least one among $a_{i,z}$ and $a_{j,z}$ is equal to 0 (i.e., at least one among $\rho_i$ and $\rho_j$ is not stored at $v_z$). Requiring the sum over all nodes to be equal to 0 then implies that no node stores at the same time $\rho_i$ and $\rho_j$.

- `not_together`* *requirement.* Given two resources $r$ and $s$ involved in a `not_together`* requirement, an allocation cannot place a version of $r$ on a same node that stores a version of $s$. We model this requirement with the following constraint, defined for each `not_together`* requirement:

$$\forall \texttt{nt}^* = \texttt{not_together}^*(r,s):$$
$$\sum_{z=1}^{|\mathcal{V}|} \left( \sum_{\substack{\rho_i \in \mathbb{R}(r),\\ \rho_j \in \mathbb{R}(s)}} a_{i,z} \cdot a_{j,z} \right) = 0$$

Given a node $v_z$, requiring the sum over all versions of $r$ and $s$ of $a_{i,z} \cdot a_{j,z}$ (i.e., the innermost sum) to be equal to 0 implies that no node $v_z$ stores at the same time a version of $r$ and a version of $s$. The outermost sum enforces this guarantee for all nodes.

- `split` *requirement.* Given a resource $r$, an allocation must not place the original version $r^0$ on the nodes storing the replicas. We model this requirement with the following constraint, defined for each `split` requirement:

$$\forall \texttt{sp} = \texttt{split}(r): \sum_{z=1}^{|\mathcal{V}|} a_{r^0,z} \cdot \left( \sum_{\rho_i \in \mathbb{R}(r) \setminus \{r^0\}} a_{i,z} \right) = 0$$

Given a node $v_z$, $a_{r^0,z} \cdot \left( \sum_{\rho_i \in \mathbb{R}(r) \setminus \{r^0\}} a_{i,z} \right)$ is equal to 0 if the original $r^0$ is not allocated to the same node as a replica $\rho_i \in \mathbb{R}(r) \setminus \{r^0\}$ of $r$. The outermost sum enforces this guarantee for all nodes.

- `all_split` *requirement.* Given a resource $r$, an allocation must not place two versions of $r$ on the same node. We model this requirement with the following constraint, defined for each `all_split` requirement:

$$\forall \texttt{as} = \texttt{all_split}(r):$$
$$\sum_{z=1}^{|\mathcal{V}|} \left( \sum_{\substack{\rho_i,\rho_j \in \mathbb{R}(r),\\ \rho_i \neq \rho_j}} a_{i,z} \cdot a_{j,z} \right) = 0$$

Similarly to the previous constraint, the innermost sum guarantees that, given a node $v_z$, no pair of versions of $r$ are allocated together at $v_z$. The outermost sum enforces this guarantee for all nodes.

- `alone` *requirement.* Given a specific version $\rho_i$ of a resource $r \in R$, an allocation must not place it on a node storing also another version of another resource. We model this requirement with the following constraint, defined for each `alone` requirement:

| Resources | $v_1$ | $v_2$ | $v_3$ | $v_4$ | $v_5$ | $v_6$ | $v_7$ | $v_8$ | $v_9$ | $v_{10}$ |
|---|---|---|---|---|---|---|---|---|---|---|
| clinical$^0$ | | | | | | | ✓ | | | |
| clinical$^1$ | ✓ | | | | | | | | | |
| insurance$^0$ | | | | ✓ | | | | | | |
| insurance$^1$ | | | | | | ✓ | | | | |
| insurance$^2$ | | | | | | | | | ✓ | |
| equipment$^0$ | | | | | | | ✓ | | | |
| research$^0$ | | | | | | | | | ✓ | |
| research$^1$ | | | | | | | | | ✓ | |
| staff$^0$ | | | | | | | | | ✓ | |
| staff$^1$ | | | | | | | ✓ | | | |
| admin$^0$ | | | | | | | | ✓ | | |
| admin$^1$ | | | ✓ | | | | | | | |
| payroll$^0$ | | | | | | | ✓ | | | |
| payroll$^1$ | | | | | | | | | ✓ | |

