Multi-objective Optimization of Data Placement in a Storage-as-a-Service Federated Cloud

Amina Chikhaoui, Laurent Lemarchand, Kamel Boukhalfa, Jalil Boukhobza

Multi-objective Optimization of Data Placement in a Storage-as-a-Service Federated Cloud

Amina Chikhaoui, Laurent Lemarchand, Jalil Boukhobza, Kamel Boukhalfa

Multi-objective Optimization of Data Placement in a Storage-as-a-Service Federated Cloud

AMINA CHIKHAOUI[1,2,3], LAURENT LEMARCHAND[1], JALIL BOUKHOBZA[1], and KAMEL BOUKHALFA[2],

[1]Univ Brest, Lab-STICC, CNRS, UMR 6285, F-29200 Brest, France, France,

[2]University of Science and Technology Houari Boumediene, Algeria, and

[3]Ecole Normale Supérieure, Kouba, Algeria

Cloud federation enables service providers to collaborate to provide better services to customers. For cloud storage services, I/O performance and network latency are important to ensure end-to-end QoS. Therefore, effectively placing customer objects for a cloud that is a member of a federation is a real challenge. In order to optimize data placement, storage, migration and latency costs need to be considered. These costs are contradictory in some cases. In this paper, we modeled object placement as a multi-objective optimization problem. The proposed model takes into account parameters related to the local infrastructure, the federated environment, customer workloads and their SLAs. For resolving this problem, we propose CDP-NSGAII$_{IR}$, a Constraint Data Placement matheuristic based on NSGAII with injection and repair functions. The objective of the injection function is to enhance the solutions' quality. It consists to calculate some solutions using an exact method then inject them into the initial population of NSGAII. The repair function was designed to ensure that the solutions obey the problem constraints in term of storage and so prevents from exploring large sets of infeasible solutions. It allows to reduce the execution time of NSGAII. Our experimental results show that, the injection function improves the HV of NSGAII and the exact method by up to 94% and 60% respectively while the repair function reduces the execution time by an average of 68%.

CCS Concepts: • **Federated cloud** → **Hybrid storage system**; • **Data placement** → *Cost optimization*.

Additional Key Words and Phrases: Data placement, optimization, cloud, cloud federation, NSGAII

ACM Reference Format:
Amina chikhaoui[1,2,3], Laurent Lemarchand[1], Jalil Boukhobza[1], and Kamel Boukhalfa[2]. 2020. Multi-objective Optimization of Data Placement in a Storage-as-a-Service Federated Cloud. 1, 1 (November 2020), 32 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Cloud computing [67] is the most popular distributed computing paradigm [31]. It provides IT resources to users under the form of ready-to-use services. Storage-as-a-Service (STaaS) is one of the most important services [32, 51] offered by Cloud Service Providers (CSPs) to customers to manage the storage of their data. For STaaS, the storage I/O performance is a primary QoS indicator. In fact, I/Os take about 90% of a transaction execution time [65].

Authors' address: Amina chikhaoui[1,2,3], achikhaoui@usthb.dz; Laurent Lemarchand[1], laurent.lemarchand@univ-brest.fr; Jalil Boukhobza[1], boukhobza@univ-brest.fr; Kamel Boukhalfa[2], k.boukhalfa@usthb.dz,
[1]Univ Brest, Lab-STICC, CNRS, UMR 6285, F-29200 Brest, France, Brest, France,
[2]University of Science and Technology Houari Boumediene, Algiers, Algeria,
[3]Ecole Normale Supérieure, Kouba, Algiers, Algeria.

Cloud federation [12, 52] may constitute a key solution for such an issue. It consists in making several CSPs cooperate by sharing resources. Thus, CSPs can insource and outsource their customers' data and services to offer them a better QoS and at the same time to increase their own revenue (or reduce costs, for instance, by deploying less hardware resources). Cloud federation may give the opportunity to small and medium size CSPs to become very competitive [19].

Providing a storage service with the required I/O performance and network latency at a minimum cost is the main objective of CSPs. To handle the I/O bottleneck, cloud infrastructures host different storage classes. These are mechanical Hard Disk Drive (HDD) and Solid State Disks (Flash-based SSD and 3D XPoint memories, such as Intel Optane memory)[18, 45]. On the other hand, to manage network latency, CSPs try to dynamically bring data close to their users. This helps to reduce the latency experienced by the users and lower the overall traffic among data-centers [82].

In the context of a Cloud federation, a CSP member tries to leverage both local and the other CSPs storage services to place customers data in order to reduce costs and at the same time meet the required QoS. This is a complex problem because of the heterogeneity of storage and network resources on one hand, and because of the variable federation environment and customers workloads and their respective SLAs on the other hand.

The global data placement cost is roughly composed of the storage, the migration and the network latency costs. These are interdependent and in some cases contradictory costs. For instance, placing data locally may not be expensive in terms of storage and migration costs but may generate a high latency for customers located in far regions. So, trade-offs are inevitable for this type of data placement problem.

The problem of data placement has been widely investigated in the Cloud environment. The proposed techniques have considered many issues related to the hybrid storage system of a centralized cloud as in [17, 35, 41, 57, 62, 78, 81] and distributed clouds like [30, 43, 46, 50, 72, 76, 79, 83]. State-of-the-art studies have considered different objectives under different constraints. Mono and multi-optimization techniques have been proposed to reach the desired objective(s). Existing work on data placement in centralized cloud cannot be applied in distributed environment because the network factor was not considered. On the other hand, we noticed that the storage cost in existing work is mainly based on volume in terms of GB and I/O performance was not always considered. Furthermore, the storage-related SLA metrics were seldom considered especially when it comes to a distributed cloud.

In this paper, we address the data placement in Cloud federation by modeling it as a multi-objective constrained optimization problem. The storage, migration and latency costs constitute the different objective functions which are subject to different constraints. The heterogeneity of the local and the federated storage classes and services, the customers workloads and the requested QoS have been taken into account.

The data placement problem in this work follows a MAPE-K (Monitor, Analyse, Plan, Execute, and Knowledge) autonomic feedback loop model [36]. We focus especially on the Analyse and Plan steps of this model by formulating and resolving the placement problem.

In multi-objective optimization (MOO), it is rarely the case that it exists a single solution that simultaneously optimizes all the objective functions [22]. In fact, there is a set of solutions, called non-dominated solutions or Pareto set. These solutions define the trade-offs between the different objectives. The notion of non-dominance is related to the fact that no solution is better than the others for all objective functions. (see Section 4.1).

Our goal is to efficiently obtain a set of non-dominated solutions that simultaneously minimize the storage, the network latency and the migration costs. The selection of a final placement solution from the Pareto set is out of the scope of this paper. It is a subjective task and might be based on other factors and parameters not considered in the optimization process [53].

Due to its popularity and efficiency, we used NSGAII [26] meta-heuristic to solve this multi-objective problem. However, being an evolutionary meta-heuristic, NSGAII suffers from solutions unfeasability caused by the presence of constraints, especially when the problem size increases. Furthermore, NSGAII lacks good initial population [63] due to the random initialization. To deal with such problems, we propose an upgrade of NSGAII into a matheuristic [73] by combining it with an exact method. We called this matheuristic CDP-NSGAII$_{IR}$ for Constraint Data Placement matheuristic based on Injection and repair functions.

CDP-NSGAII$_{IR}$ works as follows. First, it generates a small number of non-dominated solutions by using an exact method. This exact method transforms the MOO problem into a single objective optimisation problem (SOO) by aggregating and weighting the objective functions into one linear function. We used CPLEX [1] to solve the SOO problem. Then, the obtained solutions are injected into the initial population of NSGAII in order to populate the initial generation with some good intividuals. Furthermore, we repair all the individuals in each iteration so as to transform unfeasible solutions into feasible ones. The injection function helps NSGAII to improve its solutions quality as NSGAII suffers from a bad initial population which is randomly generated while the repair function helps NSGAII to perform swiftly because with reparation we do not need to check the constraints for the individuals no more as they are already repaired.

In short, the main contributions of this paper are twofold.

- Formulating the data placement problem as a multi-objective integer linear program. The proposed model takes into account the local and the external storage systems characteristics, the customers SLAs and their workloads.
- Upgrading NSGAII to a matheurstic (CDP-NSGAII$_{IR}$) by merging it with an exact method. Different tools and methods have been used to design and implement the placement problem. CDP-NSGAII$_{IR}$ consists in the following steps:
  - A pre-processing step: Calculation of a set of solutions using CPLEX solver.
  - An evolutionary step: First, the set of solutions previously obtained is injected to the initial population of NSGAII. Then, the evolutionary process of NSGAII is executed with a reparing function for the individuals in each iteration.

The evaluation results show that CDP-NSGAII$_{IR}$ reaches quickly the optimal Pareto front for small problem instances with only the repair function. For example, it took 0.3 second for a problem composed of 15 objects and 4 locations while an exact method took about 48 minutes. For large instances, the injection function improves the Pareto front quality of both NSGAII meta-heuristic and the exact method while the repair function enhances the execution time of NSGAII. In fact, the injection function improves the HV of NSGAII and the exact method by up to 94% and 60% respectively and the repair function reduces the execution time of NSGAII by 68%.

The rest of this paper is organized as follows. first, some preliminaries on cloud federation and MOO are given in Section 2. Then, we define the system model and formulate the problem of data placement in Section 3. Section 4 describes the contribution. Experimental results are described in Section 5. Related work are summarized in Section 6 and at last, conclusions and perspectives are given in Section 7.

2 BACKGROUND

This section gives some preliminary background about the concepts used in this paper. First the cloud federation paradigm is discussed, then, some background knowledge related to the multi-objective optimization field is given.

2.1 Cloud federation

Cloud federation refers to a set of geographically distributed CSPs that interact with each other by sharing their resources in a cooperative manner via centralized and decentralized marketplaces [12, 60, 70, 71]. This collaboration helps the CSPs to increase their profit while providing a continuous provisioning of high-quality services by exploiting temporal and spatial availability of resources and diversity of operational costs [44]. Cloud federation appeared to overcome some limitations [13, 29, 70] such as resource contention, service interruption and Quality of Service (QoS) degradation. These limitations appeared with the massive deployment of Cloud infrastructures and the technology maturity of cloud computing [12] and the emergence of many related new paradigms such as mobile cloud computing, IoT, and big data applications. To work properly and to maintain the integrity of the organization, the federated CSPs are governed by a Federation Level Agreement (FLA) that specifies interconnection rules and describes each participant's responsibilities and permissible behaviors, along with the financial, administrative, or other penalties for violating its terms [39].

In this section, we first discuss the motivations behind the adoption of federated clouds. Then, a classification of the different federation forms is presented to help positioning our work.

2.1.1 Motivations for the cloud federation. Cooperation between CSPs creates benefits. We cite, among others, the following reasons that motivate the utilization of a cloud federation [12, 60]:

Elasticity and availability: Elasticity is one of the main cloud computing properties. It allows to dynamically adapt resource provisioning by increasing or decreasing their amount according to customers workloads [86]. Cloud federation enables providers to efficiently adjust their hosting capacity through cooperation with others instead of provisioning (new) extra capacities which increases energy waste and costs. Furthermore, the geographic distribution of federated cloud infrastructures makes it possible to migrate services from areas that may be affected by outages. It allows also for data to be geographically replicated which maintains the availability of customer services.

Cost and performance efficiency: Cloud federation allows CSPs to expand their geographic coverage without the need to build new data-centers [59]. This helps to save costs and at the same time improves the offered QoS. Also, the latency can be minimized by satisfying requests from a closer provider and the response time can be reduced by executing the request on a more powerful host from other providers. Furthermore, with cloud federation the load can be balanced between the involved clouds according to some metrics such as energy consumption.

Economic Barriers: cloud infrastructures were not designed to inter-operate, therefore there is a lack of standardization and compatibility between the underlying technologies. This generates the "vendor lock-in" phenomenon where customers stay confined to certain CSPs. This vendor lock-in may lead to economic and functional losses for customers because migrating their services to other clouds is expensive and requires a large deal of technical effort. In a cloud federation, the FLA can be used to specify interfaces and data representation protocols supported on the environment, and then interested parties can migrate among federation providers if needed [12].

2.1.2 Taxonomy of cloud federation. Two main classifications of cloud federation exist as is seen in Figure 1:

According to the interaction level: There are vertical, horizontal and hybrid federations [12, 69]. In a vertical federation, the interconnection, occurs between clouds at different levels of services, while in a horizontal federation the interconnection occurs between clouds at the same service level. Finally, hybrid federations perform horizontal and vertical expansion in accordance with the interest of both customers and providers [13].

According to membership: There are three forms of federations [13, 13, 25, 70]:

(i) ad-hoc federation: A CSP or a broker may use resources of other public CSPs such as Amazon AWS and Google apps.

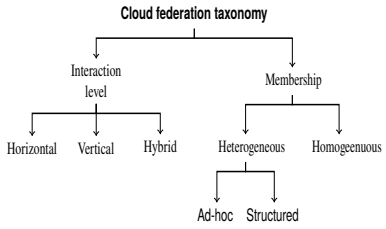

Fig. 1. Cloud federations taxonomy

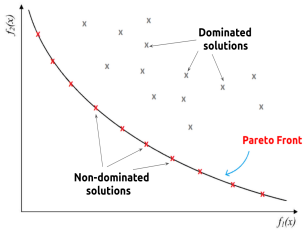

Fig. 2. Pareto front

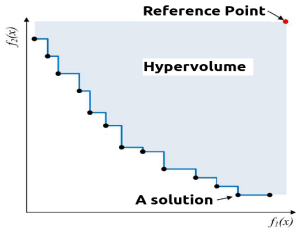

Fig. 3. Hypervolume indicator

In this type of federation, public clouds are not necessarily aware that they participate in a federation. Consequently, the CSP who uses resources from these external CSPs has to establish a specific SLA (Service Level Agreement) with each one of them.

(ii) Structured federation: this federation is represented by small and medium private CSPs (companies which handle their own resources as a Cloud infrastructure) that collaborate with each other to maintain all properties of the cloud paradigm. The CSPs are governed by a Federation Level Agreements (FLA) [1, 17, 25] which describes the relation between the different CSPs through functional and nonfunctional properties. This can be interesting and useful in many critical environments. In fact, FLA determines the behavior of heterogeneous and autonomous clouds [1]. Therfore, it alleviates the complications between the federated clouds because resource sharing is well-behaved, obeying to this contract [12].