Figure 7: Optimal allocation of the resources in Figure 3 to the nodes in Figure 2 (acceptable nodes are denoted with a colored background).

$$\forall \mathtt{a} = \mathtt{alone}(\alpha) : \sum_{z=1}^{|\mathcal{V}|} \left( a_{i,z} \cdot \sum_{\rho_j \in \{\mathcal{R} \setminus \{\rho_i\}\}} a_{j,z} \right) = 0$$

Given a node $v_z$, $a_{i,z} \cdot \sum_{\rho_j \in \{\mathcal{R} \setminus \{\rho_i\}\}} a_{j,z}$ is equal to 0 if $v_z$ does not store both $\rho_i$ and a version of another resource. The outermost sum enforces this guarantee for all nodes.

**Objective Function.** The objective function of our binary programming problem, which needs to be minimized, models the cost of the allocation represented by the allocation variables $a_{i,z}$. The allocation cost illustrated in Section 5.1 is therefore computed taking into account the values assumed by $a_{i,z}$, to guarantee equivalence between the binary programming problem and Problem 5.1. The objective function is defined as follows:

$$min \sum_{i=1}^{|\mathcal{R}|} \sum_{z=1}^{|\mathcal{V}|} a_{i,z} \cdot size(\rho_i) \cdot price(v_z)$$

where $size(\rho_i)$ represents the size of resource $\rho_i$ (e.g., in GB) and $price(v_z)$ represents the unitary cost (e.g., in USD/GB) to be paid for using $v_z$. Note that, for simplicity and in line with previous works (e.g., (De Capitani di Vimercati et al., 2021a)), in our examples we consider a linear cost function and estimate the cost $cost(\rho, v)$ of allocating resource $\rho$ to node $v$ by multiplying the size of $\rho$ (e.g., in GB) by the unitary storage price (e.g., in USD/GB) that is applied by $v$, with the note that such an estimate may consider different cost factors (e.g., transfer costs) and/or cost functions.

Figure 7 illustrates an optimal allocation computed over the resources in Figure 3 and the acceptable plans in Figure 5, considering the replica and allocation constraints in Figure 6.

The problem can be efficiently solved using off-the-shelf optimization solvers; for instance, we implemented our approach using Google OR-Tools, demonstrating its effectiveness in computing optimal allocations respecting all specified constraints.

# 6 RELATED WORK

This work contributes to different lines of research at the intersection between distributed storage systems and data allocation methodologies, supporting owner-specified requirements in storage selection.

The majority of works addressing the problem of computing optimal allocations in distributed systems do not take into consideration specific requirements imposed by data owners, but rather focus on guaranteeing recovery possibility (e.g., (Jakovetić et al., 2015; Leong et al., 2012; Roshandeh et al., 2017; Bhattacharya et al., 2019; Peng et al., 2021)). For example, the approach in (Leong et al., 2012) introduces coding strategies to store data optimally for maximizing reliability in terms of recovery possibility. The approach in (Roshandeh et al., 2017) focuses on heterogeneous data in distributed storage, proposing an approach that assumes data of the same type are minimally spread across nodes. The approach in (Bhattacharya et al., 2019) aims to balance storage and network costs, focusing on bandwidth-constrained environments. The approach in (Jakovetić et al., 2015) proposes a distributed algorithm for optimizing data allocations based on local communication between nodes, ensuring data retrieval is achieved with nodes only communicating with neighboring ones. The approach in (Peng et al., 2021) focuses instead on determining an optimal subset of nodes for storing data. A distributed storage system for edge and IoT devices with proof of replication is proposed in (Wu et al., 2022). Our work differs from those falling in this line of research due to the fact that we are not concerned with coding and fragmenting data across nodes, as we aim at supporting heterogeneous architectures (e.g., cloud, fog, and edge) and complex owner-defined constraints, regulating allocations of data and replica to optimally place them in a distributed architecture.

Another related line of work addresses the problem of cloud plan selection, possibly in the context of multi-cloud scenarios and possibly supporting owner-specified requirements. Approaches that permit owners to specify resource-level requirements are orthog-

onal to our work and can be used to determine acceptable nodes on which then computing optimal allocations. Other approaches falling in this category do not typically support arbitrary owner-specified requirements or do not consider replica management, which is a primary objective of our work. The approach in (Ruiz-Alvarez and Humphrey, 2012) proposes an automated solution for selecting storage providers. While sharing with our work some support for user-specified desiderata, this approach does not explicitly support the management of replicas and of their allocation, nor the management of complex constraints regulating the joint/disjoint allocation of data to the same provider, and has a specific focus on cost and performance factors. The approaches proposed in (Dastjerdi and Buyya, 2014; Esposito et al., 2016; De Capitani di Vimercati et al., 2019) adopt fuzzy logic and reasoning for cloud providers selection. In particular, fuzzy logic is adopted in (De Capitani di Vimercati et al., 2019) for language. The approach in (De Capitani di Vimercati et al., 2021b) proposes a specification language for permitting owners to formulate requirements and preferences for cloud plan selection and formal model and different strategies for reasoning on requirements, preferences, and acceptable plans. These approaches are complementary to ours, and can be used for the identification of acceptable nodes that are then used to compute our optimal allocations. The work presented in (Taha et al., 2017), while aligning with our objective of facilitating the selection of multiple providers or services, bases its requirements on the importance levels assigned to predefined Service Level Objectives (SLOs). Other related efforts focus on solving various aspects of resource allocation optimization, such as considering provider workloads (Wendell et al., 2010), addressing fault tolerance mechanisms (Jhawar et al., 2013), leveraging multi-cloud solutions for application development and management (Ferry et al., 2018), and integrating multi-cloud storage systems with existing NAS-based programs (Chen and Zadok, 2019).

Related yet orthogonal works have investigated approaches for permitting owners effectively protecting and securely deleting resources while relying on decentralized cloud services for storage. For example, the approach in (Bacis et al., 2020) combines All-Or-Nothing-Transform for strong resource protection, and ad-hoc strategies for slicing resources and for their decentralized allocation in the storage network. Our work builds on existing research in distributed storage optimization and multicloud data placement, extending their applicability to modern architectures (e.g., fog and edge computing) and ensuring full support for obeying owner-specified requirements and managing replica, while ensuring economic efficiency.

## 7 CONCLUSIONS

This paper presented an approach to the optimal allocation of data and replicas in distributed storage systems, with a focus on balancing economic cost and operational constraints. Our approach can fit diverse architectural environments (e.g., cloud, fog, and edge computing) by generalizing the concept of a "node", and permitting to accommodate varying requirements and characteristics. A peculiarity of our approach consists in permitting owners to specify in a friendly way, and have enforced in the computation of the allocation, complex requirements and constraints that take into consideration both the characteristics of the storage nodes, as well as the interplay among the allocations of different data items and replicas. To enable the computation of optimal allocations, satisfying all constraints while minimizing the overall economic cost entailed by the allocation, this work proposed a formulation of the problem in terms of a binary programming problem, which can then be efficiently solved leveraging off-the-shelves solvers. Further research will include the definition of additional constraints that could be specified and taken into consideration when computing allocations.

## ACKNOWLEDGEMENTS

This work was supported in part by the EC under project GLACIATION (101070141), by the Italian MUR under PRIN projects POLAR (2022LA8XBH) and KURAMi (20225WTRFN), and by project SERICS (PE00000014) under the MUR NRRP funded by the EU - NGEU. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the Italian MUR. Neither the European Union nor the Italian MUR can be held responsible for them.