(iii) Internal federation: here the cloud federation is composed of different geographically distributed data-centres belonging to the same institution. In this class there are no complex agreements to maintain between the clouds because they are managed by the same entity.

In this work, we suppose that we have a structured and horizontal federation where an FLA governes the federation CSPs and each one of them devotes a part of his resources to his partners (it can also be adapted to internal federations). We are specifically interested in the object placement problem of a CSP member of a horizontal structured federation. We assume that each CSP of the federation devotes a part of his resources to the federation. The concerned CSP can use his own resources or that of the other federation members for the placement of his customers data objects. The main objective of this work is to provide the decision-maker with the different placement trade-offs that satisfy the customers requirements and minimize the placement costs.

2.2 Multi-objective optimization

Multiobjective optimization is a very important research topic as the nature of most problems is multi-objective in nature. Unlike single-objective optimization (SOO), the purpose of MOO is to find a set of good solutions [55] representing trade-offs between objectives since a single solution optimizing all of the objectives simultaneously rarely exists [42]. A good solution is often defined by the *dominance* concept (see Section 2.2.2). A decision maker has to choose the final solution among the good ones [54], by eventually considering factors not included in the optimization process because they can be hardly modeled and are sometimes subjective.

2.2.1 Multi-Objective Optimization Problem. In MOO, an optimization problem over n decision variables can be defined by the optimization (minimization or maximization, minimization in the sequel) of a given objective function vector (of k objectives) and satisfaction of a set of m constraints. The constraints define the space of feasible solutions [55].

$$\begin{array}{ll} \underset{x \in X}{\text{minimize}} & f(x) = [f_1(x), ...f_k(x)] \\ \text{subject to} & h_j(x) \leqslant 0, \qquad\qquad j = 1, ..., m \\ & x = [x_1, ...x_n] \in X \qquad f(x) = [f_1, ...f_k] \in Y \end{array}$$

wherein x is the decision (solution) vector, X the decision space, f the objective vector and Y the objective space.

2.2.2 Pareto-optimality. The Pareto optimality [23] concept is a way of defining good solutions when more than one objective is considered. A Pareto optimal solution is one that is not dominated by another solution from the feasible space, i.e. there exists no solution that is better for all considered goals.

More formally, the Pareto dominance relation is defined as follows:

Definition 1: [23] A vector $v = (v_1, v_2, ..., v_k)$ dominates another vector $w = (w_1, w_2, ..., w_k)$ if and only if $\forall i \in \{1, 2, ..., k\}, v_i \leqslant w_i$ and $\exists j \in \{1, 2, ..., k\}, v_j < w_j$. This is denoted by $v \prec w$.

Definition 2: the Pareto optimal set is defined by: $P = \{x \in X, \nexists x' \in X, F(x') \prec F(x)\}$

Definition 3: the Pareto front is defined by: $PF = \{F(x), x \in P\}$

Definition 4: the Pareto set approximation is a set of approximate solutions where no element of this set is dominated by any other.

Figure 2 illustrates a Pareto front of a MOO problem where two objective functions $f_1(x)$ and $f_2(x)$ are being minimized. Objective values in red color correspond to non-dominated solutions while those in gray color are dominated ones.

2.2.3 Multiobjective evolutionary algorithms (MOEAs). Solving a MOO problem needs to go through two steps (i) finding the Pareto set, (ii) selecting a solution among the Pareto set through decision making process. In this work, we are interested in the first step.

MOEAs are well-known non-deterministic population-based algorithms proposed and used to solve MOO problems. They are generally based on the Darwinian natural selection theory. MOEAs help the decision makers to find a Pareto set within a reasonable time in case where the exact methods are not applicable. This is generally the case for large-scale problems due to the prohibitive execution time that their solving requires.

Standard MOEAs perform sometimes poorly for difficult problems with complex feasible spaces or a high number of constraints [64]. Repair heuristics, solution penalization methods and special representations and operators are the classical techniques for constraint handling in evolutionary algorithms. Designing special representations and operators can be helpful but is also time consuming and does not take benefit of previous work. Penalization methods are the oldest approach used to handle constraints in optimization algorithms including MOEAs [21]. They consist to transform a constrained-optimization problem into an unconstrained one by adding/subtracting a certain value to/from the objective function based on the amount of constraint violation present in a certain solution [21]. However, they do not guarantee to find feasible solutions. Repair operators consist to reduce the search space to feasible solutions by making some local search in order to transform unfeasible solutions to feasible ones. The repair operators avoid repetitive and costly evaluation of unfeasible solutions maintained by penalization methods and they handle directly the problem constraints[64]. Also, a repair operator allows to use all the well-known and efficient standard operators.

Three goals must be considered when solving a MOO problem. These are [61, 87] (i) *precision:* this means that the found non-dominated solutions should be as close as possible to the Pareto-optimal front, (iii) exhaustivity: the solutions must be numerous and (iii) *diversity:* as the found solutions should be well spaced over the Pareto front.

2.2.4 Performance indicators. In single-objective minimization, the quality of a given solution is trivial to quantify, however, evaluating the quality of an approximation of a Pareto set is a non-trivial task [14].

To quantify and compare the quality of solution sets produced by different MOEAs, numerous performance indicators have been proposed in the literature. These performance indicators consist in assigning scores to Pareto front approximations [14, 15]. We quote, among others, *Generational Distance*, *Inverted Generational Distance*, *Hypervolume*, *Maximum Pareto Front Error* and *Spacing*. These indicators allow to compare Pareto fronts, in term of precision, exhaustivity and diversity or any combination. In our evaluations, we used the *Hypervolume* indicator.

Hypervolume (HV), also called hyperarea, is one of the most popular measures for the performance assessment of multi-objective algorithms [77]. It encompasses the exhaustivity, the precision and the diversity metrics [14, 15]. It computes the portion of the objective space dominated by all the solutions bounded by a reference point r. The HV of an approximate Pareto front PF_{approx} is obtained by $\lambda_k(\bigcup_{z \in PF_{approx}} [z; r])$ where λ_k is the k-dimensional Lebesgue measure [14]. The greater the HV value, the better the approximation is. An illustration is given in Figure 3 for the bi-objective case ($k = 2$).

3 SYSTEM MODEL AND PROBLEM FORMULATION

The placement of customers data objects for a CSP member of a federation is a hard problem [48, 75]. In order to build a smart placement solution, we first need to formulate it under the form of an optimization problem. The objective function of this problem seeks the minimization of the placement cost. This work is based on the cost model proposed in [20] and built for data object placement in a federated cloud. Furthermore, object placement must be realized while respecting a set of constraints related to the system model.

Notation used in this paper is summerized in Table 1.

In this section, we start by briefly discussing the challenges and objectives of this study. Then, we describe the target system. After that, the used cost model is shortly introduced. Finally, we end up by formulating the placement problem.

3.1 Challenges and objectives

The main objective in this work is to assist the cloud administrator of a CSP in order to find a good placement configuration of the customers data objects. The local storage infrastructure and other storage services from other CSPs are used to manage the customers data objects. The I/O storage performance and the network latency are the targeted QoS metrics for the customers. The sought placement configuration should respect the customers QoS requirements and at the same time minimize the CSP costs.

Minimizing the placement cost is contradictory with satisfying the customers requirements. On the other hand, the satisfaction of both I/O storage performance and network latency may be contradictory in certain circumstances. For example, placing a customer object in one location may respect the requested I/O storage performance but may generate a high network latency. In addition, the placement configuration must adapt to the dynamic and heterogeneous environment. In fact the used local and external resources have different characteristics in term of capacity, performance and prices. Also, the CSPs of the federation may update their storage resource prices according to the availability of their resources. Furthermore, the customers have different workloads which may change over time. Customers may be mobile or have users geographically spread, which impacts the network latency to access their objects. Dealing with such a **dynamic** and **heterogeneous** system makes data placement strategies very challenging and a thorough formulation of the placement

Table 1. Notation Table

Symbol	Meaning
M, Z_m	The number of geographic zones, the zone m
D	the number of clouds in the federation
CSP_d, ss_d,css_d, io_d	cloud provider d, its storage service, the capacity (GB) of the storage service and its IOPS performance
J	The number of local storage classes
sc_j, csc_j, $io_j(op)$	The storage class j, its capacity and its IOPS per operation type op
op	($op \in \{rr, rw, sr, sw\}$) where rr: random read, rw:random write, sr:sequential read, sw:sequential write
R, r	The number of request types, the request type r (e.g. get, put)
K	The number of customers
U, u_k, O_k	The set of customers, customer k and the set of his objects
$ioHard_k$,$ioSoft_k$	the hard iops SLA and the soft iops SLA of customer u_k
$ioOffered_k$	The IOPS delivered to the customer u_k
$o_{i,k}$,$s_{o_{i,k}}$, $io_{o_{i,k}}(op)$	the object i of customer u_k, its size and its issued I/O requests per operation type
P	the number of possible storage locations, $P = D + J - 1$
$occ_k^{m,r}$	the number of requests of type r issued by the customer u_k from the zon Z_m
$acc_k^{r,o_{i,k}}$	The fraction requests of type r issued by the customer u_k and accesses the object $o_{i,k}$
$tot_k^{m,o_{i,k}}$	the total number of requests of customer u_k issued from the zone of Z_m and that access to $o_{i,k}$
$l_{Z_m,Zm'}$	the network latency between the zone Z and the zone Z'.
T	Period of time
x	Decision variable representing a placement configuration
Objective functions	
$store$	The storage cost for a given x placement configuration
$migrate$	The migration cost for a given x placement configuration
$latency$	The latency cost for a given x placement configuration
Costs	
$Cost_{plc,T}$	The placement cost of internal customers data objects
$Cost_{lcl,T}$	The local placement cost
$Cost_{lstg,T}$	The local storage cost
$Cost_{lmgr,T}$	The local migration cost
$Cost_{outsrc,T}$	The outsourcing cost
$Cost_{estr,T}$	The external storage cost
$Cost_{gmgr,T}$	The geo-migration cost
$Cost_{bmgr,T}$	The back-migration cost
$Cost_{pnt,T}$	The penalty cost
$Cost_{pnt_{io},T}$	The storage performance penalty cost
$Cost_{pnt_{ltc},T}$	The network latency penalty cost
$Cost_{pnt_{mgr},T}$	The migration penalty cost

problem must take into account most of the system related factors in order to make adequate decisions for placing data objects.

3.2 System overview

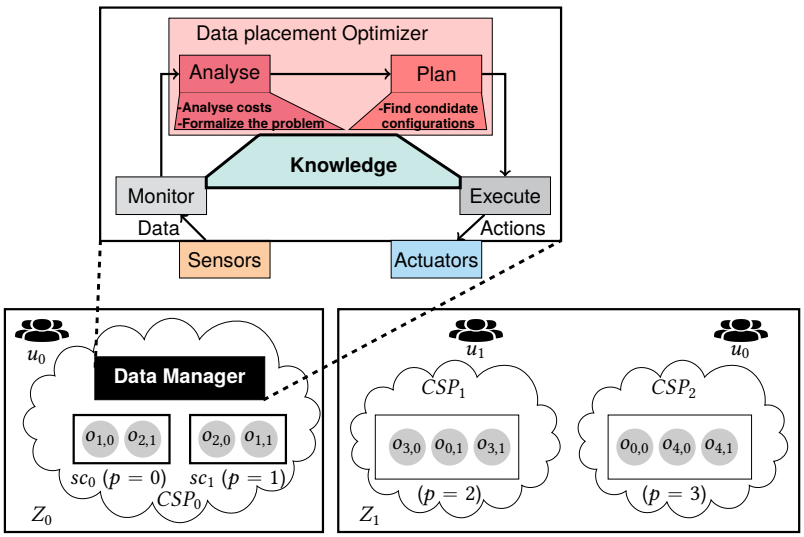

Fig. 4. System architecture

In this work, we focus on the data objects placement from the stand point of a single CSP. Indeed, in a Cloud federation, each CSP is free to apply his own optimization techniques. We suppose that the CSPs use interfaces and tools like OCCI standard interface [27] or apache LibCloud [6] in order to unify the interface and facilitate migrating services among clouds.

Let CSP_h be the considered provider that wants to optimize the placement of data objects of his customers (CSP_0 in the example of Figure 4).

The data placement optimization process is performed by the Data Manager component. It incorporates the different steps of a MAPE-K model which is described below.

First, The *Monitor* step consists to perform a periodic monitoring of the federated environment by collecting and aggregating the monitored parameters such as the available storage resources of the CSPs and their prices, the customers locations and the latency of their requests, the objects current placement, etc. Monitoring tools, such as Amazon CloudWatch [5] or OpenStack Watcher [7], may be used in this step. Also, each CSP may share part of his own monitoring data to the federation.

Then, the monitoring data are passed to the **Data Placement Optimizer** module. This module is composed of the *Analyze* and *Plan* steps of the MAPE-K process. The Analyze step consists of two phases. In the first one, the monitored metrics are introduced in a cost model in order to calculate the costs related to the data placement. The second phase uses the monitored metrics in addition to the results of the first phase in order to build and formalize an optimization problem. The *Plan* step incorporates some methods that aim to find the data placement configurations that achieve a trade-off between some given criteria. Finally, based on the results obtained by the Data Placement Optimizer and some non-technical aspects such as the CSP reputation, in the *Execute* step, a placement scheme is selected and the customers objects placement are autonomously re-configured.

In this work, we focused on the **Data Placement Optimizer** module.

Table 2. Network latency price and penalty per request. ($b_0 > b_i > b_{i+1} > 0$)

$ltc_{offered}$	Price (%)
$[0..B_1]$	b_0
$[B_i..B_{i+1}]$	b_i
$> B_n$	0

(a)

$ltc_{offered}$	Penalty (%)
$[0..B_1]$	0
$[B_i..B_{i+1}]$	$b_0 - b_i$
$> B_n$	b_0

(b)

3.3 System model

To illustrate our problem, we describe, in this section, the targeted system and the hypotheses of our work.

We consider a cloud federation composed of D CSPs, spread over M different geographical zones with ($D \geqslant M$). There may be one or more CSPs in each zone with different characteristics.

For instance, in Figure 4, we have 3 CSPs spread over 2 zones Z_0 and Z_1.

We suppose that CSP_h accommodates J different storage classes. Each storage class sc_j (e.g. $J = 2$ in Figure 4, HDD for sc_0 and SSD for sc_1) is characterized by its capacity csc_j, its iops according to the type of I/O operation issued $io_j(op)$. We considered the following I/O operations as they proved to perform differently according to state-of-the-art work: sequential read (sr), sequential write (sw), random read (rr) and random write (rw). Each storage class has also other characteristics such as the price and the guarantee period.

We assume that CSP_h can use storage services offered by the other CSPs of the federation. A storage service ss_d of a distant partner CSP_d (for instance CSP_1 and CSP_2 in Figure 4) is defined by the storage capacity css_d and the storage performance io_d. The storage services can be billed differently by partner CSPs.

The provider CSP_h handles a set U of K internal customers. Customers may be located in different geographical zones. They may be mobile or have end-users geographically spread. For example, in Figure 4, there are two customers u_0 and u_1. Customer u_0 end users are spread on the two zones Z_0 and Z_1 while end users of customer u_1 are located in zone Z_1. Each customer u_k has a data application composed of a set of objects O_k. It generates a workload made up of several types of requests such as get, put and update. The workload generates a set of different I/O operations (that can be captured with tracing tools as in [56]). Each object i of a customer k, denoted $o_{i,k}$, has a size $s_{o_{i,k}}$ and is accessed through different I/O requests per operation type $io_{o_{i,k}}(op)$.

The customer also requests a QoS (SLA) in terms of number of IOPS. The IOPS SLA is expressed by a hard SLA $ioHard_k$ which must be respected and a soft SLA $ioSoft_k$. This is a proportional penalty model related to the degree and the delay of the SLA storage performance violation [16]. Note that $ioSoft_k$ represents the agreed upon I/O throughput of customer u_k while $ioHard_k$ is the limit of I/O throughput degradation tolerated by u_k. $ioOffered_k$ represents the IOPS effectively delivered to the customer u_k. If $ioOffered_k$ is lower than the soft SLA, CSP_h undergoes a storage IOPS related penalty.

We suppose, as in [68], that the network performance of the queries is billed according to the experienced latency as in Table 2a. It defines intervals of network latency QoS values and the corresponding prices. [β_i,β_{i+1}] represents the interval of the network latency and b_i is the price to pay by the customer for the network latency corresponding to this interval. b_0 indicates the (highest) price of the lowest latency interval. If the network latency exceeds a certain limit, its price becomes zero (the higher the latency, the lower the price). So, If the offered latency is within the interval [β_i,β_{i+1}], CSP_h loses an amount corresponding to $b_0 - b_i$ as shown in Table 2b. We denote by $penalty(l_{Z_m,Z_{m'}})$ the latency penalty between the zones Z_m and $Z_{m'}$, $l_{Z_m,Z_{m'}}$ being the latency between the two zones. For simplicity, $l_{Z_m,Z_m} = 0$ as the network latency in a given zone is small.

We suppose that the cloud administrator periodically makes decisions about objects placement. T is the time period during which monitoring is executed to extract objects I/O patterns for making such a placement decision. One example of a framework that considers such periodic changes is the OpenStack Watcher project [7].

As already discussed, Cloud customers are mobile or have many users geographically spread. So, the locations from which requests originate may change over time.

We want to model the overall workload applied by a customer u_k (its users) issued from the different zones in order to calculate the network latency penalty. For that, we use the matrix occ_k which elements $occ_k^{m,r}$ represent the number of requests of type r issued by the customer u_k from the zone Z_m. As well, let the matrix acc_k represent the access of the requests to the objects. Each element $acc_k^{r,o_{i,k}}$ of this matrix indicates the fraction of requests of type r issued by the customer u_k and that access the object $o_{i,k}$. $acc_k^{r,o_{i,k}} \in [0..1]$ such as for each request type r, $\sum_{o_{i,k} \in o_k} acc_k^{r,o_{i,k}} = 1$, see example below.

We denote the overall workload by the matrix tot_k. It is the product of the matrix occ_k and the matrix acc_k, see Eq. (1)

$$tot_k = occ_k * acc_k \tag{1}$$

So, each element $tot_k^{m,o_{i,k}}$ represents the overall number of all requests types issued by the customer u_k from the zone Z_m and that access the object $o_{i,k}$.

Example : Let us take the same federation configuration depicted in Figure 4. Let customer u_0 issue 2 types of requests (get, and update). Its end-users are spread over the 2 zones. Matrix occ_0 represents the number of requests issued by the users of customer u_0 from different zones.

$$occ_0 = \begin{array}{c} \\ Z_0 \\ Z_1 \end{array} \begin{array}{c} \begin{array}{cc} get & update \end{array} \\ \begin{pmatrix} 5500 & 1000 \\ 2000 & 100 \end{pmatrix} \end{array}$$

Matrix acc_0 represents the access fraction of these requests per object.

$$acc_0 = \begin{array}{c} \\ get \\ update \end{array} \begin{array}{c} \begin{array}{ccccc} o_{0,0} & o_{1,0} & o_{2,0} & o_{3,0} & o_{4,0} \end{array} \\ \begin{pmatrix} 0.6 & 0.1 & 0.2 & 0 & 0.1 \\ 0.5 & 0 & 0.2 & 0.1 & 0.2 \end{pmatrix} \end{array}$$

So, the resulting matrix tot_0 represents the total requests issued from the location of each zone and accessing to the different objects and is given as follows:

$$tot_0 = \begin{array}{c} \\ Z_0 \\ Z_1 \end{array} \begin{array}{c} \begin{array}{ccccc} o_{0,0} & o_{1,0} & o_{2,0} & o_{3,0} & o_{4,0} \end{array} \\ \begin{pmatrix} 3800 & 550 & 1300 & 100 & 750 \\ 1250 & 200 & 420 & 10 & 220 \end{pmatrix} \end{array}$$

The element $tot_0^{1,o_{0,0}} = 1250$ represents the total number of requests issued from the zone Z_1 by the end-users of the customer u_0 that access to the object $o_{0,0}$.

3.4 Cost model

A cost model was proposed in [20] for object data placement in a federated cloud. The overall placement cost for data objects of internal customers in a time period T consists of a local placement cost $Cost_{lcl,T}$, an outsourcing cost $Cost_{outsrc,T}$, a back-migration cost $Cost_{bck\text{-}mgr,T}$ and a penalty cost $Cost_{pnt,T}$ as shown in Eq. (2). Note that is in this paper, every time a cost is discussed, it is related to a monetary cost.

$$Cost_{plc,T} = Cost_{lcl,T} + Cost_{outsrc,T} + Cost_{bmgr,T} + Cost_{pnt,T} \tag{2}$$

$$Cost_{lcl,T} = Cost_{lstg,T} + Cost_{lmgr,T} \tag{3}$$

$$Cost_{outsrc,T} = Cost_{gmgr,T} + Cost_{estr,T} \tag{4}$$

$$Cost_{pnt,T} = Cost_{pnt_{io},T} + Cost_{pnt_{ltc},T} \tag{5}$$

We briefly describe the different components here-under.

Local placement cost: The local placement cost $Cost_{lcl,T}$ is the storage cost of internal customers objects in the local infrastructure. It consists of a local storage cost $Cost_{lstg,T}$ and a migration cost $Cost_{lmgr,T}$ as shown in Eq. (3). $Cost_{lstg,T}$ is composed of the occupation cost in terms of purchase cost and guarantee, and the I/O workload execution cost in terms of energy and wear-out. $Cost_{lmgr,T}$ is related to the cost of local migration between storage classes within the CSP infrastructure (energy and wear-out).

Outsourcing cost: This cost ($Cost_{outsrc,T}$) is related to the placement of internal customers objects in partner CSPs. It is composed of geo-migration cost $Cost_{gmgr,T}$ of customers objects, and the storage cost fixed by the partner CSP $Cost_{estr,T}$ (see Eq. (4)).

Back-migration cost: It ($Cost_{bmgr,T}$) represents the cost of bringing back the previously outsourced objects to the home infrastructure.

Penalty cost: $Cost_{pnt,T}$ is composed of the I/O storage performance penalty cost $Cost_{pnt_{io},T}$ related to the violation of I/O SLA term and the network latency penalty cost $Cost_{pnt_{ltc},T}$ related to the degradation of the delivered network latency as noted in Eq. (5). As previously discussed, the CSP undergoes an I/O penalty when $ioOffered_k$ are less than the $ioSoft_k$ for any customer u_k. Concerning the network latency penalty, it is related to the number of issued requests tot_k (see Eq. (1)) by each customer u_k and the latency level between their origin zones and that of the requested objects.

3.5 Problem formulation

Our objective is to find a good placement configuration for customers' objects of a CSP using opportunistically both its internal storage classes and the storage services of partner CSPs. The sought placement configuration should respect the customers QoS requirements and at the same time minimize CSP costs.

When we analyze the placement cost (see Section 3.4), we find that it is composed of three main components which are: *(1) the storage cost* related to the occupation and I/O workload costs, *(2) the migration cost* which is defined by the different migrations in the system, and *(3) the network latency cost* that is the penalty generated by the deterioration of the QoS caused by the network latency. These costs are in some cases contradictory. For instance, placing data locally may be cheap in term of storage and migration costs but may generate a bad network latency for customers located in other

regions. Another example is that there exists a location that may minimize both storage and latency costs but migration to this location generates a prohibitive cost. So, making a trade-off while placing data objects is inevitable.

In order to give more flexibility to the cloud administrator to update the placement configuration of customers objects, we formulate the placement problem under the form of a multi-objective optimization problem taking into account the three functions namely (1) storage cost, (2) migration cost, and (3) latency cost, and constraints related to the limited local and external resources and customers SLAs.

In this section, we will, first, specify how a solution representing a placement configuration is encoded. Then, we will define its cost functions and show how to evaluate them. Finally the constraints to meet will be described.

3.5.1 Individual representation. In our work, a solution for the placement of customers objects is represented by means of a classical integer encoding schema. With such an encoding, each individual, named also a chromosome, is defined as an integer array which represents a placement configuration of all the objects of all the customers in P locations. So, a chromosome is a set of genes. A gene number i in a chromosome corresponds to the object number i. The value of a gene i corresponds to the location of object i. As explained already, from a CSP perspective, the overall number of locations P equals to the number of the local storage classes plus the number of partner CSPs ($P = D + J - 1$).

So a placement configuration is represented by the vector $x = (x_{o_{0,0}}, x_{o_{1,0}}, \cdots, x_{o_{i,k}} \cdots)$ where $x_{o_{i,k}}$ is a potential location for the object $o_{i,k}$.

For simplification, in our representation, a location $p \in [0, J[$ corresponds to local storage classes while $p \in [J, D+J-1[$ corresponds to a location in a partner cloud.

In the example depicted in Figure 4. CSP_0 is the concerned one by the optimization placement. It accommodates 2 storage classes, then $p = 0$ and $p = 1$ correspond to the local placement. $p = 2$ corresponds to a placement in CSP_1 while $p = 3$ represents a placement in CSP_2.

The particular assignment sketched in Figure 4 corresponds to the chromosome $x = (x_{0,0}, ..., x_{4,0}, x_{0,1}, ..., x_{4,1})$ represented by the vector $x = (3, 0, 1, 2, 3, 2, 1, 0, 2, 3)$. The chromosome length indicates the numbers of all customers' objects. In this example we have 2 customers each one has 5 objects which means, the length of the chromosome is 10. The gene $x_{o_{0,0}}$ holds a value equals to 3 meaning that the object $o_{0,0}$ is assigned to the location $p = 3$, it corresponds to CSP_2.

3.5.2 Objective functions. The addressed problem consists in determining a placement of customers data objects on local and/or external storage systems so as to provide a better data access performance while minimizing the placement cost. The data access patterns are changing in time and the customers are mobile or their users can be geographically spread. The customers QoS requirements are expressed in term of I/O storage performance and network latency. Placing a customer object in one location may satisfy one QoS requirement but not the other. Besides, given the characteristics of the system, to reach the main objective (minimizing the placement cost and satisfying the customers SLAs), the customers data objects placement configuration may change over time. This means that some data objects need to be migrated. Migrating data objects cost is variable according to the source and destination properties, which makes the migration cost another dimension to be optimized. Hence, storage, latency and migration costs are considered as the dimensions of the fitness function in the placement problem as shown in Eq. (6).

$$F(x) = \begin{bmatrix} store(x) \\ migrate(x) \\ latency(x) \end{bmatrix} \tag{6}$$

Where x, $store(x)$, $migrate(x)$ and $latency(x)$ respectively represent the decision variable, the storage, migration and latency costs. Figure 5 shows an overview of the considered objective function costs. They encompass the overall placement cost defined in Section 3.4 and will be described in the following sections.

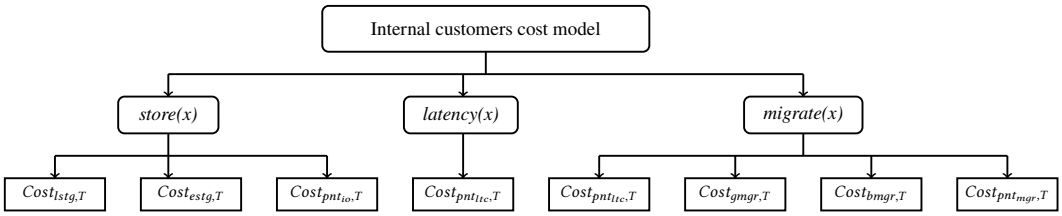

Fig. 5. Objective functions based cost model

Storage cost: The storage cost $store(x)$ is the cost of storing objects in a placement configuration x. It is composed of the local and the external storage costs ($Cost_{lstg,T}$, $Cost_{estr,T}$ from Eq. (3) and Eq. (4)). It also includes the I/O storage performance part penalty cost part $Cost_{pnt_{io},T}$ from the penalty cost (see Eq. (5)). We recall that this penalty represents the additional monetary cost caused by the I/O SLA violation (the network penalty is considered in the latency cost). $store(x)$ is defined as in Eq. (7):

$$store(x) = Cost_{lstg,T} + Cost_{estr,T} + Cost_{pnt_{io},T} \tag{7}$$

Migration cost: The migration cost $migrate(x)$ encompasses the local migration cost $Cost_{lmgr,T}$ from Eq. (3) when the objects are moved between local storage classes, the external migration cost $Cost_{gmgr,T}$ from Eq. (4) when the objects are sent to another CSP and the back migration cost $Cost_{bmgr,T}$ from Eq. (2) when the objects are returned back to the local infrastructure. We have also considered a penalty cost related migration operation $Cost_{pnt_{mgr},T}$. It is induced by the degradation of the QoS caused by the migration. We have calculated it in term of the total number of customers K, the maximum migration time $mrg_{time,T}$ and some unitary costs α as shown in Eq (8).We considered α as a storage performance and latency unitary cost assuming that during the migration time the QoS are not met. By doing so, this migration penalty cost represents an upper bound as the QoS degradation is not necessarily experienced by all customers. In our evaluation, we set α as the sum of the cost corresponding to the average IOPS needed by customers and the latency cost corresponding to the average requests sent by customers per second.

$$Cost_{pnt_{mgr},T} = \alpha * K * mrg_{time,T} \tag{8}$$

$$\begin{aligned} migrate(x) = Cost_{lmgr,T} + Cost_{gmgr,T} \\ + Cost_{bmgr,T} + Cost_{pnt_{mgr},T} \end{aligned} \tag{9}$$

Latency cost: The network latency has been considered as a cost in many research studies as in [11, 49, 68]. In our work, this cost ($latency(x)$) reflects the network penalty cost. As we supposed in Section 3.3, the network latency is billed for customers, and the deterioration of the delivered latency causes losses for the CSP according to the level of its

degradation. Concretely, it is the part of the penalty cost related to the network latency ($Cost_{pnt_{ltc},T}$ taken from Eq. (5)). This cost is the sum of all customers latency penalties as shown in Eq. 10

$$latency(x) = Cost_{pnt_{ltc},T} = \sum_{u_k \in U} Cost_{pnt_{ltc},T}(u_k) \tag{10}$$

$Cost_{pnt_{ltc},T}(u_k)$ represents the network latency penalty of customer u_k. We calculated it by multiplying the number of requests issued by the customer u_k or its end-users from the different zones by the network latency penalty. This penalty is related to the latency between the location of u_k or its end-users and that of the requested objects. It is defined according to Table 2b. So, $Cost_{pnt_{ltc},T}(u_k)$ is calculated as in Eq. (11):

$$Cost_{pnt_{ltc},T}(u_k) = \sum_{Z_m} \sum_{o_{i,k} \in O_k} tot_k^{m,o_{i,k}} * penalty(l(m, x_{o_{i,k}})) \tag{11}$$

3.5.3 Constraints. In this work, we have constraints related to the capacity and the performance throughput limitation of the local and the external storage systems. In addition, there are also constraints related to the customers SLAs.

For each local storage sc_j, the sum of the assigned objects sizes ($S_{o_{i,k}}$) should not exceed the capacity of that storage class csc_j. This is expressed in Eq. (12). Also, the allocated storage service capacity css_d of each CSP_d of the federation is limited. Therefore, the sum of objects sizes affected to each partner storage service ss_d should not exceed its capacity as shown in Eq. (13)

$$\sum_{u_k} \sum_{o_{i,k} \in sc_j} S_{o_{i,k}} \leqslant csc_j \qquad \forall j < J \tag{12}$$

$$\sum_{u_k} \sum_{o_{i,k} \in ss_d} S_{o_{i,k}} \leqslant css_d \qquad \forall d \geqslant J \tag{13}$$

The throughputs of the local and the external storage systems are limited. For the local storage classes, Eq. (14) expresses that the sum of I/O workload composed of the different I/O operations ($op \in \{rr, rw, sr, sw\}$) of the objects ($io_{o_{o_{i,k}}}(op)$) affected to each storage class sc_j must be lower than the I/O throughput delivered by that storage class $io_j(op)$. On the other hand, Eq. (15) indicates that the overall I/O workload issued by all the objects assigned to a given storage service ss_d of a partner CSP_d should not exceed the I/O performance of that storage service io_d.

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in sc_j} io_{o_{o_{i,k}}}(op)}{io_j(op)} \leqslant 1, \qquad \forall j < J \tag{14}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in ss_d} io_{o_{o_{i,k}}}(op)}{io_d} \leqslant 1, \qquad \forall d \geqslant J \tag{15}$$

Regarding the customers I/O SLA, when $ioOffered_k$ for a given customer u_k is between $ioSoft_k$ and $ioHard_k$, the CSP undergoes a storage I/O penalty. However, $ioHard_k$ must be satisfied. This implies that the $ioOffered_k$ storage performance of each customer must exceed his $ioHard_k$ as shown in Eq. (16).

$$ioOffered_k >= ioHard_k \qquad \forall u_k \in U \tag{16}$$

Finally we assume that the objects are not replicated. This condition is guaranteed by the used encoding.

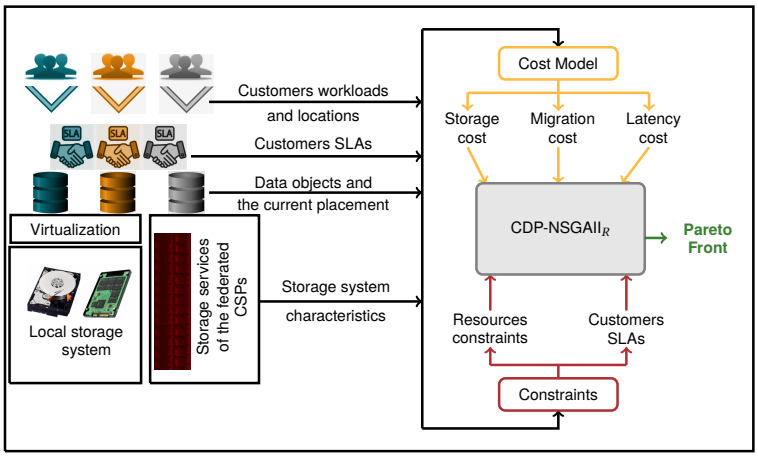

Fig. 6. Federated cloud based hybrid storage system

3.5.4 Summary. Given the system inputs, the objective function and the constraints, we define the multi-objective optimization placement problem as follows:

Let x be a decision variable representing a placement configuration.

$$\min \begin{bmatrix} store(x) \\ migrate(x) \\ latecy(x) \end{bmatrix} \tag{17a}$$

$$S.T. \quad \sum_{u_k} \sum_{o_{i,k} \in sc_j} S_{o_{i,k}} \leqslant csc_j \qquad \forall j < J \tag{17b}$$

$$\sum_{u_k} \sum_{o_{i,k} \in ss_d} S_{o_{i,k}} \leqslant css_d \qquad \forall d \geqslant J \tag{17c}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in sc_j} io_{o_{o_{i,k}}}(op)}{io_j(op)} \leqslant 1, \qquad \forall j < J \tag{17d}$$

$$\sum_{op \in OP} \frac{\sum_{u_k} \sum_{o_{i,k} \in ss_d} io_{o_{o_{i,k}}}(op)}{io_d} \leqslant 1, \qquad \forall d \geqslant J \tag{17e}$$

$$io\text{-}offered_k >= ioHard_k \qquad \forall u_k \in U \tag{17f}$$

Eq. (17a) represents the fitness function, where $store(x)$ is the storage cost, $migrate(x)$ the migration cost and $latency(x)$ the network latency cost.

The first two constraints (17b) and (17c) express internal and external storage resources' capacity limitation. Constraints (17d) and (17e) indicate that these storage resources have a finite performance. Finally, constraint (17f) ensures that the SLA (storage performance) should be bounded.

To solve the above problem, we need to use a multi-objective optimization algorithm tailored to our problem. As shown in Figure 6, this algorithm will use different inputs related to the local storage classes and and foreign services specifications and locations, the workload, SLAs and location of the customers, and finally, the objects specifications and their current placement. For the calculation of objective functions, we used the previously proposed cost model. We have chosen NSGAII to solve the placement problem. However, as evolutionary meta-heuristics do not perform well in difficult problems with complex combinatorial spaces or a high number of constraints [64], we have tailored NSGAII to our placement optimization. This is the subject of the next section.

4 CDP-NSGAII$_{IR}$: CONSTRAINT DATA PLACEMENT NSGAII WITH INJECTION AND REPAIR OPERATORS

Due to the dynamic nature of cloud federation environment and customers workloads, data objects placement strategies need to be performed online.

The placement problem is known to be NP-hard [28, 48]. As a matter of fact, enumerating all possible placement options and eventually choosing the best configuration leads to a combinatorial explosion.

Meta-heuristics constitute a good alternative to alleviate the execution time complexity. Given the popularity and effectiveness of Non-dominated Sorting Genetic Algorithm (NSGAII), we will use it to solve our placement problem. However, being an evolutionary algorithm, NSGAII suffers from issues related to solutions unfeasability when the size of the problem increases. In fact, solutions unfeasability has a negative impact of the resulting Pareto front quality and the execution time. Thus, our implementation of NSGAII needs some modifications to address those issues.

For that, we upgraded our first implementation of NSGAII to a matheuristic [73] by merging it with an exact method. This latter calculates some solutions which are injected to the initial population of NSGAII. Furthermore, we propose a repair function to improve the NSGAII execution time. It modifies the invalid placement configuration by moving some objects to other locations in order to meet the placement constraints. We call the proposed heuristic CDP-NSGAII$_{IR}$ for **C**onstrained **D**ata **P**lacement **NSGAII** with a **I**njection and **R**epair functions.

In this section we will, first, describe the issue of NSGAII evolutionary operators which pushes us to upgrade NSGAII in order to enhance its results for our problem. Then, we will describe the proposed matheuristic.

4.1 NSGAII based approach

Among the strategies used to overcome the execution time complexity are heuristics and meta-heuristics. As we are facing a multi-objective problem, multi-objective evolutionary algorithms (MOEAs) are effective techniques for finding multiple efficient solutions in a reasonable time [40], see Section .

Evolutionary algorithms are iterative stochastic algorithms based on the Darwinian natural selection theory. Their idea is to let a set of solutions (population) evolve according to a given problem over generations, in order to find the best results. The evolution is performed via a set of operators. Generally, selection operators replace individuals for the next generation. *Mutation* and *crossover* operators discover new solutions by, respectively, transforming existing ones or mixing parts of different solutions.

Non-dominated Sorting Genetic Algorithm (NSGAII) is a well-known MOEA used in many optimization problems such as [40, 66, 80]. As for other meta-heuristics, NSGAII has to be customized for the targeted problem. An encoding and operators must be chosen on the shelf or specifically designed. In our case, we have adapted a standard codification to represent placement configurations (see Section 3.5.1 for chromosome strings representation). Also, standard mutation and crossover operators for strings of numbers are used. The standard search operators can produce any combination of values, since they do not consider specific problem constraints.

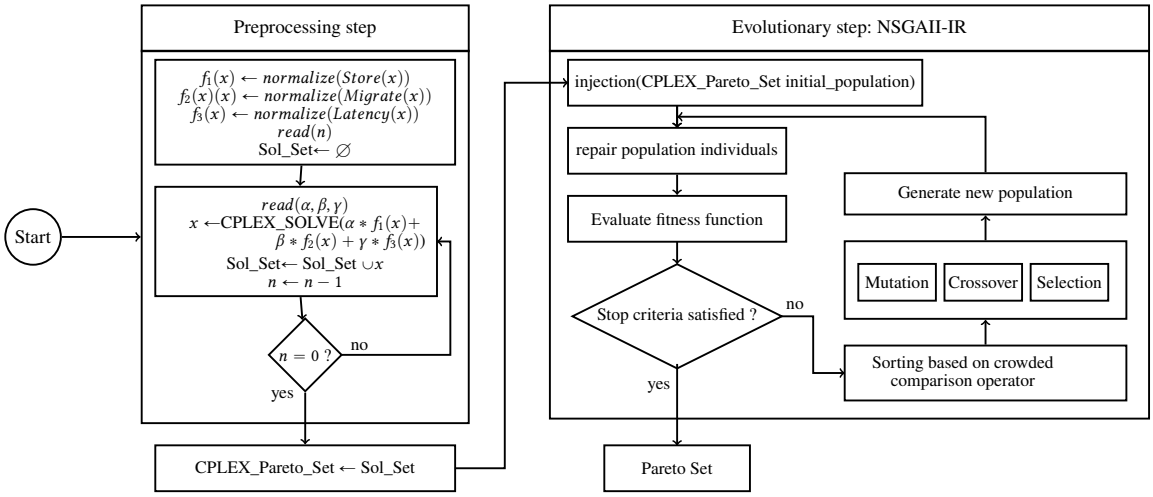

Fig. 7. Matheuristic optimization approach

For example, for $N = 5$ objects and $P = 2$ locations, the $[00011]$ and $[11000]$ solutions can be valid (no resource overflow), but they can be mixed, e.g., into $[11011]$ by a crossover operator, leading to an unfeasible solution, because constraint on location 1 may not be respected or this location may not satisfy the SLAs of all the customers whose objects are placed in this location.

More generally, the search space size induced by the encoding is P^N, while the feasible space can be much smaller, due to resource and customers constraints. This could make the search inefficient, especially for large problems, when the unfeasible region explored uselessly is huge. This results in a poor Pareto front.

There exists three possible ways to deal with the solutions unfeasbility caused by problem constraints (See Section 4.1). The first one consists in modifying the exploration operators, the second one penalizes unfeasible solutions and the third one relies on building a repair function in order to fix an unfeasible solution into a feasible one after applying standard search operators. We used the repair solution for two main reasons: (i) it is simple to implement and allows to use all the well-known and efficient standard operators; (ii) it does not slow down the application of those operators by checking the placement constraints.

Furthermore, NSGAII initial population is enriched with some solutions calculated by an exact method making NSGAII a matheuristic which is the subject of the following section.

4.2 CDP-NSGAII$_{IR}$ matheuristic

The matheuristic optimization process is achieved in two steps : the Preprocessing step and the Evolutionary step as shown in Figure 7. In the former, a set of placement solutions are calculated using CPLEX solver. This process is described later in Section 4.2.1. The output of this step can be used as it is in the Execute phase of the MAPE-K loop. However, in our solution, the set of solutions obtained from the first step is sent to the Evolutionary step to enhance the quality of the final placement. It is injected into the initial population of NGSAII and a repair function (see Section 4.2.2) intervenes before the evaluation of each population individual. Each iteration, the termination conditions are verified, if they are not

satisfied, the population are sorted according to crowded comparison operator and new population is generated based on genetic algorithm operators namely selection, crossover and mutation.

In the following, we will explain how some initial solutions are calculated. Then the repair function is detailed.

4.2.1 Exact method based on MILP tool. The exact method in this work consists in transforming the MOO problem into a single-objective problem and to solve it using a MILP tool. Algorithm 1 formulates the exact method process. First, the min and max of each objective function are calculated (line 2 and line 3 of Algorithm 1). Then, the values are normalized (line 5 to line 7 of Algorithm 1). These functions are weighted and combined into a single objective function. CPLEX [1], is used to solve the obtained single-objective problem (line 9 and line 10 of Algorithm 1). This solving process is repeated several times with different weights in order to generate a set of exact solutions.

Algorithm 1 Calculation of some exact solutions using a MILP solver

1: **function** CALCULATE_EXACT_SOLUTIONS(n)
2: calculate $minStr$, $minMgr$, $minLtc$
3: calculate $maxStr$, $maxMgr$, $maxLtc$
4: population $\leftarrow \varnothing$
5: $Store_N(x) \leftarrow \frac{Store(x)-\min(Store(x))}{\max(Store(x))-\min(Store(x))}$
6: $Migrate_N(x) \leftarrow \frac{Migrate(x)-\min(Migrate(x))}{\max(Migrate(x))-\min(Migrate(x))}$
7: $Latency_N(x) \leftarrow \frac{Latency(x)-\min(Latency(x))}{\max(Latency(x))-\min(Latency(x))}$
8: **for** $i \leftarrow 1$ to n **do**
9: read(α,β, γ) // $\alpha+\beta+\gamma=1$
10: $x \leftarrow CPLEX\_SOLVE(\alpha * Store_N(x) + \beta * Migrate_N(x) + \gamma * Latency_N(x)$ Under all constraints)
11: population $\leftarrow$ population $\cup$ x
12: **end for**
13: **return** (population)

4.2.2 Repair Function. When a solution does not meet all placement constraints defined in Eq. (12), Eq. (13), Eq. (14), Eq. (15) and Eq. (16), objects are moved leading to a feasible solution. Algorithm 2 shows how our repair function works.

The repair function works according to two steps. The first one consists in finding the objects that violate the placement constraints while the second one tries to revise the placement of these objects by moving them to suitable locations responding to both resources limitations and customers SLAs. These steps are explained as follows:

Step 1: To detect all the objects that break the constraints in a chromosome (a placement configuration), we span the chromosome sequentially gene by gene, checking objects locations one by one. If placing a given object to the corresponding gene value does not imply some constraints violation, the value of the gene is kept unchanged. In the other case, it is added to an unfeasible list called $listToRepair$.

At the end, $listToRepair$ contains all the objects that, with the current placement values, do not meet the capacity or the performance of the storage resources constraints or whose locations do not satisfy customers SLAs.

This step corresponds to Algorithm 2, line 2 to line 9.

Let us take the Cloud federation depicted in Figure 4. For the sake of simplicity, we consider only the constraint related to the capacities of local storage classes and external storage services. Suppose that the two local storage classes capacities (csc_0 and csc_1) are respectively 10 and 5 units. CSP_1 storage service capacity cs_1 equals to 8 and CSP_2 capacity cs_2 is 5 units. We assume also that the size of each object equals to 2 units. A placement configuration $x = (1, 3, 1, 0, 1, 1, 0, 3, 3, 2)$ depicted in Figure 8 is not feasible as the capacity of the resources corresponding to locations $p = 1$ (8 units used from 5 available) and $p = 3$ (6 units used from 5 available) is exceeded.

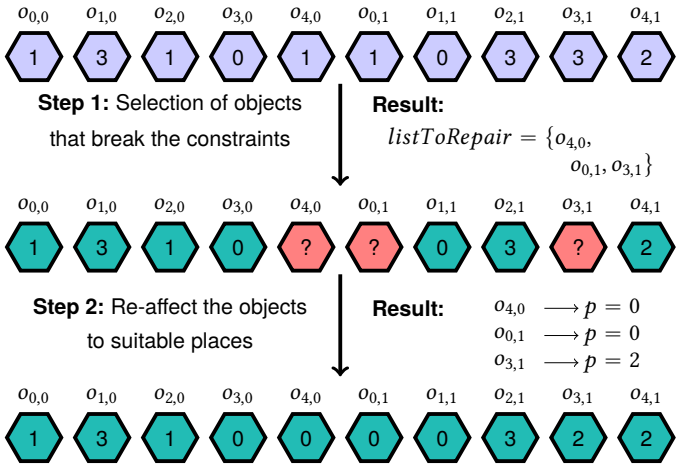

Fig. 8. Repair function: example

The first step of the repair function looks for the objects that violate the capacity constraint. We check the chromosome object by object. The locations of the first four objects $o_{0,0}, o_{1,0}, o_{2,0}, o_{3,0}$ are kept unchanged because they respect the capacity constraints. At this stage, the remaining storage capacities csc_0, csc_1, cs_1, cs_2 are respectively (8,1,8,3). Placing the object $o_{4,0}$ in $p = 1$ corresponding to the local storage class sc_1 exceeds its remaining capacity. So, we place $o_{0,4}$ in $listToRepair$. By doing the same process on the remaining objects, the outcome of this 1^{st} step is the list $listToRepair = \{o_{4,0}, o_{0,1}, o_{3,1}\}$ as illustrated in Figure 8. The remaining storage capacities $(csc_0, csc_1, cs_1, cs_2)$ are respectively (6,1,6,1).

Step 2: we browse $listToRepair$ in a random order for assigning (repairing) the objects to new locations. We prefer the random browsing in order not to favor one object over another. For each object, we build a list $listPlaces$ containing all the existing locations except the one the object is currently assigned to. Then, we randomly choose a new location from $listPlaces$ and check the assignment against resource and customers constraints. If it satisfies all the constraints, we accept the assignment and set the gene value accordingly. If it is not the case, we choose randomly another location from $listPlaces$, and so forth. We preferred to choose a new location randomly to repair a given solution in order to reduce the computational effort. In effect, checking all the potential location for each object and then choosing a location among those that best optimize the objective functions increases significantly the execution time.

Then, if, for a given object, we do not find any location that satisfies all the constraints, the repair process is interrupted and the initial configuration of the chromosome will be kept still, as a potential candidate for future mutation. Algorithm 2 from line 10 to line 27 corresponds to this second step.

In the previous example, during this 2^{nd} step, we browse randomly $listToRepair$. Suppose that the first randomly returned object is $o_{0,1}$. The potential locations for this object are $listPlaces = \{0, 2, 3\}$, that is all other locations regardless of the capacity . We search randomly for a location from this list to fix $o_{0,1}$. Let the returned location be $p = 3$. This location corresponds to ss_2. It could not accommodate $o_{0,1}$ because it does not have enough capacity. So, $p = 3$ will be deleted from $listPlaces$ which will contain now only $\{0, 2\}$. We repeat the process, randomly selecting a location from $listPlaces$. Suppose that $p = 0$ is picked up next, it can accommodate $o_{1,1}$ from a capacity perspective. So, this object is repaired and will be placed in the location $p = 0$. At this stage, $x = (1, 3, 1, 0, ?, 0, 0, 3, ?, 2)$

We delete $o_{0,1}$ from $listToRepair$. We repeat the same process for the remaining objects in $listToRepair$. As shown in Figure 8, the $x = (1, 3, 1, 0, 0, 0, 0, 3, 2, 2)$ represents a possible complete repaired version.

The success rate of the repair function is the fraction of successfully repaired individuals with regard to the total number of unfeasible individuals during the evaluation. It depends on the size of the local and external storage infrastructure, the customers objects, workloads and SLAs. This success rate is related to the pressure that I/O workloads may put on the storage system. The higher the pressure, the lower the success rate as it would be hard for the repair function to meet storage constraints.

The complexity of the repair function equals to $O(N * P)$ where N is the overall number of objects and P is the number of locations. In the worst case, the assignment of each object does not respect at least one of the constraints. Thus, the 1^{st} step outcome $listToRepair$ contains all the objects which means that its size is N. In the second step, the worst case happens when the object is affected to the last location found in the list or in case we do not find a location to fix it after $P - 1$ attempts.

Algorithm 2 Repair function

```
1:  function REPAIR(chromosome)
2:      listToRepair ← ∅
3:      foreach gene g in the chromosome do
4:          o_{i,k} ← object corresponding to g
5:          Check constraints for o_{i,k} using Eqs.{12,13,14,15,16}
6:          if (o_{i,k} does not respect all constraints) then
7:              listToRepair ← listToRepair ∪ o_{i,k}
8:          end if
9:      end for
10:     while (listToRepair ≠ ∅ ) do
11:         listPlaces ← ∅
12:         objectToRepair ← Choose randomly an object from listToRepair
13:         listPlaces.add(all the places except that affected to objectToRepair)
14:         while (listPlaces ≠ ∅ ) do
15:             p ← choose randomly a place from listPlaces
16:             if (objectToRepair respects all constraints in Eqs. (12,13,14,15,16) when affected to p) then
17:                 Affect objectToRepair to p
18:                 listToRepair - {objectToRepair}
19:                 Go to 11
20:             end if
21:             listPlaces - {p}
22:         end while
23:         if ( listPlaces = ∅ and objectToRepair is not fixed) then
24:             return the initial chromosome
25:         end if
26:     end while
27:     return the new chromosome
```

This repair function was integrated in the NSGAII algorithm right before the evaluation of the population individuals.

5 EVALUATION

In this section, we first describe the experiments setting in terms of CSPs and customers characteristics used to evaluate the effectiveness of the proposed approach. Then, we validate the efficacy of the CDP-NSGAII$_{IR}$ matheuristic in terms of execution time and quality of Pareto front. Furthermore, we generalize the proposed approach (injection and replication)

on another meta-heuristic. Finally, we show the flexibility of our approach which makes it possible to propose a set of trade-offs between the execution time and the quality of the obtained solution.

One may note that our study does not focus on the objective function behavior according to input parameters. The relevance of the cost models was already discussed in [20] which is not the objective of this paper. We focus rather on the ability of CDP-NSGAII$_{IR}$ to find a set of solutions close to the Pareto set for such objective functions. By providing an exhaustive and diverse set of precise solutions, we make sure that the CSP has a large set of good solutions from which to choose the most relevant one in his case. Choosing the most relevant solution consists in making a trade-off between the objective functions already discussed, which are the storage, latency an migration costs.

5.1 Experimental setting

We used NSGAII meta-heuristic implementation provided by MOEA framework [2]. We extended it to a matheuristic (CDP-NSGAII$_{IR}$) by integrating an injection and a repair functions. In order to compare the quality of the sets of solutions provided by different algorithms, we use the Hypervolume indicator (HV) to quantify the quality of a set of solutions. This way, we can compare the algorithms, considering that an algorithm providing a higher HV is better.

The experiments were designed based on the following scenario. A cloud federation contains a set of CSPs, each one composed of only one data-center (for simplicity). Each CSP is located in a different geographical zone. The network latency between each pair of zones is randomly assigned in an interval]100ms, 700ms]. The considered CSP handles a set of customers, each of which having one data object (yet again for simplicity). The CSP tries to optimize the placement of its internal customers data objects in order to find a set of relevant solutions. It accommodates a hybrid storage system composed of a set of 20 HDDs and 20 SSDs having respectively the characteristics of 1TB Hitachi Deskstar 7K1000.D model HDS721010DLE630 7200 RPM HDD and 1TB Samsung 850 PRO SSD. We suppose that 30% of the customers are mobile (located in a zone that is different from their CSP's data center).

In our work, we consider that customers' SLA is related to storage performance metrics. If performance are not met, a penalty represented by a percentage of the total charge that is deducted from the customer bill (see Section 3.3). In our evaluation this penalty was set to 10% of the total charge paid by the customer.

As in [11, 68], we suppose that the customers requests network latency is billed to the customers. The price of the latency is given in Table 3 and its related penalty in Table 4.

Table 3. Latency price/req.

Latency (ms)	Price by request ($)
< 100	0.00001
< 300	0.000008
> 300	0

Table 4. Penalty/req.

Latency (ms)	Penalty ($)
< 200	0
< 300	0.000002
> 300	0.00001

Table 5. Storage prices and performance

	Minimum value	Maximum value
Occupancy cost	0.025 $	0.225 $
IOPS cost	0.055 $	0.075 $
Network cost	0.08 $	0.1 $

We used the following configuration for data centers (DC) specifications, customers' workloads, and testing parameters.

DCs specifications: The different CSPs of the federation offer heterogeneous storage services in terms of price and performance. Also, only the outgoing network cost is considered by the CSPs when migrating data objects. We suppose, as in Amazon storage service EBS io1, that the storage price is related to the capacity, performance and duration. In fact, the price of EBS io1 is set to 0.125$/GB-Month for occupation and 0.065$/IOPS-month [4]. Also in Amazon, the network price is 0.09$GB [3]. In order to attribute the CSPs of the federation different characteristics in term of storage performance and services costs, we selected values from ranges specified in Table 5 following a uniform distribution as in [10, 60, 83]. In our evaluation, the Amazon EBS costs and network were set as median values of the used interval. For the

Table 6. Exact method coeffecient values

α	1	0	0	0.5	0.5	0	0.6	0.2	0.2	0.4
β	0	1	0	0.5	0	0.5	0.2	0.6	0.2	0.3
γ	0	0	1	0	0.5	0.5	0.2	0.2	0.6	0.3

storage performance, the maximum IOPS value which can be provided by each CSP was set randomly in [250, 64 000]. This interval bounds reflect the minimum and maximum IOPS that can be supplied by Amazon's EBS disks. By doing so, the different CSPs will have different characteristics in terms of storage services costs and performance.

As in a federation, the service prices may change over time [70] to foster the federation, we suppose that the storage occupancy prices changes according to the occupancy rate as in [60, 70]. In our work, we suppose that the occupancy rate is correlated to the period of time during the day as the generated load has a daily cycle with peak hours [70]. We suppose that it follows a normal distribution ($\mathcal{N}(12,\ 6)$) with a mean equals to 12am and a deviation of 6 hours. Hence, each CSP sets storage prices weighted according to this distribution as in [60].

Workloads: we experimented with the YCSB benchmark [24]. We used different configurations in terms of ratio of read, update, scan and insert requests. We varied the number of request types per customer. We also varied the number of submitted requests per hour.

Two workloads were evaluated in this study. Workload 1 with (100% read) and workload 2 which is 50% read and 50% update.

Workloads generate different I/O operations patterns which we captured using Blktrace tool [8]. As for the occupancy rate, the number of requests per client was weighted according to the aforementioned normal distribution. As in [11], the workload was set according the number of online customers that changes according to a normal distribution.

Coefficients of the exact method: Coefficients in Table 6 were chosen to calculate solutions with the exact method. As CPLEX cannot get optimal solutions in a reasonable time for some instances, we bounded its execution time to 60 seconds.

Testing parameters: We executed the experiments by varying the number of objects N and the number of the locations P that is the number of local storage classes plus the number of CSPs in the federation. These two parameters define the search space size in our work. They allow to test the scalability of the designed heuristic. For the customers workloads, we experimented using different configurations.

We evaluated CDP-NSGAII$_{IR}$ for small and large problem instances in term of both HV and execution time. In our evaluation, an instance is considered as small if it can be evaluated within an hour using the exact method. We compared our contribution results with the standard version of NSGAII and a multi-objective version of Particle Swarm Optimization used in [46] (designated as PSO in the sequel). Finally, we evaluated the success rate of the repair function. Note that the standard NSGAII and PSO meta-heuristics are implemented with penalization technique.

5.2 Evaluation results

In this section, we will first evaluate the effectiveness of the repair function for small problem instances. Then, we will evaluate the efficiency of the proposed matheuristic (injection+repair functions) in terms of HV and execution time for large problem instances. After that, we show the results of the generalization of the proposed approach on other meta-heuristics. Finally, we evaluate the flexibility of the proposed approach by giving different trade-offs between the execution time and the HV in function of the number of solutions calculated by the exact method.

5.2.1 Effectiveness of the repair function for small problem instances. The objective of this evaluation is to investigate the effectiveness of the repair function for problems of small sizes in term of HV and execution time.

In this evaluation, we compared our approach integrating only the repair function (called CDP-NSGAII$_R$) with standard NSGAII, PSO and an exhaustive enumeration method providing the exact Pareto front, in terms of both HV and execution time.

For that, we have calculated the HV and the execution time for 3 small instances of the problem that can be handled by the exact method within one hour. For these instances, we have considered only 3 CSPs in the federation, 2 local storage classes which constitutes 4 different locations to store objects. The experiment was repeated with 10, 13 and 15 objects. The results are presented in Table 7.

Table 7. HV and execution time for small instances

Objects	10		13		15	
Algorithms	HV	Time (s)	HV	Time (s)	HV	Time (s)
Exact Method	0.269	2.4	0.311	154	0.131	2886
CDP-NSGAII$_R$	0.269	0.3	0.310	0.3	0.129	0.3
NSGAII	0.269	0.8	0.311	0.6	0.131	0.6
PSO	0.269	0.2	0.310	0.3	0.130	0.2

As one can see in Table 7, the execution time of the exact method suffers an exponential growth. It takes about 48 minutes for only 15 objects and 4 locations. So, by extrapolation, it would take about 8 years for evaluating a problem with only 5 locations and 20 objects giving 5^{20} possible solutions.

The exact method computes the exact Pareto front, and thus the exact hypervolume. All the other evaluated algorithms obtain more or less the same HV in a small time (less than 1 second). Therefore, contrary to the evaluated evolutionary algorithms, the exact method is unusable for large instances neither offline nor online, because of the combinatorial nature of the placement problem.

For evolutionary algorithms, NSGAII gives the exact HV but doubles the time required by CDP-NSGAII$_R$ and PSO for a non-significant HV enhancement. PSO has the smallest execution time with a HV error that does not exceed 0.7%.

5.2.2 Effectiveness of injection and repair operators. The purpose of this evaluation is to see the impact of injection and repair operators in terms of HV and execution time. For that, we compare the different versions of our approach, PSO and the set of solutions obtained by CPLEX. The different versions of our approach are:

- CDP-NSGAII (standard version of NSGAII tailored to our problem),
- CDP-NSGAII$_R$ (NSGAII with repair operator),
- CDP-NSGAII$_I$ (NSGAII with injection operator),
- CDP-NSGAII$_{IR}$ (NSGAII with injection and repair operators)

Note that we considered that the local CSP has two different storage classes. Each experiment was carried out 10 times. The solutions set of CPLEX is composed of 10 solutions obtained using coefficients in Table 6.

Figure 9 shows, for each algorithm, the hypervolume derivated from its resulting front according to the number of objects, number of CSPs and the workload type. From this figure, we notice that CDP-NSGAII$_{IR}$ and CDP-NSGAII$_I$ give always the best results in term of HV. CDP-NSGAII$_{IR}$ improves the HV of CPLEX by 15% to 60% and NSGAII

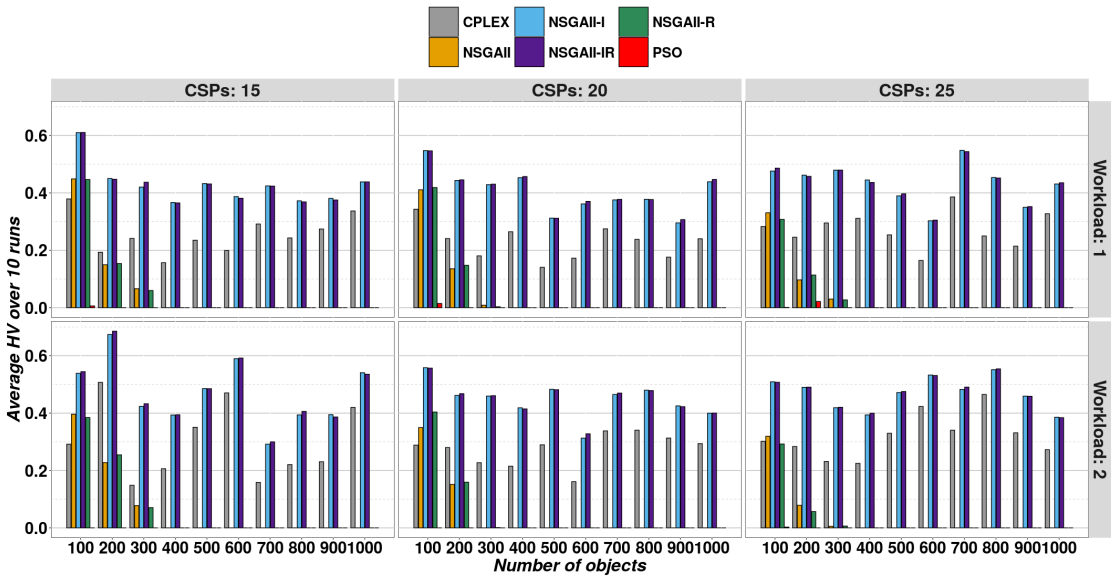

Fig. 9. Hypervolume comparison

meta-heuristic by 4% to 94%. Whereas CDP-NSGAII$_I$ enhances the HV of CPLEX by 14% to 60% and NSGAII meta-heuristic by 4% to 94%.

Regarding standard NSGAII, and PSO, they show small HV, especially PSO, and in most evaluated cases, they lead to an almost empty front with a negligible HV. These results for PSO and NSGAII are explained by the limitations of standard meta-heuristics when the search space increases and valid solutions become proportionally rare in this search space.

Moreover, we notice also that in the cases where NSGAII and PSO could find solutions, NSGAII meta-heuristic improves that PSO by up to 79% in term of HV.

For CDP-NSGAII$_R$, it has roughly the same HV as CDP-NSGAII; nevertheless, from Figure 10, we notice that CDP-NSGAII$_R$ highly improves the execution time of NSGAII. This enhancement is between 40% and 86%. In fact, NSGAII takes more time because it must classify the individuals according to the degree of constraints violation (to penalize invalid individuals). As CDP-NSGAII$_R$ repairs all invalid solutions, it does not perform this additional processing. PSO takes more time than CDP-NSGAII$_R$ as it needs twice more evaluations to obtain a comparable HV.

Moreover, from this figure, we see that the execution time of CDP-NSGAII$_R$ increases linearly with the growth of the number of objects while the number of CSPs impacts slightly the execution time. This is due to the fact that we considered a large number of objects compared to the number of CSPs which is consistent with the theoretical complexity.

From these remarks, we notice that the injection operator has a big impact on the resulting HV while the repair operator has a great impact on the execution time. We can also observe that NSGAII meta-heuristic outperforms the PSO meta-heuristic for the implemented problem.

5.2.3 Generalization of the proposed approach on PSO meta-heuristic . The matheuristic approach proposed in this work can be generalized to other meta-heuristics. In this evaluation, we apply the repair and injection functions to NSGAII and

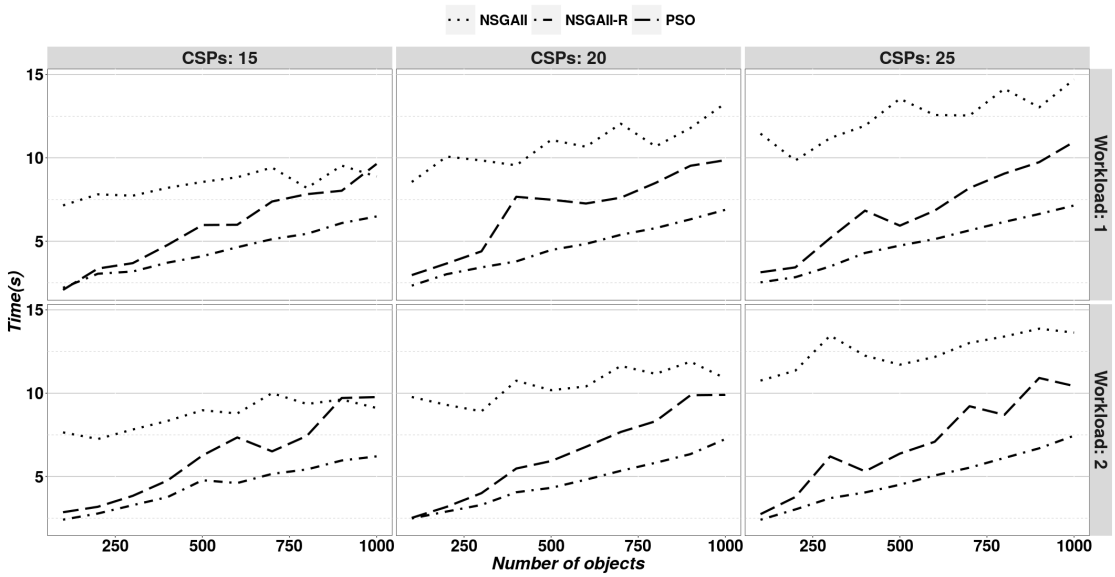

Fig. 10. Time execution evolution

PSO meta-heuristics and evaluate them with and without these two operators. The cumulative HV values of the different tested use cases have been considered.

In addition to the results of the proposed approach with NSGAII meta-heuristic, Figure 11 shows its generalization on PSO meta-heuristic. The legend symbol **(-)** means the standard version of the algorithm, while **(R, I, IR)** symbols mean that the algorithms are upgraded respectively by repair, injection and both injection and repair operators. The horizontal blue line represents the cumulative HV of CPLEX. Note that for this experiment, the solution set of CPLEX is also composed of ten solutions calculated using the coefficient combinations in Table 6.

From this figure we notice that the set of solutions calculated by CPLEX outperforms the HV of the standard versions of NSGAII and PSO respectively by 46% and 57%. CPLEX outperforms also the two meta-heuristics augmented by the repair function. It outperforms NSGAII$_R$ by 44% and PSO$_R$ by 51%. We observe also that both NSGAII and PSO upgraded with injection and repair functions outperform CPLEX and all the other algorithms variants. In fact, NSGAII$_IR$ improves the HV of the CPLEX by 35% while PSO$_IR$ enhances CPLEX by 16%.

5.2.4 Flexibility of the proposed approach. In this evaluation, we show how the proposed approach allows to compromise between the quality and the execution time of the algorithm. For that, we calculate the HV and the execution time of CPLEX and CDP-NSGAII$_{IR}$ by varying the number of solutions calculated by CPLEX and used in the injection step of the matheuristic. We varied the coefficient parameters in the range [0..1] with a step equals to 0.05 for each objective function which gives 231 different coefficient combination. We constitute 11 different solutions sets. The first one contains 21 solutions, the second one contains 42 solutions and so on until the 11th set which contains all (231) solutions. Each time, we inject a set of solutions, we calculate the HV and execution time of CDP-NSGAII$_{IR}$ and CPLEX.

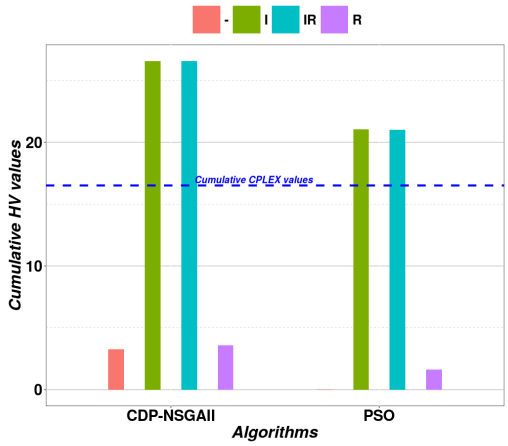

Fig. 11. Matheuristic generalization

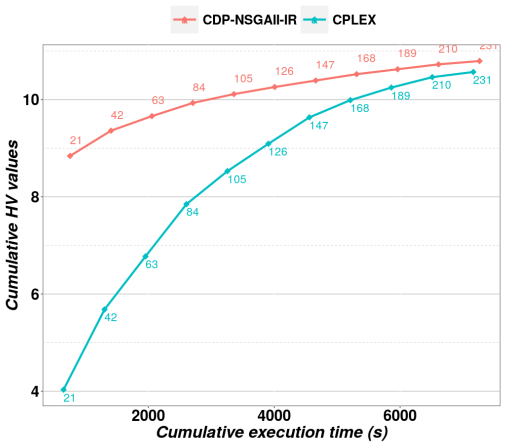

Fig. 12. Matheuristic flexibilty

In this evaluation, we considered both the cumulative execution time and HV values. This is done for different problem instances. The problem instances consist of 15 CSPs and we varied the number of objects between 100 and 1000 objects. We considered both workload 1 and 2.

in Figure 12, red and green points correspond to the execution time and HV of CPLEX and CDP-NSGAII$_{IR}$ respectively. The point labels correspond to the number of solutions calculated by CPLEX. First, we observe that CDP-NSGAII$_{IR}$ takes some additional time to execute because it increases the initial CPLEX set by the evolutionary calculation needed by NSGAII and the repair function. Nevertheless, this extra time of CDP-NSGAII$_{IR}$ is mostly constant because it depends on the number of evaluations of NSGAII meta-heuristic and has no relation with the number of injected solutions. This time corresponds to the slight shifting to the right of the CDP-NSGAII$_{IR}$ dots on the x-axis as compared to the CPLEX dots (better observed on the 231 dot).

From this figure, we noticed that the (cumulative) HV of CDP-NSGAII$_{IR}$ is always higher than that of CPLEX for a given execution time. The matheuristic enhances CPLEX by up to 2.2 times (for 21 injected solutions in the figure). We notice that the difference of HV is more significant for small CPLEX sets and when we increase the number of solutions calculated by CPLEX the difference of HV decreases.

More importantly, one may observe in this Figure that by drawing a horizontal line for a given CDP-NSGAII$_{IR}$, let us say 21 injected solutions, CPLEX needs more than 100 solutions. In addition, with such number of solutions CPLEX execution time is around 10x more significant than CDP-NSGAII$_{IR}$.

So reducing the number of solutions provided by CPLEX is a way to improve the scalability of the matheuristic approach because when the problem size increases, CPLEX takes much more time to find the required exact solutions while the execution time of the heuristic part of CDP-NSGAII$_{IR}$ stays unchanged. Of course, this is done at the expense of the HV and a compromise is to be achieved.

6 RELATED WORK

Great efforts have been made on the data object placement problem in cloud environments based on hybrid storage system. Mono and multi objective optimization techniques and methods have been used and proposed to handle this problem

generally classified to be NP-Hard. This section discusses state-of-the-art studies related to our contribution. We can classify them following six criteria related to our work which are: the storage cost, the storage performance, the SLA, the network latency, the federation environment and finally the MOO as shown in Table 8

Categories	References	Storage cost	Storage performance	Storage SLA	Latency	Federation	MOO
Centralized clouds	[38, 57, 81]	✓	✓				
	[17, 85]	✓	✓	✓			
	[35, 78]		✓				✓
	[37, 46, 47]	✓					✓
Distributed clouds	[50]	✓	✓				
	[76, 79, 83]	✓					
	[9]					✓	✓
	[34]				✓	✓	✓
	[33]						✓
	[74]	✓				✓	✓

Table 8. Related work

In centralized clouds, state-of-the-art studies mixed various types of storage devices and structures in order to find a trade-off between cost and performance within one storage infrastructure. These studies have considered the optimization of one or more factors related to the storage system as the storage and workload cost [17, 38, 57, 81, 85] consumption and the I/O response time. Some of them have also considered the storage SLA [17, 85]. Other studies have addressed the multi-objective optimization in centralized clouds based hybrid storage system as [35, 62, 78].

Regarding distributed clouds, several studies have been proposed [30, 43, 46, 50, 58, 74, 75, 83, 84]. They considered different applications such as bigdata analysis, workflow optimization and geo-replication, etc. The placement problem in these studies have considered the diversity of storage and network services prices offered by the federated clouds. Some of them have used multi-objective optimization [30, 43, 46, 58, 84].The placement problem in these works have been modeled under different objective functions and constraints according to the faced issues. However, generally the storage cost was only related to the capacity (Price/GB) and only [50] have considered the workload at a high level. Also, the storage services characteristics have been rarely considered and to the best of our knowledge the storage services SLA have not been taken into account in distributed clouds.

In one hand, the placement techniques in centralized clouds are generally based on the storage system characteristics and architecture, the workloads I/O patterns and the storage SLA requirements. However, these strategies cannot be used directly in distributed clouds due to the absence of the network factor and the difference in managing federated Clouds. Concerning the placement techniques in distributed clouds, the storage cost was only related to the capacity (Price/GB) and only [50] have considered the workload at a high level. Also, the storage services characteristics have been rarely considered and to the best of our knowledge the storage services SLA have not been taken into account in distributed clouds.

As one can see in Table 8, several costs related to storage systems were not considered in Distributed Cloud studies while Federation properties, network and SLA were generally not considered in Centralized work. We try to fill this gap through the study presented in this paper. Even if the storage issues are not the sole problems of a CSP, strategies such as the one presented in this paper may be used as is or be inserted in an encompassing optimization systems that may take more parameters into account These parameters could be technical (e.g. CPU occupation), or not (e.g. legal, trust issues)

7 CONCLUTION

For a Cloud provider participating in a Federation, choosing a good storage location between local devices and external services provided by other CSPs is a crucial concern especially for time-varying workloads. In this paper, we modeled the problem of customers data objects placement in a cloud federation as a multi-objective optimization problem. The data placement model takes into account the different characteristics of the local storage system and the external storage services, the variable workload of customers and their SLAs.

To resolve the proposed multi-objective problem we designed CDP-NSGAII$_IR$ matheuristic. CDP-NSGAII$_IR$ extends NSGAII by a preprocessing step consisting in calculation of a set of exact solutions that is injected in the initial population of NSGAII. Furthermore a repair function is designed in order to fix a given placement if it is unfeasible in NSGAII populations.

The performed evaluation proved the effectiveness of the proposed matheuristic. CDP-NSGAII$_IR$ improves the NSGAII and CPLEX hypervolume up to 94% and 60% respectively while the repair function enhances the execution time of NSGAII by 68% on average.

For future work, we plan to integrate other objectives to the placement problem such as the availability and the consistency in case of object replication. For that, we plan to propose a replication technique for the data placement in a federated cloud taking into account the cost model considered in this study.

REFERENCES

[1] accessed December, 2018. CPLEX Optimizer. "https://www.ibm.com/fr-fr/analytics/cplex-optimizer".

[2] accessed January, 2020. MOEA Framework. "http://moeaframework.org/".

[3] accessed May, 2020. Amazon Data Transfer. "https://aws.amazon.com/s3/pricing/".

[4] accessed May, 2020. Amazon EBS features. "https://aws.amazon.com/ebs/features/".

[5] accessed November, 2020. Amazon CloudWatch. "https://aws.amazon.com/fr/cloudwatch/".

[6] accessed November, 2020. One Interface To Rule Them All. "http://libcloud.apache.org/".

[7] accessed November, 2020. OpenStack Watcher project. "https://wiki.openstack.org/wiki/Watcher".

[8] Brunelle Alan D. 2008. blktrace User Guide. (2008).

[9] Javier Alsina, Santiago Iturriaga, Sergio Nesmachnow, Andrei Tchernykh, and Bernabé Dorronsoro. 2016. Virtual machine planning for cloud brokering considering geolocation and data transfer. In *2016 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 352–359. http://dx.doi.org/10.1109/CloudCom.2016.0062.

[10] Jörn Altmann and Mohammad Mahdi Kashef. 2014. Cost model based service placement in federated hybrid clouds. *Future Generation Computer Systems* 41 (2014), 79–90. https://doi.org/10.1016/j.future.2014.08.014.

[11] Masoud Saeida Ardekani and Douglas B Terry. 2014. A self-configurable geo-replicated cloud storage system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. 367–381. https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-ardekani.pdf.

[12] Marcio RM Assis and Luiz Fernando Bittencourt. 2016. A survey on cloud federation architectures: identifying functional and non-functional properties. *Journal of Network and Computer Applications* 72 (2016), 51–71. http://dx.doi.org/10.1016/j.jnca.2016.06.014.

[13] Marcio RM Assis, Luiz Fernando Bittencourt, Rafael Tolosana-Calasanz, and Craig A Lee. 2016. Cloud Federations: Requirements, Properties. *Developing Interoperable and Federated Cloud Architecture* (2016), 1.

[14] Charles Audet, J Bigeon, D Cartier, Sébastien Le Digabel, and Ludovic Salomon. 2018. Performance indicators in multiobjective optimization. *Optimization Online* (2018).

[15] Rahma Bouaziz, Laurent Lemarchand, Frank Singhoff, Bechir Zalila, and Mohamed Jmaiel. 2018. Multi-objective design exploration approach for ravenscar real-time systems. *Real-Time Systems* 54, 2 (2018), 424–483.

[16] Djillali Boukhelef, Jalil Boukhobza, and Kamel Boukhalfa. 2016. A cost model for dbaas storage. In *International Conference on Database and Expert Systems Applications*. Springer, 223–239. https://doi.org/10.1007/978-3-319-44403-1_14.

[17] Djillali Boukhelef, Jalil Boukhobza, Kamel Boukhalfa, Hamza Ouarnoughi, and Laurent Lemarchand. 2019. Optimizing the cost of DBaaS object placement in hybrid storage systems. *Future Generation Computer Systems* 93 (2019), 176–187. http://dx.doi.org/10.1016/j.future.2018.10.030.

[18] Jalil Boukhobza and Pierre Olivier. 2017. *Flash Memory Integration: Performance and Energy Issues*. Elsevier. https://www.sciencedirect.com/book/9781785481246/flash-memory-integration.

[19] Antonio Celesti, Francesco Tusa, and Massimo Villari. 2012. Toward cloud federation: concepts and challenges. In *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*. IGI Global, 1–17. http://dx.doi.org/10.4018/978-1-4666-1631-8.ch001.

[20] Amina Chikhaoui, Kamel Boukhalfa, and Jalil Boukhobza. 2018. A Cost Model for Hybrid Storage Systems in a Cloud Federations. In *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 1025–1034. http://dx.doi.org/10.15439/2018F237.

[21] Carlos A Coello Coello. 2002. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: a survey of the state of the art. *Computer methods in applied mechanics and engineering* 191, 11-12 (2002), 1245–1287.

[22] Carlos A. Coello Coello. 2018. Multi-objective Optimization. In *Handbook of Heuristics*, Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende (Eds.). Springer, 177–204. https://doi.org/10.1007/978-3-319-07124-4_17

[23] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. 2007. *Evolutionary algorithms for solving multi-objective problems*. Vol. 5. Springer. https://link.springer.com/book/10.1007/978-0-387-36797-2.

[24] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 143–154. http://dx.doi.org/10.1145/1807128.1807152.

[25] George Darzanos, Iordanis Koutsopoulos, and George D Stamoulis. 2019. Cloud Federations: Economics, Games and Benefits. *IEEE/ACM Transactions on Networking* 27, 5 (2019), 2111–2124.

[26] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation* 6, 2 (2002), 182–197. http://dx.doi.org/10.1109/4235.996017.

[27] Andy Edmonds, Thijs Metsch, Alexander Papaspyrou, and Alexis Richardson. 2012. Toward an open cloud standard. *IEEE Internet Computing* 16, 4 (2012), 15–25.

[28] Kapali P. Eswaran. 1974. Placement of Records in a File and File Allocation in a Computer. In *Information Processing, Proceedings of the 6th IFIP Congress 1974, Stockholm, Sweden, August 5-10, 1974*, Jack L. Rosenfeld (Ed.). North-Holland, 304–307.

[29] Yu Gu, Dongsheng Wang, and Chuanyi Liu. 2014. DR-Cloud: Multi-cloud based disaster recovery service. *Tsinghua Science and Technology* 19, 1 (2014), 13–23.

[30] Lizheng Guo, Zongyao He, Shuguang Zhao, Na Zhang, Junhao Wang, and Changyun Jiang. 2012. Multi-objective optimization for data placement strategy in cloud computing. In *International Conference on Information Computing and Applications*. Springer, 119–126. https://doi.org/10.1007/978-3-642-34041-3_18.

[31] Arunima Hota, Subasish Mohapatra, and Subhadarshini Mohanty. 2019. Survey of different load balancing approach-based algorithms in cloud computing: a comprehensive review. In *Computational Intelligence in Data Mining*. Springer, 99–110. https://doi.org/10.1007/978-981-10-8055-5_10.

[32] Binbing Hou, Feng Chen, Zhonghong Ou, Ren Wang, and Michael Mesnier. 2017. Understanding I/O performance behaviors of cloud storage from a client's perspective. *ACM Transactions on Storage (TOS)* 13, 2 (2017), 1–36. https://doi.org/10.1145/3078838.

[33] Santiago Iturriaga, Sergio Nesmachnow, Andrei Tchernykh, and Bernabé Dorronsoro. 2016. Multiobjective workflow scheduling in a federation of heterogeneous green-powered data centers. In *2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE, 596–599. http://dx.doi.org/10.1109/CCGrid.2016.34.

[34] Lei Jiao, Jun Lit, Wei Du, and Xiaoming Fu. 2014. Multi-objective data placement for multi-cloud socially aware services. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 28–36. http://dx.doi.org/10.1109/INFOCOM.2014.6847921.

[35] Elena Kakoulli and Herodotos Herodotou. 2017. OctopusFS: A distributed file system with tiered storage management. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 65–78. http://dx.doi.org/10.1145/3035918.3064023.

[36] Jeffrey O Kephart and David M Chess. 2003. The vision of autonomic computing. *Computer* 36, 1 (2003), 41–50.

[37] Nagma Khattar, Jaiteg Singh, and Jagpreet Sidhu. 2019. Multi-criteria-Based Energy-Efficient Framework for VM Placement in Cloud Data Centers. *Arabian Journal for Science and Engineering* (2019), 1–15. https://doi.org/10.1007/s13369-019-04048-6.

[38] Youngjae Kim, Aayush Gupta, Bhuvan Urgaonkar, Piotr Berman, and Anand Sivasubramaniam. 2011. HybridStore: A cost-efficient, high-performance storage system combining SSDs and HDDs. In *2011 IEEE 19th annual international symposium on modelling, analysis, and simulation of computer and telecommunication systems*. IEEE, 227–236. http://dx.doi.org/10.1109/MASCOTS.2011.64.

[39] Dimitrios G Kogias, Michael G Xevgenis, and Charalampos Z Patrikakis. 2016. Cloud federation and the evolution of cloud computing. *Computer* 49, 11 (2016), 96–99.

[40] Hemant Kumar and Shiv Prasad Yadav. 2019. Fuzzy rule-based reliability analysis using NSGA-II. *International Journal of System Assurance Engineering and Management* 10, 5 (2019), 953–972. https://doi.org/10.1007/s13198-019-00826-5.

[41] Dongwoo Lee, Changwoo Min, and Young Ik Eom. 2015. Effective flash-based SSD caching for high performance home cloud server. *IEEE Transactions on Consumer Electronics* 61, 2 (2015), 215–221. https://doi.org/10.1109/TCE.2015.7150596.

[42] Laurent Lemarchand, Damien Massé, Pascal Rebreyend, and Johan Håkansson. 2018. Multiobjective Optimization for Multimode Transportation Problems. *Advances in Operations Research* 2018 (2018). http://dx.doi.org/10.1155/2018/8720643.

[43] Chunlin Li, YaPing Wang, Hengliang Tang, and Youlong Luo. 2019. Dynamic multi-objective optimized replica placement and migration strategies for SaaS applications in edge cloud. *Future Generation Computer Systems* 100 (2019), 921–937. https://doi.org/10.1016/j.future.2019.05.003.

[44] Hongxing Li, Chuan Wu, Zongpeng Li, and Francis CM Lau. 2013. Profit-maximizing virtual machine trading in a federation of selfish clouds. In *INFOCOM, 2013 Proceedings IEEE*. IEEE, 25–29. http://dx.doi.org/10.1109/infcom.2013.6566728.

[45] Zhichao Li, Ming Chen, Amanpreet Mukker, and Erez Zadok. 2015. On the trade-offs among performance, energy, and endurance in a versatile hybrid drive. *ACM Transactions on Storage (TOS)* 11, 3 (2015), 1–27. https://doi.org/10.1145/2700312.

[46] Xiyang Liu, Lei Fan, Liming Wang, and Sha Meng. 2015. PSO based multiobjective reliable optimization model for cloud storage. In *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure*

Computing; Pervasive Intelligence and Computing. IEEE, 2263–2269. http://dx.doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.334.

[47] Xiyang Liu, Lei Fan, Liming Wang, and Sha Meng. 2016. Multiobjective reliable cloud storage with its particle swarm optimization algorithm. *Mathematical Problems in Engineering* 2016 (2016).

[48] Mostafa Mahi, Omer Kaan Baykan, and Halife Kodaz. 2018. A new approach based on particle swarm optimization algorithm for solving data allocation problem. *Applied Soft Computing* 62 (2018), 571–578. https://doi.org/10.1016/j.asoc.2017.11.019.

[49] Yaser Mansouri and Rajkumar Buyya. 2016. To move or not to move: Cost optimization in a dual cloud-based storage architecture. *Journal of Network and Computer Applications* 75 (2016), 223–235. https://doi.org/10.1016/j.jnca.2016.08.029.

[50] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. 2017. Cost optimization for dynamic replication and migration of data in cloud data centers. *IEEE Transactions on Cloud Computing* (2017). http://dx.doi.org/10.1109/tcc.2017.2659728.

[51] Yaser Mansouri, Adel Nadjaran Toosi, and Rajkumar Buyya. 2017. Data storage management in cloud environments: Taxonomy, survey, and future directions. *ACM Computing Surveys (CSUR)* 50, 6 (2017), 1–51. http://dx.doi.org/10.1145/3136623.

[52] Rafael Moreno-Vozmediano, Eduardo Huedo, Ignacio M Llorente, Rubén S Montero, Philippe Massonet, Massimo Villari, Giovanni Merlino, Antonio Celesti, Anna Levin, Liran Schour, et al. 2016. BEACON: a cloud network federation framework. In *Communications in Computer and Information Science*. Springer, 325–337. http://dx.doi.org/10.1007/978-3-319-33313-7_25.

[53] Anirban Mukhopadhyay, Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Carlos Artemio Coello Coello. 2013. A survey of multiobjective evolutionary algorithms for data mining: Part I. *IEEE Transactions on Evolutionary Computation* 18, 1 (2013), 4–19.

[54] Anirban Mukhopadhyay, Ujjwal Maulik, Sanghamitra Bandyopadhyay, and Carlos Artemio Coello Coello. 2014. A survey of multiobjective evolutionary algorithms for data mining: Part I. *IEEE Transactions on Evolutionary Computation* 18, 1 (2014), 4–19. http://dx.doi.org/10.1109/TEVC.2013.2290086.

[55] Nadia Nedjah and Luiza de Macedo Mourelle. 2015. Evolutionary multi–objective optimisation: a survey. *International Journal of Bio-Inspired Computation* 7, 1 (2015), 1–25. http://dx.doi.org/10.1504/IJBIC.2015.067991.

[56] Hamza Ouarnoughi, Jalil Boukhobza, Frank Singhoff, and Stéphane Rubini. 2014. A multi-level I/O tracer for timing and performance storage systems in IaaS cloud.. In *REACTION*. https://doi.org/10.1145/3041710.3041715.

[57] Mehdi Pirahandeh and Deok-Hwan Kim. 2018. EGE: A New Energy-Aware GPU Based Erasure Coding Scheduler for Cloud Storage Systems. In *2018 Tenth International Conference on Ubiquitous and Future Networks (ICUFN)*. IEEE, 619–621. http://dx.doi.org/10.1109/ICUFN.2018.8436594.

[58] Fabio López Pires and Benjamín Barán. 2013. Multi-objective virtual machine placement with service level agreement: A memetic algorithm approach. In *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*. IEEE Computer Society, 203–210. https://doi.org/10.1109/UCC.2013.44.

[59] Benay Kumar Ray, Avirup Saha, Sunirmal Khatua, and Sarbani Roy. 2019. Toward maximization of profit and quality of cloud federation: solution to cloud federation formation problem. *The Journal of Supercomputing* 75, 2 (2019), 885–929.

[60] Salma Rebai, Makhlouf Hadji, and Djamal Zeghlache. 2015. Improving profit through cloud federation. In *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE, 732–739. http://dx.doi.org/10.1109/ccnc.2015.7158069.

[61] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. 2015. Performance metrics in multi-objective optimization. In *2015 Latin American Computing Conference (CLEI)*. IEEE, 1–11.

[62] Amine Roukh, Ladjel Bellatreche, Selma Bouarar, and Ahcene Boukorca. 2017. Eco-physic: Eco-physical design initiative for very large databases. *Information Systems* 68 (2017), 44–63. https://doi.org/10.1016/j.is.2017.01.003.

[63] Takfarinas Saber, Anthony Ventresque, Xavier Gandibleux, and Liam Murphy. 2014. Genepi: A multi-objective machine reassignment algorithm for data centres. In *International workshop on hybrid metaheuristics*. Springer, 115–129. https://doi.org/10.1007/978-3-319-07644-7_9.

[64] Sancho Salcedo-Sanz. 2009. A survey of repair methods used as constraint handling techniques in evolutionary algorithms. *Computer science review* 3, 3 (2009), 175–192. http://dx.doi.org/10.1016/j.cosrev.2009.07.001.

[65] Mohamed A Sharaf, Panos K Chrysanthis, Alexandros Labrinidis, and Cristiana Amza. 2009. Optimizing i/o-intensive transactions in highly interactive applications. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. ACM, 785–798. https://doi.org/10.1145/1559845.1559927.

[66] A Sathya Sofia and P GaneshKumar. 2018. Multi-objective task scheduling to minimize energy consumption and makespan of cloud computing using NSGA-II. *Journal of Network and Systems Management* 26, 2 (2018), 463–485. https://doi.org/10.1007/s10922-017-9425-0.

[67] Amir Taherkordi, Feroz Zahid, Yiannis Verginadis, and Geir Horn. 2018. Future cloud systems design: challenges and research directions. *IEEE Access* 6 (2018), 74120–74150. http://dx.doi.org/10.1109/ACCESS.2018.2883149.

[68] Douglas B Terry, Vijayan Prabhakaran, Ramakrishna Kotla, Mahesh Balakrishnan, Marcos K Aguilera, and Hussam Abu-Libdeh. 2013. Consistency-based service level agreements for cloud storage. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 309–324. https://doi.org/10.1145/2517349.2522731.

[69] Adel Nadjaran Toosi, Rodrigo N Calheiros, and Rajkumar Buyya. 2014. Interconnected cloud computing environments: Challenges, taxonomy, and survey. *ACM Computing Surveys (CSUR)* 47, 1 (2014), 7. http://dx.doi.org/10.1145/2593512.

[70] Adel Nadjaran Toosi, Rodrigo N Calheiros, Ruppa K Thulasiram, and Rajkumar Buyya. 2011. Resource provisioning policies to increase iaas provider's profit in a federated cloud environment. In *High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on*. IEEE, 279–287. http://dx.doi.org/10.1109/hpcc.2011.44.

[71] Adel Nadjaran Toosi, Ruppa K Thulasiram, and Rajkumar Buyya. 2012. Financial option market model for federated cloud environments. In *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. IEEE, 3–12.

[72] Paolo Viotti, Dan Dobre, and Marko Vukolić. 2017. Hybris: Robust hybrid cloud storage. *ACM Transactions on Storage (TOS)* 13, 3 (2017), 1–32. https://doi.org/10.1145/3119896.

[73] Stefan Voss, V Maniezzo, and T Stützle. 2009. MATHEURISTICS: Hybridizing Metaheuristics and Mathematical Programming (Annals of Information Systems). (2009).

[74] Pengwei Wang, Caihui Zhao, Wenqiang Liu, Zhen Chen, and Zhaohui Zhang. 2020. Optimizing data placement for cost effective and high available multi-cloud storage. *Computing and Informatics* 39, 1-2 (2020), 51–82.

[75] Zhenyu Wen, Jacek Cała, Paul Watson, and Alexander Romanovsky. 2016. Cost effective, reliable and secure workflow deployment over federated clouds. *IEEE Transactions on Services Computing* 10, 6 (2016), 929–941. http://dx.doi.org/10.1109/TSC.2016.2543719.

[76] Zhenyu Wen, Jacek Cała, Paul Watson, and Alexander Romanovsky. 2017. Cost effective, reliable and secure workflow deployment over federated clouds. *IEEE Transactions on Services Computing* 10, 6 (2017), 929–941. https://doi.org/10.1109/CLOUD.2015.86.

[77] Lyndon While, Philip Hingston, Luigi Barone, and Simon Huband. 2006. A faster algorithm for calculating hypervolume. *IEEE transactions on evolutionary computation* 10, 1 (2006), 29–38.

[78] Yizi Wu and Youtao Zhang. 2015. GA Based Placement Optimization for Hybrid Distributed Storage. In *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. IEEE, 198–203. https://doi.org/10.1109/HPCC-CSS-ICESS.2015.89.

[79] Wenhua Xiao, Weidong Bao, Xiaomin Zhu, and Ling Liu. 2017. Cost-aware big data processing across geo-distributed datacenters. *IEEE Transactions on Parallel and Distributed Systems* 28, 11 (2017), 3114–3127. https://doi.org/10.1109/TPDS.2017.2708120.

[80] Xiaolong Xu, Shucun Fu, Yuan Yuan, Yun Luo, Lianyong Qi, Wenmin Lin, and Wanchun Dou. 2019. Multiobjective computation offloading for workflow management in cloudlet-based mobile cloud using NSGA-II. *Computational Intelligence* 35, 3 (2019), 476–495. https://doi.org/10.1111/coin.12197.

[81] Shu Yin, Bing Jiao, Xiaomin Zhu, Xiaojun Ruan, Si Chen, and Zhuo Tang. 2018. DuoFS: A Hybrid Storage System Balancing Energy-Efficiency, Reliability, and Performance. In *2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*. IEEE, 478–485. http://dx.doi.org/10.1109/PDP2018.2018.00082.

[82] Boyang Yu and Jianping Pan. 2015. Location-aware associated data placement for geo-distributed data-intensive applications. In *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 603–611. http://dx.doi.org/10.1109/INFOCOM.2015.7218428.

[83] Linquan Zhang, Chuan Wu, Zongpeng Li, Chuanxiong Guo, Minghua Chen, and Francis CM Lau. 2013. Moving big data to the cloud: An online cost-minimizing approach. *IEEE Journal on Selected Areas in Communications* 31, 12 (2013), 2710–2721. http://dx.doi.org/10.1109/JSAC.2013.131211.

[84] Miao Zhang, Huiqi Li, Li Liu, and Rajkumar Buyya. 2018. An adaptive multi-objective evolutionary algorithm for constrained workflow scheduling in Clouds. *Distributed and Parallel Databases* 36, 2 (2018), 339–368. http://dx.doi.org/10.1007/s10619-017-7215-z.

[85] Ning Zhang, Junichi Tatemura, Jignesh M Patel, and Hakan Hacigümüş. 2011. Towards cost-effective storage provisioning for DBMSs. *Proceedings of the VLDB Endowment* 5, 4 (2011), 274–285. http://dx.doi.org/10.14778/2095686.2095687.

[86] Qi Zhang, Lu Cheng, and Raouf Boutaba. 2010. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications* 1, 1 (2010), 7–18.

[87] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. 2003. Performance assessment of multiobjective optimizers: An analysis and review. *IEEE Transactions on evolutionary computation* 7, 2 (2003), 117–132.