## REFERENCES

Bacis, E., De Capitani di Vimercati, S., Foresti, S., Paraboschi, S., Rosa, M., and Samarati, P. (2020). Securing resources in decentralized cloud storage. *IEEE TIFS*, 15(1):286–298.

Bhattacharya, H., Chattopadhyay, S., Chattopadhyay, M., and Banerjee, A. (2019). Storage and bandwidth op-

timized reliable distributed data allocation algorithm. *IJACI*, 10(1):78–95.

Chen, M. and Zadok, E. (2019). Kurma: Secure geo-distributed multi-cloud storage gateways. In *Proc. of ACM SYSTOR 2019*, Haifa, Israel.

Dastjerdi, A. V. and Buyya, R. (2014). Compatibility-aware cloud service composition under fuzzy preferences of users. *IEEE TCC*, 2(1):1–13.

De Capitani di Vimercati, S., Foresti, S., Livraga, G., Piuri, V., and Samarati, P. (2019). A fuzzy-based brokering service for cloud plan selection. *IEEE ISJ*, 13(4):4101–4109.

De Capitani di Vimercati, S., Foresti, S., Livraga, G., Piuri, V., and Samarati, P. (2021a). Security-aware data allocation in multicloud scenarios. *IEEE TDSC*, 18(5):2456–2468.

De Capitani di Vimercati, S., Foresti, S., Livraga, G., Piuri, V., and Samarati, P. (2021b). Supporting user requirements and preferences in cloud plan selection. *IEEE TSC*, 14(1):274–285.

Esposito, C., Ficco, M., Palmieri, F., and Castiglione, A. (2016). Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory. *IEEE TC*, 65(8):2348–2362.

Ferry, N., Chauvel, F., Song, H., Rossini, A., Lushpenko, M., and Solberg, A. (2018). CloudMF: Model-Driven management of multi-cloud applications. *ACM TOIT*, 18(2):16:1–16:24.

Jakovetić, D., Minja, A., Bajović, D., and Vukobratović, D. (2015). Distributed storage allocations for neighborhood-based data access. In *Proc. of IEEE ITW 2015*, Jerusalem, Israel.

Jhawar, R., Piuri, V., and Santambrogio, M. (2013). Fault tolerance management in cloud computing: a system-level perspective. *IEEE ISJ*, 7(2):288–297.

Leong, D., Dimakis, A. G., and Ho, T. (2012). Distributed storage allocations. *IEEE TIT*, 58(7):4733–4752.

Peng, P., Noori, M., and Soljanin, E. (2021). Distributed storage allocations for optimal service rates. *IEEE TCOM*, 69(10):6647–6660.

Roshandeh, K. P., Noori, M., Ardakani, M., and Tellambura, C. (2017). Distributed storage allocation for multi-class data. In *Proc. of IEEE ISIT 2017*, Aachen, Germany.

Ruiz-Alvarez, A. and Humphrey, M. (2012). A model and decision procedure for data storage in cloud computing. In *Proc. of IEEE/ACM CCGrid 2012*, Ottawa, Canada.

Russo Russo, G., Ferrarelli, D., Pasquali, D., Cardellini, V., and Lo Presti, F. (2024). QoS-aware offloading policies for serverless functions in the Cloud-to-Edge continuum. *FGCS*, 156:1–15.

Taha, A., Manzoor, S., and Suri, N. (2017). SLA-based service selection for multi-cloud environments. In *Proc. of IEEE EDGE 2017*, Honolulu, HI, USA.

Wendell, P., Jiang, J. W., Freedman, M. J., and Rexford, J. (2010). DONAR: Decentralized server selection for cloud services. In *Proc. of ACM SIGCOMM 2010*, New Delhi, India.

Wu, C., Chen, Y., Qi, Z., and Guan, H. (2022). DSPR: Secure decentralized storage with proof-of-replication for edge devices. *JSA*, 125:102441.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS