Efficient Replica Migration Scheme for Distributed Cloud Storage Systems

Amina Mseddi, *Student Member, IEEE,* Mohammad A. Salahuddin, *Member, IEEE,* Mohamed Faten Zhani, *Senior Member, IEEE,* Halima Elbiaze, *Member, IEEE,* and Roch H. Glitho, *Senior Member, IEEE*

Abstract—With the wide adoption of large-scale internet services and big data, the cloud has become the ideal environment to satisfy the ever-growing storage demand. In this context, data replication has been touted as the ultimate solution to improve data availability and reduce access time. However, replica management systems usually need to migrate and create a large number of data replicas over time between and within data centers, incurring a large overhead in terms of network load and availability. In this paper, we propose CRANE, an effiCient Replica migrAtion scheme for distributed cloud Storage systEms. CRANE complements any replica placement algorithm by efficiently managing replica creation in geo-distributed infrastructures in order to (1) minimize the time needed to copy the data to the new replica location, (2) avoid network congestion, and (3) ensure the minimum desired availability for the data. Through simulation and experimental results, we show that CRANE provides a sub-optimal solution for the replica migration problem with lower computational complexity than its integer linear program formulation. We also show that, compared to OpenStack Swift, CRANE is able to reduce by up to 60% the replica creation and migration time and by up to 50% the inter-data center network traffic while ensuring the minimum required data availability.

Index Terms—Cloud storage, data availability, data migration, replica management

1 Introduction

With the wide adoption of large-scale Internet services and the increasing amounts of exchanged data, the cloud has become the ultimate resort to cater to the ever-growing demand for storage, providing seemingly limitless capacity, high availability and faster access time. Typically, cloud providers build several large-scale data centers in geographically distributed locations. They then rely on data replication as an effective technique to improve fault-tolerance, reduce end-user latency and minimize the amount of data exchanged through the network. Consequently, effective replica management has become one of the major challenges for cloud providers [1].

In recent years, a large body of work has been devoted to address this challenge and, more specifically, to address the problem of replica placement considering several goals, such as minimizing storage costs, improving fault-tolerance and access delays [2], [3], [4], [5], [6]. However, replica placement schemes may result in a large number of data replicas created or migrated over time between and within data centers, incurring significant amounts of traffic exchange. This might happen in several scenarios requiring the creation and the relocation of a large number of replicas: when a new data center is added to the cloud provider's infrastructure, when a data center is scaled up or down, when recovering from a disaster or simply when replicas are relocated to achieve performance or availability goals.

Naturally, several impacts may be expected when such large data bulk transfer of replicas is triggered. First, as copying data consumes resources (e.g., CPU, memory, disk I/O) at both the source and the destination machines, these nodes will experience more contention for the available capacity, which may slow down other tasks running on them. Second, recent research revealed that traffic exchanged between data centers account for up to 45% of the total traffic in the backbone network connecting them [7]. This ever-growing exchange of tremendous amounts of data between data centers may overload the network, especially when using the same paths or links. This can hurt the overall network performance in terms of latency and packet loss. Moreover, replica migration processes are usually distributed and asynchronous as is the case for Swift, the OpenStack project for managing data storage [8]. That is, when a replica is to be relocated or created in a new destination machine, every machine in the infrastructure already storing the same replica will try to copy the data to the new destination. There is no coordination or synchronization between the sending nodes. This will not only lead to unneeded redundancy as the same data is copied from different sources at the same time, but will also further exacerbate the congestion in the data center network. Finally, Replicas are usually placed in geographically distributed locations in order to increase data availability over time and reduce user-perceived latency. When a replica has to be created/migrated in a new location, it will not be

- *A. Mseddi and H. Elbiaze are with the Department of Computer Science, University of Quebec at Montreal, Montreal, Quebec, Canada. E-mail: mseddi.amina@courrier.uqam.ca, E-mail: elbiaze.halima@uqam.ca.*
- *M. A. Salahuddin is with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. E-mail: mohammad.salahuddin@ieee.org*
- *M. F. Zhani is with the Department of Software and IT Engineering, École de Technologie Supérieure (ÉTS Montreal), Montreal, Quebec, Canada. E-mail: mfzhani@etsmtl.ca*
- *R. H. Glitho is with Concordia Institute for Information Systems Engineering, Concordia University, Montreal, Quebec, Canada. E-mail: glitho@ciise.concordia.ca*

available until all its content is copied from other existing replicas. If this process takes too long, it might hurt the overall data availability if the number of available replicas is not sufficient to accommodate all user requests. For instance, in order to ensure data availability, Swift ensures that at least two replicas of the data are available at any point in time (according to the default configuration [8]).

In order to alleviate all the aforementioned problems, it is critical to make sure that replicas are created as soon as possible in their new locations without incurring network congestion or high creation time. This requires to carefully select the source replica from which the data will be copied, the paths through which the data will be transferred and the order in which replicas will be created.

To address these challenges, we propose CRANE an efficient replica migration scheme for distributed cloud storage systems. CRANE is a novel scheme that manages replica creation in geo-distributed infrastructures with the goal of (1) minimizing the time needed to copy the data to the new replica location, (2) avoiding network congestion, and (3) ensuring a minimal availability for each replica. CRANE can be used along with any existing replica placement algorithm in order to optimize the time to create and copy replicas and to minimize resources needed to reach the new placement of the replicas. In addition, it ensures that at any point in time, data availability is above a predefined minimum value.

This paper is an extension of our previous work [9]. It provides a more comprehensive overview of existing replica migration solutions and compares them to CRANE. We also extend the performance evaluation results to run real experiments and also realistic simulations for large-scale infrastructures. Hence, we have implemented CRANE and integrate it into OpenStack. We use this implementation to run real experiments that show the performance of CRANE over OpenStack Swift for different scenarios. We also compare the solution found with CRANE with the optimal solution found by the ILP of the problem that we have proposed and implemented using IBM ILOG CPLEX optimizer [10]. We implemented the replica migration problem with AMPL [11].

This paper is organized as follows. Section 2 surveys the related work on replica placement and migration in the cloud. Section 3 presents an example illustrating how replica creation and migration strategy can impact performance metrics. In Section 4 we formulated the replica migration problem as an Integer Linear Program (ILP). We then present our proposed solution in Section 5. We describe in Section 6 the simulation and experimental results comparing the optimal replica migration plan found by the ILP to that found with CRANE and OpenStack Swift. Finally, we conclude in Section 7.

2 Related Work

A large body of work has been devoted to data management in distributed storage systems. As our work is tightly related to data and replica management in storage clouds, we first present existing literature on data migration in general then focus on replica placement and migration work.

2.1 Data Migration

The data migration problem aims at finding an efficient schedule to move large amounts data between infrastructures. The basic migration problem is the case where all storage devices have point to point and equal capacity links, data objects have the same size and have one copy and all devices can migrate one data object at a time. To address this problem, researchers have applied multigraph edge-coloring algorithms to produce efficient data migration schedule [12], [13], [14]. The idea is to model the migrations between different infrastructure as the edges of a graph where the nodes are the data source and destination. Edges with the same color represent migrations that can be scheduled simultaneously. However, finding a migration plan with the minimum number of rounds is NP-hard.

Kari et al. [14] proposed a scheme that tries to find an optimal migration schedule for data that minimizes the total migration time while taking into consideration the heterogeneity transfer capacity of storage nodes. However, their solution overlooks availability requirements as well as network-related constraints such as bandwidth limits and propagation delays.

Wu et al. [15] have designed a service that schedules the dynamically-arising inter-data center migration requests with different urgency levels requiring different data transfer finishing deadlines. These migration requests can be optimally and dynamically arranged to fully exploit the available bandwidth at any time. Therefore, in our work, we are going to deal with multiple replicas of the same data, so that we can choose the source of the migration to fully exploit the available bandwidth of the network linking different data centers. Besides, their work deals with different finishing migration deadlines for each data. In our case we aim to minimize as much as possible the overall migration time. [16] have also considered dynamically-arising inter-data center migration requests. They proposed a control-theoretical approach to statistically guarantee a bound on the amount of impact on foreground work during a data migration, while still accomplishing the data migration in as short a time as possible. Works that consider dynamically-arising inter-data center migration requests, often have to make decisions either to accept the request or not. However in our work we satisfy all migration requests. That is we find the optimal migration schedule to reach the new optimal placement of replicas.

Petrol et al. [17] proposed an optimization plan to data migration in order to speed up scaling the cloud. For that, they have divided migrations tasks on scaling migrations and residual migration. Indeed scaling migrations considers data transfer to adjacent nodes and residual migrations are for more distant nodes.

2.2 Replica management

Recently, cloud data centers are being created in large numbers [18], [19], [20]. This trend is driven primarily by the need to place data closer to customers to improve QoS [1] and to provide failure tolerance [21]. Moreover, in order to achieve these objectives, data replication is often considered. In fact, the right placement of these replicas leads to minimization of communication cost [5], maximization of availability [1], [5], [6] and improvement in QoS [1], [22]. Over time, and depending on the targeted performance objectives, some replicas are torn down and others should be created or migrated across the infrastructure. Such operations might have a huge overhead and impact on the overall performance. Recently, few proposals have tried to address this problem and designed efficient replica creation and migration schemes with the goal of minimizing such overhead [23], [24], [25], [26].

For instance, S. Khuller et al. [23] mapped replica migration problem to the gossiping and the broadcasting problems and proposes several approximation algorithms to solve the problem in order to minimize replica migration time. N. Tziritas et al. [24] have considered two selection criterion options for selecting the replicas to transfer first, namely *Earliest Start Time* (EST) and the *Earliest Completion Time* (ECT), computed based on the transfers that have been scheduled and the available link bandwidth.

T. Loukopoulos et al. focus on minimizing bandwidth consumption during the replica creation [26], [27]; however, they assume that the links have the same capacity. In [25], the authors have studied the case where the same object needs to be migrated to multiple destinations. They hence use the best multicast-tree (Steiner tree) to migrate objects to other servers. However, this solution requires all the intermediate nodes to be able to start forwarding the object on the fly even if they do not have the entire object.

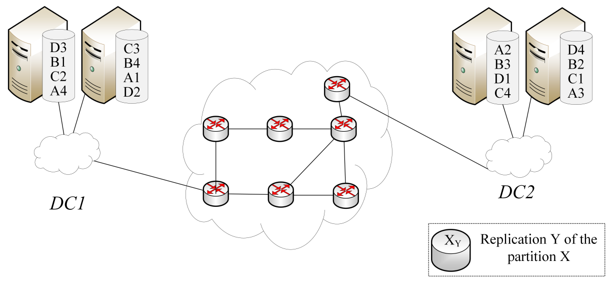
Swift, the OpenStack project for managing data storage [8], implements a replica placement algorithm along with a replica migration scheme. Using Swift, blocks of data (called also partitions) has a defined number of replicas (three by default) that are distributed across the infrastructure according to the *as-unique-as-possible* algorithm [28]. This algorithm ensures that replicas are physically stored as far as possible from each other in order to improve data availability. Swift computes the optimal replica placement periodically with a predefined time interval (usually, one hour) and then triggers replica migration to reach the new optimal placement. However, it limits to one the number of migrations per partition so as to ensure that at least two replicas are available at any time. Such constraint increases the time needed to reach the new optimal placement, especially when multiple replicas of the same partition need to be created. Furthermore, Swift does not take into consideration available bandwidth in the network when migrating replicas between different locations, which might cause congestion in the network.

Different from previous work, CRANE takes into account not only the network available bandwidth but also data availability during the creation of the new replicas. It also capitalizes on the existence of multiple replicas across the network to carefully select the source of the data and the transmission path so as to avoid network congestion and minimize data migration time.

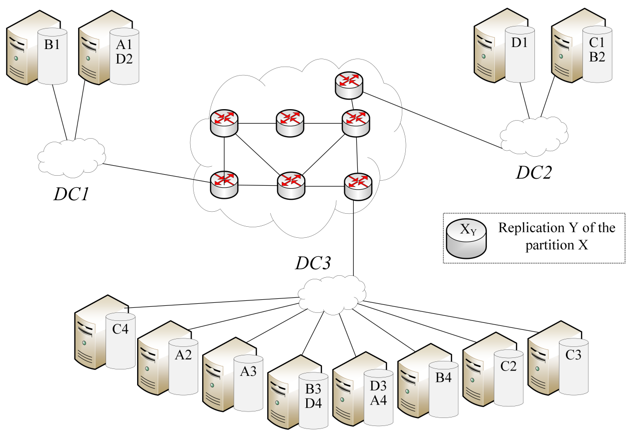
Table 1 compares the surveyed proposals in terms of assumptions and goals. As seen in the table, CRANE work is the first to focus on minimizing the migration time while taking into consideration the required data availability and, at the same time, avoiding network congestion.

3 Motivating Example

To introduce our proposed replica placement solution, we provide in this Section a motivating example to highlight some limitations of distributed storage systems. Let us consider a cloud system composed of two data centers (DC1 and DC2) located at different geographic regions and connected through a backbone network, as shown in Fig. 1(a). We use Swift to manage the storage distributed over the two data centers. We assume that we have 4 partitions A, B, C and D with sizes 300 GB, 100 GB, 500 GB and 200 GB, respectively. Each partition has 4 replicas that are placed by the *as-unique-as-possible* algorithm that strives to increase data availability. Fig. 1(a) shows the initial mapping of the replicas across the infrastructure. For instance, the four replicas of partition A (denoted by A1, A2, A3 and A4) are distributed across the two data centers. The same applies to the other partitions.



(a) Initial replica mapping with two data centers



(b) Final replica mapping after adding a third data center

Fig. 1. Replica Relocation

When a new data center is added to the infrastructure (i.e., DC3), replicas are relocated again according to the *as-unique-as-possible* algorithm used by Swift [28]. Fig. 1(b) shows the optimal locations of the replicas according to the *as-unique-as-possible* algorithm. During this relocation, two issues could arise. Firstly, the amount of exchanged data to create the replicas could be huge and could overload the network. Secondly, the replicas that are not yet created or are in the process of being created are unavailable, and thus cannot process clients' requests. Indeed, the management tool that directs user requests to the appropriate locations of data should have an updated view of all replica placements. In Swift, the new placement is used to direct clients' requests, even before the migration

TABLE 1
Comparison of replica migration schemes

	S. Khuller et al. [23]	T. Tziritas et al. [24]	T. Lookopoulos et al. [26]	T. Lookopoulos et al. [27]	T. Tziritas et al. [25]	OpenStack Swift [8]	CRANE
Assumptions							
Object Size	Equal	Different	Equal	Different	Different	Different	Equal
Links	Equal capacity	Different capacities	Equal capacity but different cost	Equal capacity but different cost	Different capacities	Paths between servers have overlapping links with different capacities	Paths between servers have overlapping links with different capacities
Bypass Nodes	Using bypass nodes to minimise migration time	No	No	No	No	No	No
Goals							
Minimize migration time	Approximation algorithms for broadcating and gossiping problems	Heuristics scheduling transfers with the earliest completion time or the smallest hop count	No	No	Heuristics using multicast trees	No	Choosing replica with earliest completion using the path capacity.
Minimize migration cost	No	No	Minimizing network usage	Heuristics for minimizing network cost, scheduling deletion and migration of replicas	No	No	No
Ensure minimum data availability	No	No	No	No	No	Recalculate replica placement each 1-hour, and change the placement of one replica of same partition at each new placement	Use parallel migrations that ensure the minimum availability of data

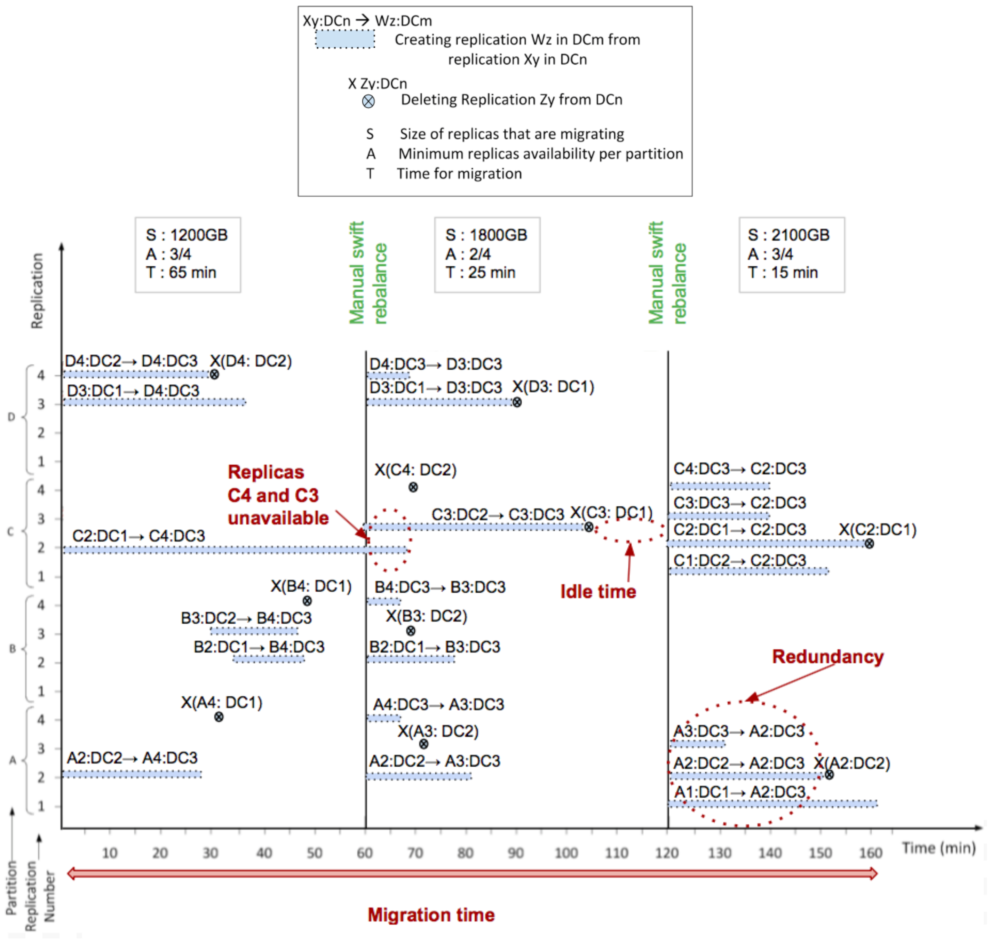
finishes. That is, the management tool becomes oblivious to the old placement of replicas. Therefore, some client requests may be directed to the new placement, even if some replicas haven't yet wholly arrived at their final destination, thus negatively impacting availability. Moreover, to ensure availability of data during migration, the management tool limits the number of migrating replicas of each data for a time interval. Indeed, a new placement of replicas is computed after 1-hour delay to move only one replica of each data in the respective interval, with the assumption that the availability of replicas will be ensured (i.e., all clients' requests will be accommodated).

Thus, Swift tries to satisfy data availability requirement by ensuring that at least 3 out of 4 replicas are available at any time. Fig. 2(a) shows the replica migration sequence using Swift. The x-axis represents time and the y-axis shows the replica identifiers of each partition. We can see in the figure that migrations are triggered every hour and only one replica of each partition can be migrated at a time during the beginning of the time interval.

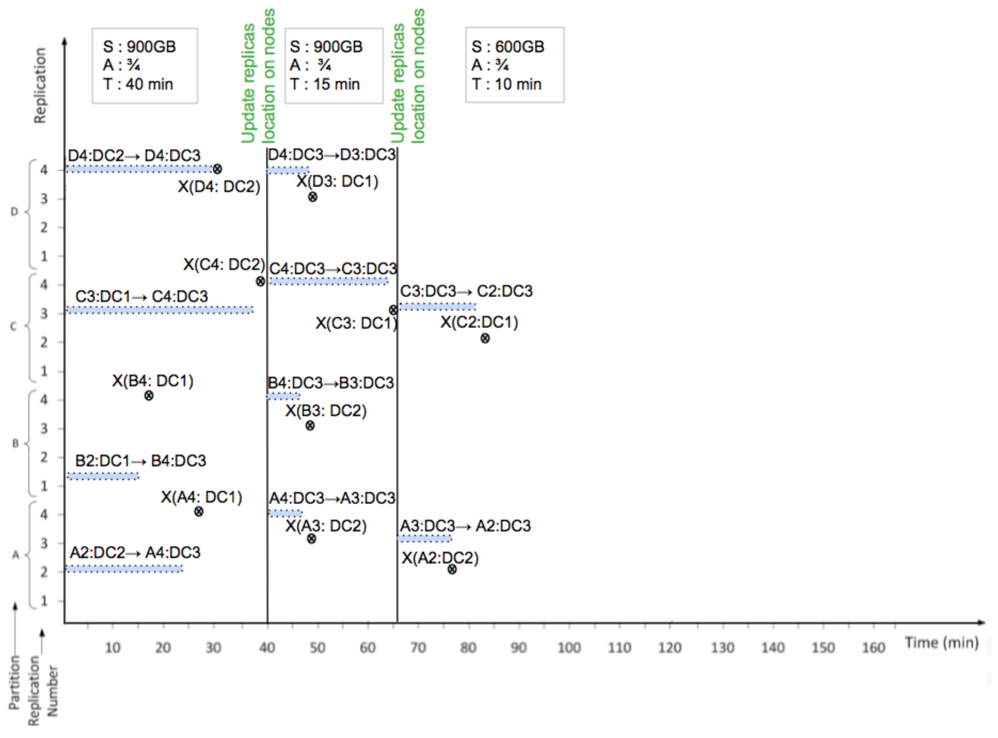
However, using the swift replica creation and migration scheme, the following metrics could be affected:

• *Availability:* The time interval separating the migrations of the same partition replicas is a parameter defined by the storage provider (by default it is set to 1 hour in Swift), which is not optimal. For instance, if two replicas of the same partition need to be migrated, Swift migrates the first one in the current time interval and the second one in the following time interval. However, if the migration of the first replica takes more than one hour, there will be two replicas being migrated at the same time and hence will not be available to process clients' requests. Fig. 2(a) shows how this happens in the studied example. At $t = 0$, Swift is creating the replica C4 (hosted in DC3) and it takes more than one hour to copy the data (8 minutes more as shown in the figure). During this time, C4 is not available. At the second time interval (i.e., at $t = 1$ hour), C3 starts being migrated and hence becomes unavailable. As a result, replicas C3 and C4 are both not available during the first 8 minutes of the second time interval and only the other 2 replicas (C1 and C2) are available. This clearly violates the replica availability requirement for the partition C (i.e., 3 replicas out of 4).

• *Redundancy and migration time:* Replica migration



(a) Swift replica migration sequence



(b) An example of a better replica migration sequence

Fig. 2. Replication migration sequences

in Swift is a peer-to-peer asynchronous and distributed process. Each storage server that holds a replica of a partition will sequentially check all the other replicas and update them if necessary. As a result, when a new replica is created, several other replicas might start sending data at the same time to the new location. This redundancy consumes additional bandwidth, which might overload the network. Fig. 2(a) shows how replica D4 is created using data copied from replicas D3 (located at DC1) and D4 (located DC2) at the same time (at $t = 0$). Ideally, redunduncy should be avoided and the source of the data could be selected based on the available bandwidth and the propagation delay between data centers in order to reduce migration time and the bandwidth consumption.

• *Idle time:* To ensure data availability, Swift restricts the number of replicas of the same partition that could be moved during one time interval to one replica. However, the migration of the replica can finish in less than one hour and hence, the system remains idle until the next time interval, as shown in Fig. 2(a). This increases the overall time needed to create and migrate replicas and to reach the new mapping configuration. As a result, an efficient migration sequence should avoid this idle time while ensuring the availability of the replicas by carefully sequencing their migration.

To avoid the aforementioned drawbacks, we need to i) avoid the redundancy when copying replicas to decrease bandwidth consumption, ii) carefully select the source replicas in order to reduce the migration time, iii) ensure the replica availability during the data migrations (i.e., 3 replicas out of 4 should be available at any time), and finally iv) avoid idle times in order to reduce the overall migration time. Fig. 2(b) shows an example of a migration sequence that achieves these objectives and reaches the final replica mapping with a migration time less than 50% of the one achieved by Swift.

In this paper, we propose CRANE a novel replica migration solution able to generate near-optimal replica migration sequences that achieve the sought-after objectives. In the following, we start first by mathematically formulating the problem before presenting the details of CRANE.

4 The Replica Migration Problem

The replica migration problem aims to find an optimal sequence of migrations from an initial to a final placement. As each partition has several stored replicas across the servers, the optimal migration sequence should select the best sources for migration. The goal is to minimize the replica migration time while meeting the minimum partition availability threshold A and abiding by links bandwidth capacity. Moreover, we assume that servers perform continuous sequential migrations.

Availability threshold is the minimum number of replicas that should not be migrating simultaneously. That is, if there are more than A replicas that are not migrating, the data is considered to be available. This is dictated by the fact that a management tool responsible for redirecting user requests to the appropriate replica placement only knows the final placement of replicas. Thus, the replica will be available only when it has entirely arrived at its final destination.

On the other hand, the topology that connects the storage devices is composed of different capacity links that are only dedicated to the migration process. Separating management network from "users" network has been considered by many cloud providers in order to improve the security and the performances of their system [15], [29], [30].

4.1 Problem Statement

Given a network represented by a graph $G = (S,\ E)$, where $S = \{s_1, s_2, ..., s_i, ..., s_k, ..., s_{|S|}\}$ is the set of all servers across data centers. We assume that data centers are connected through a backbone network. The backbone's links are represented by a set of edges E. Each edge $e \in E$ is characterized by a bandwidth capacity B_e. Let $P = \{p_1, p_2, ..., p_j, ..., p_{|P|}\}$ denote the set of partitions, replicas of which are stored across the servers where $|p_j|$ is the size of replica of partition p_j. We define a configuration as a particular placement of the replicas of partitions within servers. Given an initial and a final configurations denoted respectively by C^I and C^F, any discrepancy in the configurations necessitates either the migration or the deletion of the partition replicas. Consider that the migration or deletion of the replica is identified by variables $y_{j,k}$ and $d_{k,j}$, respectively. Our goal is to find the optimal sequence of replica migrations that minimizes each replica migration time and the total migration time from C^I to C^F while meeting the minimum partition availability threshold A and abiding by edge bandwidth capacities. We model this as an Integer Linear Programming (ILP) problem. Tables 2 and 3 show respectively the inputs of the ILP and its variables.

4.2 Constraints

Before we can initiate the migration, we identify the servers s_i that hold the replica of partition p_j at time t using the variable $z_{i,j,t}$. In our model, only the servers s_i that hold a replica of partition p_j can participate to the replica migration.

Furthermore, we consider that if a server s_i holds a replica of the partition p_j in the initial configuration, this replica cannot be deleted and is kept alive during all time instances in the considered time span $[1, T-1]$ except the last time instance (i.e., T), as shown in constraint (1). The goal of this constraint is to provide the model with more possibilities when selecting the source as it allows to use the server s_i as a source to create new replicas of p_j even if this replicas has to be deleted to reach the new configuration.

$$c^I_{i,j} \leq \beta \cdot z_{i,j,t} \quad \forall\ 1 \leq i \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T-1 \tag{1}$$

In constraint (2), we ensure that a server s_i cannot participate in the migration process of p_j, if it does not hold a replica of p_j in the initial or in the final configuration.

$$z_{i,j,t} \leq c^I_{i,j} + c^F_{i,j} \quad \forall 1 \leq i \leq |S|, 1 \leq j \neq 0 \leq |P|, 1 \leq t \leq T \tag{2}$$

TABLE 2
Problem inputs

Input	Definition
S	Set of servers across data centers, where $S = \{s_1, s_2, ..., s_i, ..., s_k, ..., s_{\|S\|}\}$
E	Set of edges connecting servers in S
B_e	Bandwidth capacity $\forall e \in E$
P	Set of partitions, where $P = \{p_1, p_2, ..., p_j, ..., p_{\|P\|}\}$ and $\|p_j\|$ is the size of partition p_j
C^I	$\|S\| \times \|P\|$ matrix representing an initial configuration, where $c^I_{i,j} = \begin{cases} 1, \text{ if replica of } p_j \text{ is stored on } s_i \\ 0, \text{ otherwise} \end{cases}$
C^F	$\|S\| \times \|P\|$ matrix representing a final configuration, where $c^F_{i,j} = \begin{cases} 1, \text{ if replica of } p_j \text{ is stored on } s_i \\ 0, \text{ otherwise} \end{cases}$
Y	$\|P\| \times \|S\|$ matrix representing a need for partition migration, where $y_{j,k} = \begin{cases} 1, \text{ if replica of } p_j \text{ needs to be migrated to server } s_k \\ 0, \text{ otherwise} \end{cases}$
A	Availability threshold, the minimum number of replicas to be kept stable
M	Total number of replicas per partition, a Swift parameter set by the system administrator ($M > A$)
T	Worst-case migration time
G	$\|S\| \times \|S\| \times \|E\|$ matrix representing edges used in a path, where $g_{i,k,e} = \begin{cases} 1, \text{ if edge } e \text{ is used in shortest path between } s_i \text{ and } s_k \\ 0, \text{ otherwise} \end{cases}$
α	A unit time
β	A big constant

TABLE 3
Problem variables

Variable	Definition
X	$\|S\| \times \|P\| \times \|S\| \times T$ matrix representing migration sequence, where $x_{i,j,k,t} = \begin{cases} 1, \text{ if } s_i \text{ is migrating replica of } p_j \text{ to } s_k \text{ at time } t \\ 0, \text{ otherwise} \end{cases}$
Z	$\|S\| \times \|P\| \times T$ matrix representing replica placement, where $z_{i,j,t} = \begin{cases} 1, \text{ if } s_i \text{ has replica of } p_j \text{ at time } t \\ 0, \text{ otherwise} \end{cases}$
R	$\|S\| \times \|P\| \times \|S\| \times T$ matrix, where $r_{i,j,k,t}$ represents the bandwidth allocated for migrating replica of p_j from s_i to s_k at time t
W	A vector of size T, where $w_t = \begin{cases} 1, \text{ if migration is in progress at time } t \\ 0, \text{ otherwise} \end{cases}$

The variable $z_{k,j,t}$ that indicates potential sources of replicas is updated in each time instance t, as servers s_k may begin to hold a copy of the replica of partition p_j, due to migration in earlier time instances, if this server hasn't this replica in initial configuration, as in constraints (3) and (4). A server holds a copy of the replica when the sum of all the bandwidth allocated, $r_{i,j,k,t'}$ to the migration of that replica of partition p_j from source s_j to destination s_k, in previous time instances $\forall t' < t$, equals the size of the partition p_j. Then, in following time instances, server s_k could potentially participate in the replica migration.

$$|p_j| - \sum_{i=1, i\neq k}^{|S|} \sum_{t'=1}^{t} r_{i,j,k,t'} \cdot \alpha \leq \beta \cdot (1 - z_{k,j,t+1})$$
$$\forall 1 \leq k \leq |S|, 1 \leq j, c^I_{k,j} = 0 \leq |P|, 1 \leq t \leq T-1 \quad (3)$$

$$1 - z_{k,j,t+1} \leq |p_j| - \sum_{i=1, i\neq k}^{|S|} \sum_{t'=1}^{t} r_{i,j,k,t'} \cdot \alpha$$
$$\forall 1 \leq k \leq |S|, 1 \leq j, c^I_{k,j} = 0 \leq |P|, 1 \leq t \leq T-1 \quad (4)$$

Once the model has been initialized with potential providers, we can initiate migration by associating the replica of partition migration indicator variable $x_{i,j,k,t}$ with need for migration of replica of partition p_j from s_i to s_k, in $y_{j,k}$, in constraint (5).

$$y_{j,k} \leq \sum_{t=1}^{T} x_{i,j,k,t} \quad \forall 1 \leq i,k,\ i \neq k \leq |S|, 1 \leq j \leq |P| \quad (5)$$

Furthermore, in constraint (6), we bind the source server s_i, such that, only those servers that hold complete replica of partition p_j can initiate migration.

$$\sum_{k=1}^{|S|} x_{i,j,k,t} \leq z_{i,j,t} \quad \forall 1 \leq i \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T \quad (6)$$

We ensure only sequential migration of replicas, since concurrency is set to 1. Constraint (7), ensure that each server can migrate only one replica at each time t.

$$\sum_{j=1}^{|P|} \sum_{k=1}^{|S|} x_{i,j,k,t} \leq 1 \quad \forall 1 \leq i \leq |S|, 1 \leq t \leq T \quad (7)$$

To ensure continuous sequential migration of replica of partition p_j, from the same source server s_i to the same destination server s_k for next time instance $t+1$, we set the indicator of migration is progress, in variable $x_{i,j,k,t+1}$, to 1, until the entire replica of the partition has been migrated. This is depicted in constraint (8).

$$|p_j| - \sum_{t'=1}^{t} r_{i,j,k,t'} \cdot \alpha \leq \beta \cdot x_{i,j,k,t+1}$$
$$\forall 1 \leq i,k,\ i \neq k \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T-1 \quad (8)$$

We also denote by A the minimum number of replicas to be kept stable (i.e., they should not be migrating at any given time t) in order to ensure the availability requirement. In other words, during any time t, the number of replicas of a particular partition that could be migrating should be less than or equal to $(M - A)$, where M is the total number of replicas per partition. This is depicted in constraint (9).

$$\sum_{i=1}^{|S|} \sum_{k=1}^{|S|} x_{i,j,k,t} \leq M - A \ \forall 1 \leq j \leq |P|, 1 \leq t \leq T \quad (9)$$

Furthermore, we need to ensure that the total bandwidth allocated for migrating replica of partition p_j from server s_i to server s_k does not exceed the size of the partition p_j, in constraint (10).

$$\sum_{t=1}^{T} r_{i,j,k,t} \cdot \alpha \leq y_{j,k} \cdot |p_j| \quad \forall 1 \leq i \leq |S|, 1 \leq t \leq T \quad (10)$$

The load on an edge, not exceeding edge capacity, is conjured as the sum of all partition replicas migrating in the network across all source and destination servers at time t, on edge e in the shortest path between s_i and s_k, by constraints (11).

$$\sum_{i=1}^{|S|}\sum_{j=1}^{|P|}\sum_{k=1}^{|S|} g_{i,k,e} \cdot r_{i,j,k,t} \leq B_e$$
$$\forall 1 \leq e \leq |E|, 1 \leq t \leq T \quad (11)$$

The total migration time is extended to include all migrations in progress in constraint (12) and stopping the migration process in constraint (13).

$$x_{i,j,k,t} \leq w_t$$
$$\forall 1 \leq i,k,\ i \neq k \leq |S|, 1 \leq j \leq |P|, 1 \leq t \leq T \quad (12)$$

$$w_{t+1} \leq w_t \ \ \forall 1 \leq t \leq T-1 \quad (13)$$

4.3 Objective

$$minimize \left(\sum_{t=1}^{T} w_t + \sum_{i=1}^{|S|}\sum_{j=1}^{|P|}\sum_{k=1}^{|S|}\sum_{t=1}^{T} x_{i,j,k,t} \right) \quad (14)$$

In order to have a migrating replica available as quickly as possible on the destination server, we minimize the migration time for each replica. Furthermore, to ensure that migrations are triggered as soon as possible, we minimize the total migration time. These objectives are depicted in (14). As the optimization minimizes migration times, it will select source servers for replica migration that reduce migration time, such that it selects source-destination pairs that have minimum overlapping edges in the shortest path, while ensuring sequential migrations, meeting minimum partition availability threshold and abiding by edge bandwidth capacities.

5 Solution Design

In this section, we will describe CRANE, our heuristic solution for the replicas migration problem. Given an initial and a target replicas mapping in data centers, the goal of this algorithm is to find the best sources for copying the replicas and the best sequence to send them so as to minimize the total replica creation/migration time. To this end, the following sights can guide the replica creation/migration sequence: (1) avoid redundancy, (2) select the source of data and paths having more available bandwidth, and (3) avoid idle time between sequences.

Our heuristic solution is described in Algorithm 1. Given an initial and target placement configurations (i.e., C^I and C^F), Algorithm 1 returns a set Q of sequences Q_i for migration. Each sequence contains an ordered set of replicas to be migrated/created such that the required minimum availability per partition is satisfied. After each sequence of migrations Q_i, cloud storage components will be updated with the new placement, so that data user requests can be redirected to the right partition locations. The final replica placement is reached once all the sequences $Q_i, i < n$ are executed.

Algorithm 1 CRANE

```
require: Initial configuration $C^I$
require: Final configuration $C^F$
output: Sequence for migration

1:  $P \leftarrow \{(p,d)\}$
2:  $Q \leftarrow \{\emptyset\}$
3:  $i \leftarrow 0$
4:  while $P \neq \{\emptyset\}$ do
5:      $Q_i \leftarrow \{\emptyset\}$
6:      $P_i \leftarrow \{P\}$
7:      while $P_i \neq \emptyset$ do
8:          $T_{Q_i,min} \leftarrow \infty$
9:          for each $(p,d) \in P_i$ / $\{migrating(p) < M - A\}$ do
10:             for each $r_{src} \in R_p$ / $\{is\_busy(src, Q_i) = False\}$
                do
11:                 if $T_{Q_i,r_{src}} < T_{Q_i,min}$ then
12:                     $T_{Q_i,min} = T_{Q_i,r_{src}}$
13:                     $r_s = r_{src}$
14:                     $d_s = d$
15:                 end if
16:             end for
17:         end for
18:         $Q_i = Q_i \cup (r_s, d_s)$
19:         $P_i = P_i - \{\forall(p,d)$ / $p$ is partition of replica $r_s\}$
20:         $P = P - \{$partition $p$ of replica $r_s$ to destination $d\}$
21:     end while
22:     $Q = Q \cup Q_i$
23:     $i \leftarrow i + 1$
24: end while
25: return Q
```

Initially, P contains the set of couples (p, d), where p is a partition that needs to be created/migrated in the destination server d. This set can be computed based on the initial and the final partition locations (i.e., C^I and C^F). We then initialize Q that should contain the sequence of replicas to be migrated. In line 3, we initialize a variable i that denotes the number of the sequence. The core of the algorithm aims to iteratively add a partition replica on ordered sequence Q_i. We create a new sequence whenever it is not possible anymore to add a replica creation/migration to the current sequence (not possible because otherwise we do not satisfy the minimum replica availability per partition). P_i represents the set of couples (p, d) that can be created in the sequence Q_i without violating the minimum required availability for replicas.

As long as P_i contains partitions to be created we can still add a replica in a sequence Q_i. However, this should not violate the required availability. For that, we iterate over partitions that have less than $M - A$ migrating replicas, where M is the total number of replicas per partition and A is the number of replicas to be kept stable. Then, we iterate over all replicas in R_p, where R_p is the set of replicas of partition p and the server containing the replica r is not copying another replica at the same time (line 10).

We choose the replica r_s and the destination server d_s that minimize the migration time. For that we use the variable $T_{Q_i,r_{src}}$ that denotes the time to migrate the sequence Q_i if replica r_{src} is included. $T_{Q_i,r_{src}}$ is computed considering links states during the migration of all replicas in the sequence Q_i. We compare this variable to the $T_{Q_i,min}$ variable that denotes the minimum migration time of sequence Q_i after adding a replica (line 11). This allows us to select the best replica r_s of the partition p from all replicas of all partitions (line 13). The chosen replica and destination server are then added to Q_i (line 17). The couple (p, d_s) is removed from the set P, where p is the partition whose replica r_s was selected and d_s is the destination server where the partition will be created. To ensure availability, we remove all couples (p, d), where p is the partition whose replica r_s was selected. Thus, we ensure that only one replica of the same partition is migrated in the same sequence.

When P_i doesn't contain any more couples to migrate, we detect that we cannot add any more replica to the sequence Q_i. At that time, we add the sequence to Q (Line 22), and start a new one as long as we still have partitions in P to migrate/create. The time complexity of this algorithm is $O(|P|^2)$, where $|P|$ is the number of partitions.

6 Performance Evaluation

We implemented CRANE using Python 2.7 and we evaluated its performances in two phases. We analytically compared the performances of CRANE, OpenStack Swift and the optimal replica migration schedule. Then we conduct experiments to compare performance of crane with OpenStack Swift performances.

The main performance metric for replica migration between geographically distributed data centers is latency [12], [15], [16], which is derived from link capacity. Moreover, in [31] and [32] authors have empirically studied the network performance features of data centers in a geo-distributed cloud environment. They observe that the cross-region network performance (including both bandwidth and latency) is often highly related to the geographic distance between the regions. This was explicitly demonstrated through measurements performed on EC2 instances. Therefore, to evaluate our heuristic performances, we have used Amazon EC2 inter-data center network. This topology is composed of 7 data centers that are all connected to each other. We have also used links capacities measured in [31] through real-world experiments, in order to have a realistic view of the network state. Table 4 describes these links capacities.

We considered several scenarios, each having a different number of partitions and different average partition sizes. In the beginning of each experiment, we consider only 5 data centers. After that, two new data centers are connected to the infrastructure, which triggers the placement algorithm in order to re-optimize the location of replicas. For replicas placement, we have used the standard Swift algorithm stipulating that for each data partition, three replicas have to be created and placed according to the as-unique-as possible algorithm.

Furthermore, the required availability is assumed to be $\frac{A}{M}$, where A is the minimum number of replicas to be kept stable and M is the total number of replicas per partition. To ensure availability, Swift restricts the migration of one replica in the same sequence. This assumes if 2 out of the 3 replicas are available, the data is considered to be available (i.e., all user requests can be accommodated). Hence, the minimum required availability per partition is set to 2/3.

Our experimental results have validated that, by optimally scheduling replicas to migrate, CRANE reduces migration time and the amount of data to migrate.

6.1 Analytical Results

For analytical results, we have modeled as an Integer Linear Programming (ILP) problem the special case of replica migration problem that minimizes migration time while ensuring minimum replicas availability per partition. We have implemented this model with AMPL [11] language. We used IBM CPLEX Optimizer [10] to find the optimal migration plan that reduces migration time and avoids redundancy. We considered eight scenarios as depicted in Table 5. For instance, we consider that we have 3 replicas for each partition. In the first four scenarios, partitions size vary between 500 Mb and 1 Gb. For the four last scenarios, partitions size vary between 1 Gb and 5 Gb. We distribute these partitions across the 5 data centers, this describes the initial replica placement. Then, we add two new data centers, we trigger replica placement algorithm to re-optimize replica placement. Considering these initial and final placements, we emulate Swift and CRANE operations and compare their theoretical performances to the optimal solution for each scenario.

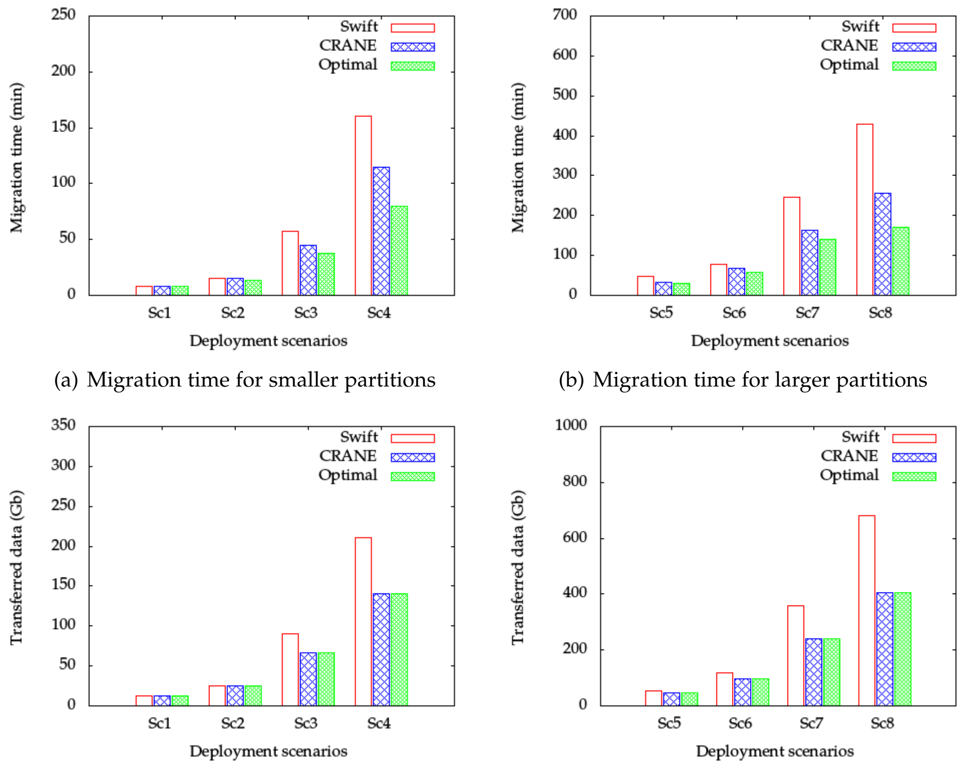
To ensure high partitions availability, OpenStack Swift relocates only one replica of a partition at each new replica placement. Then, it needs to compute a new replica placement to relocate other replicas of these partitions if needed. This can be executed after more than one hour. Having bulk data transfers, we have evaluated performances of executing the placement algorithm after one hour and after two hours, for Swift migrations. The execution of the placement algorithms computes the new placement of replicas and trigger the migration algorithm to relocate replicas to their new location.

Fig. 3(a) and Fig. 3(c) represents migration time and the amount of transferred data respectively for scenarios Sc1, Sc2, Sc3 and Sc4. For Sc1, Swift, CRANE and the optimal replica migration plans, have taken 8 min to create 16 new replicas. Moreover, the amount of transferred data is the same. For Sc2, to migrate 32 partitions, the total amount of transferred data is 25 Gb. The optimal replica migration plan has taken 2 minutes less than CRANE and Swift replica migration plans. For Sc3, CRANE and the optimal replica migration plans, have the same amount of transferred data, whereas Swift transfers 30% more data. The optimal algorithm performs 15% better than CRANE, and 35% better than Swift. For Sc4, Swift transfers 40% more data than CRANE and the optimal migration plan, in twice the optimal migration time. However, CRANE has an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCC.2018.2858792, IEEE Transactions on Cloud Computing

TABLE 4
EC2 inter-data center link capacities

Link capacity (Mbps)	North California	Oregon	Virginia	Sao Paulo	Ireland	Singapore	Tokyo
North California	-	520	252	116	98	103	173
Oregon	545	-	215	81	104	81	152
Virginia	240	210	-	139	221	81	110
Sao Paulo	40	60	11	-	22	9	62
Ireland	106	135	215	90	-	77	76
Singapore	125	110	84	57	80	-	242
Tokyo	178	143	99	61	43	116	-



(a) Migration time for smaller partitions

(b) Migration time for larger partitions

(c) Transferred data for smaller partitions

(d) Transferred data for larger partitions

Fig. 3. Analytical performance comparison between optimal migration plan, CRANE and traditional Swift

TABLE 5
Deployment scenarios

Scenario	Number of partitions	Number of replicas to migrate	Average replica size (Mb)	Replica placement computation
Sc1	16	16	750	1 hour
Sc2	32	32	750	1 hour
Sc3	64	80	750	1 hour
Sc4	128	160	750	1 hour
Sc5	16	16	3074	2 hours
Sc6	32	32	3074	2 hours
Sc7	64	80	3074	2 hours
Sc8	128	160	3074	2 hours

optimal amount of transferred data and have also improved migration time by 30% compared to Swift.

Fig. 3(b) and Fig. 3(d) represent the migration time and the amount of transferred data respectively for scenarios Sc5, Sc6, Sc7 and Sc8. The scenarios have bigger replicas size varying from 1Gb to 5Gb. Fig. 3(d) shows that CRANE has an optimal amount of transferred data, for the 4 scenarios. However, Swift transfers 15% more replicas for Sc5 and Sc6, and 35% and 45% more data in Sc7 and Sc8, respectively. This have also induced higher migration time, 35% higher than the optimal migration plan for the Sc5 and Sc6, and 45% and 60% higher than the optimal migration plan for the Sc7 and Sc8, respectively. In the other hand, CRANE achieves replica migration times 15% higher than the optimal migration plan for Sc5 and Sc6, around 20% for Sc7 and 30% for Sc8.

From these different scenarios, we can conclude that CRANE migration time is around 20% close to the optimal migration plan time. And achieves around 40% improvement compared to the Swift migration time. That is, the time needed to migrate replicas have been highly improved.

6.2 Experimentation

6.2.1 Setup environment

OpenStack Swift requires at least one *proxy node* and several *storage nodes*. The *proxy node* accepts requests to upload files, modify metadata, and create containers. The *storage nodes* manage objects and containers in a distributed manner. For our experiments, we would be interested in replicas migration. We have used the open-source software platform for cloud computing OpenStack [33] to create the infrastructure. Thus, we consider that each *storage node* is an instance hosted in one data center. Indeed, we have launched 8 Ubuntu 14.04 medium instances with 2 compute units and 2 Gb memory. We have attached a 200Gb disk to each storage instance. Each instance is launched in a different network linked with a virtual router. Then we have fixed bandwidths on the router's interfaces as described in Table 4, with Linux commands.

In the beginning of each scenario, we consider only 5 data centers. After that, two new data centers are connected to the infrastructure, which triggers the Swift placement algorithm in order to re-optimize the location of replicas. Then we use either Swift migration algorithm or CRANE algorithm to migrate replicas to the computed optimal locations. We deployed four scenarios considering 64 partitions with 3 replicas each one like recommended by some OpenStack providers [34]. And, we varied the average size of replicas. Table 6 depicts these scenarios.

TABLE 6
Experimental deployment scenarios

Scenario	**Number of partitions**	**Average replica size (Gb)**
Sc9	64	3
Sc10	64	10
Sc11	64	15

For each scenario, depicted in Table 6, we compare CRANE with traditional Swift with respect to the following performance metrics: (1) the total migration time, (2) the amount of inter-data centers exchanged data, (3) the idle time and (4) the availability of replicas per partition. We considered two Swift configurations, one computing the replica placement each hour and the other each two hours.

6.2.2 Results

Fig. 4(a) shows the total migration time for the considered scenarios. As we can see, in all scenarios, CRANE outperforms the Swift algorithm by a good margin for both configurations. For Sc9, CRANE takes 50 minutes to create all the replicas compared to 75 minutes for the 1-hour Swift algorithm and 132 minutes for the 2-hours Swift algorithm. This constitutes around 35% and 65% of improvement comparing to the 1-hour Swift algorithm and to the 2-hours Swift algorithm respectively. For Sc10 and Sc11, CRANE also achieves 30% and 50% improvements compared to the 1-hour and the 2-hours Swift configuration. This obtained improvement can be explained by the fact that CRANE always chooses to copy the replica incurring the minimal transmission time. As migration time depicts the time that takes the servers to transfer all replicas to their new placement, it's also influenced by the idle time. As depicted in Fig. 4(c), idle time for CRANE is always zero.

As CRANE compose parallel replica migrations that ensure minimum replica availability and migrate them sequentially and directly, it ensures minimum replica availability and avoids idle time. However, the idle time is higher for the 1-hour and 2-hours Swift configurations. Indeed, for Sc9, we see that the 2-hours Swift configuration, idle time is 85 minutes. As the amount of data can be transferred rapidly, the system will be idle for 85 minutes waiting for the new replica placement. During this time, the replicas aren't in their optimal locations so the system is performing sub-optimally. The more the amount of transferred data is high, the more the idle time is lower.

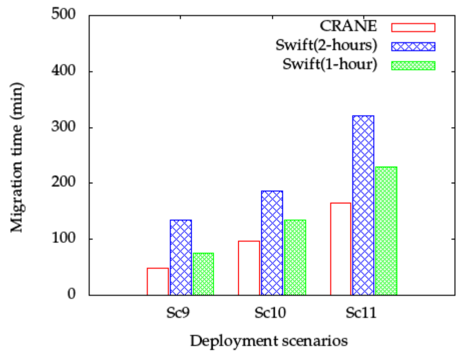
The amount of inter-data centers transferred data is reported in Fig. 4(b). For the 3 scenarios, the CRANE algorithm minimizes that data amount to be transferred. The improvement is around 25% with the 1-hour Swift configuration and 45% with the 2-hours Swift configuration. This improvement is explained by the avoidance of redundant copy of the replicas of the same partition. This is also explained by the fact that Swift computes at each time a new replica placement. At each replica placement the replicas are as-far-as possible from each other and distributed in a same way over the disks. This will induce needless migrations that are just triggered in order balance the replicas on disks and make the replicas of each partition far from each other, even if this placement isn't the optimal one. This have also induced the improvement in migration time showed in Fig. 4(a).

Finally, Fig. 4(d) shows the Inverse Cumulative Distribution Function (ICDF) of the availability. For a given availability, it provides the probability of having that availability or higher. The required minimum availability per partition (2/3 = 0.66) isn't met for the 1-hour Swift configuration. The probability of having an availability higher than 66% is 0.98. The 1-hour that separate the two new replica placement haven't been enough to ensure the availability. Indeed, there are two replicas of the same partition that are not available. However, the probability of having a high availability is always higher for the CRANE algorithm than the traditional Swift. For instance, the probability of having an availability higher than 80% is 0.60 for Swift whereas it is around 0.76 for CRANE. If compare the three curves, we can see that, on average, CRANE improves availability by up to 10%.

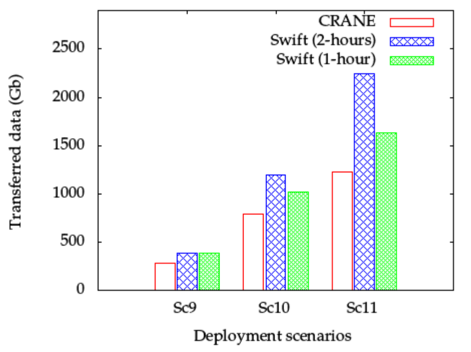
It is clear that CRANE performs significantly better than the basic Swift algorithm as it carefully selects the replica from which the data should be copied, the paths used to transmit that data while avoiding the redundant copy of replicas and eliminating idle time.

7 Conclusion

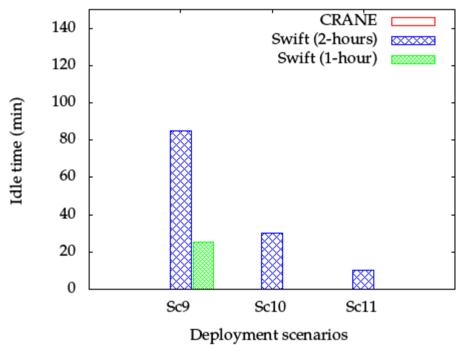
Data replication has been widely adopted to improve data availability and to reduce access time. However, replica placement systems usually need to migrate and create a



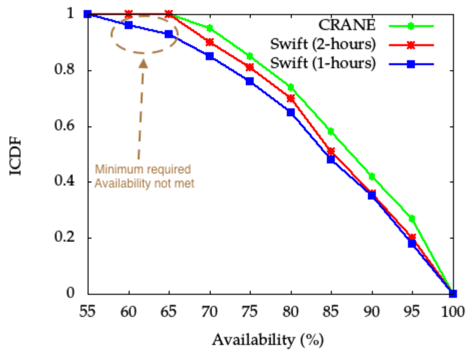
(a) Migration time



(b) Amount of transferred data



(c) Idle time



(d) Availability

Fig. 4. Experimental performance comparison between CRANE and traditional Swift

large number of replicas between and within data centers, incurring a large overhead in terms of network load and availability. In this paper, we proposed CRANE, an effiCient Replica migrAtion scheme for distributed cloud Storage systEms. CRANE complements replica placement algorithms by efficiently managing replica creation by minimizing the time needed to copy data to the new replica location while avoiding network congestion and ensuring the required availability of the data. In order to evaluate the performance of CRANE, we compare it to the optimal solution for the replica migration problem considering availability and to the standard swift, the OpenStack project for managing data storage. Results show that CRANE has sub-optimal performances in terms of migration time and an optimal amount of transferred data. Moreover, experiments show that CRANE is able to reduce up to 60% of the replica creation time and 50% of inter-data center network traffic and provide better data availability during the process of replica migration. In our future work, we will perform larger scale simulations to further scrutinize the performance of our heuristic. Other improvements will also considered to address reliability and consistency requirements.

8 Acknowledgement

This work was supported in part by the Quebec FRQNT postdoctoral research fellowship.

References

[1] B. A. Milani and N. J. Navimipour, "A comprehensive review of the data replication techniques in the cloud environments: Major trends and future directions," *Journal of Network and Computer Applications*, vol. 64, pp. 229–238, 2016.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, *The Google file system*. ACM, 2003, vol. 37, no. 5.

[3] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *IEEE Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[4] R.-S. Chang and H.-P. Chang, "A dynamic data replication strategy using access-weights in data grids," *The Journal of Supercomputing*, vol. 45, no. 3, 2008.

[5] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in *IEEE International Conference on Cluster Computing (CLUSTER),*, 2010, pp. 188–196.

[6] D.-W. Sun, G.-R. Chang, S. Gao, L.-Z. Jin, and X.-W. Wang, "Modeling a dynamic data replication strategy to increase system availability in cloud computing environments," *Journal of Computer Science and Technology*, vol. 27, no. 2, pp. 256–272, 2012.

[7] Y. Chen, S. Jain, V. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via Yahoo! datasets," in *IEEE INFOCOM*, April 2011, pp. 1620–1628.

[8] O. foundation. (2015) Swift documentation. [Online]. Available: http://docs.openstack.org/developer/swift/

[9] A. Mseddi, M. A. Salahuddin, M. F. Zhani, H. Elbiaze, and R. H. Glitho, "On optimizing replica migration in distributed cloud storage systems," in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*. IEEE, 2015, pp. 191–197.

[10] (2016) IBM CPLEX Optimizer. [Online]. Available: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/

[11] (2016) Ampl: Streamlined modeling for real optimization. [Online]. Available: http://ampl.com

[12] J. Hall, J. Hartline, A. R. Karlin, J. Saia, and J. Wilkes, "On algorithms for efficient data migration," in *Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2001, pp. 620–629.

[13] E. Anderson, J. Hall, J. Hartline, M. Hobbes, A. Karlin, J. Saia,

R. Swaminathan, and J. Wilkes, "Algorithms for data migration," *Algorithmica*, vol. 57, no. 2, pp. 349–380, 2010.

[14] C. Kari, Y.-A. Kim, and A. Russell, "Data migration in heterogeneous storage systems," in *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2011, pp. 143–150.

[15] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. Lau, "Orchestrating bulk data transfers across geo-distributed datacenters," *Cloud Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.

[16] C. Lu, G. A. Alvarez, and J. Wilkes, "Aqueduct: Online data migration with performance guarantees," in *FAST*, vol. 2, 2002, p. 21.

[17] D. L. Petrov and Y. S. Tatarinov, "Data migration in the scalable storage cloud," in *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on*. IEEE, 2009, pp. 1–4.

[18] Data Center Research. (2018) Colocation data centers. [Online]. Available: http://www.datacentermap.com/datacenters.html

[19] Synergy Research Group. (2017) Hyperscale data center count approaches the 400 mark. [Online]. Available: https://www.srgresearch.com/articles/hyperscale-data-center-count-approaches-400-mark-us-still-dominates

[20] "Cisco global cloud index: Forecast and methodology, 20162021," White Paper, Cisco, 2018.

[21] James Hamilton. (2010) Inter-datacenter replication and geo-redundancy. [Online]. Available: http://perspectives.mvdirona.com/2010/05/inter-datacenter-replication-geo-redundancy/

[22] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan, "Volley: Automated data placement for geo-distributed cloud services," 2010.

[23] S. Khuller, Y.-A. Kim, and Y.-C. J. Wan, "Algorithms for data migration with cloning," in *ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS, 2003.

[24] N. Tziritas, T. Loukopoulos, P. Lampsas, and S. Lalis, "Formal model and scheduling heuristics for the replica migration problem," in *Euro-Par 2008–Parallel Processing*. Springer, 2008, pp. 305–314.

[25] N. Tziritas, T. Loukopoulos, P. Lampsas, and S. Lalis, "Using multicast transfers in the replica migration problem: Formulation and scheduling heuristics," in *Euro-Par 2009 Parallel Processing*. Springer, 2009, pp. 228–240.

[26] T. Loukopoulos, N. Tziritas, P. Lampsas, and S. Lalis, "Implementing replica placements: feasibility and cost minimization," in *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*. IEEE, 2007, pp. 1–10.

[27] T. Loukopoulos, N. Tziritas, P. Lampsas, and S. Lalis, "Investigating the replica transfer scheduling problem," in *Proc. 18 th Int. Conf. on Parallel and Distributed Computing and Systems (PDCS06)*. Citeseer, 2006.

[28] J. Dickinson. (2013) Data placement in Swift. [Online]. Available: http://swiftstack.com/blog/2013/02/25/data-placement-in-swift

[29] Stephen Richardson. (2017) The pros and cons of a dedicated management network. [Online]. Available: https://www.ccexpert.us/network-management/the-pros-and-cons-of-a-dedicated-management-network.html

[30] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[31] Y. Feng, B. Li, and B. Li, "Jetway: Minimizing costs on inter-datacenter video traffic," in *Proceedings of the 20th ACM international conference on Multimedia*. ACM, 2012, pp. 259–268.

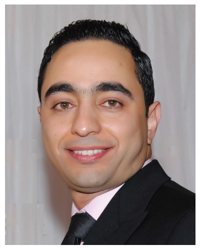
[32] A. C. Zhou, Y. Gong, B. He, and J. Zhai, "Efficient process mapping in geo-distributed cloud data centers," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 2017, p. 16.

[33] (2016) OpenStack cloud computing software. [Online]. Available: http://www.openstack.org

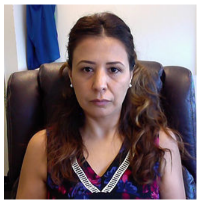
[34] Rackspace. (2016) Swift ring calculator. [Online]. Available: https://rackerlabs.github.io/swift-ppc/

Amina Mseddi is a master student with the Department of Computer Science, University of Quebec at Montreal. She received her Engineer degree in networks and telecommunications from Institut National des Sciences Appliquées et de Technologies in 2013. Her research interests include resource management in large-scale distributed systems, content delivery networks, cloud computing and software-defined networking.

Mohammad A. Salahuddin is a postdoctoral fellow at the David R. Cheriton School of Computer Science at the University of Waterloo, Waterloo, Ontario, Canada. He holds a Ph.D. in Computer Science from Western Michigan University (Kalamazoo, Michigan, USA–2014). His research interests include Wireless Sensor Networks, QoS and QoE in Vehicular Ad hoc Networks (WAVE, IEEE 802.11p and IEEE 1609.4), Internet of Things, Content Delivery Networks, Software-Defined Networking, Network Functions Virtualization and Cloud Resource Management. He serves as a Technical Program Committee member for international conferences and is a reviewer for various peer-reviewed journals, magazines and conferences.



Mohamed Faten Zhani is currently an assistant professor with the department of software and IT engineering at l'École de Technologie Suprieure (ÉTS) at the University of Quebec in Canada. He received his Ph.D. in Computer Science from the University of Quebec in Montreal, Canada in 2011. He then carried out his postdoctoral research at the David R. Cheriton School of Computer Science at the University of Waterloo. His research interests include cloud computing, virtualization, Big Data, software-defined networking and resource management in large-scale distributed systems.



Halima Elbiaze holds a Ph.D. in computer science and a M.Sc in Telecommunication systems from Institut National des Tlcommunications, Paris, France and Universit de Versailles in 2002 and 1998, and B.S. Degree in applied mathematics from university of MV, Morocco in 1996. Since 2003, she is with the Department of Computer Science, Universit du Qubec Montral, QC, Canada, where she is currently an Associate Professor. She is the author or coauthor of many journal and conference papers. Her research interests include network performance evaluation, traffic engineering, quality of service management in optical and wireless networks.

Roch Glitho [SM] holds a Ph.D. (Tekn. Dr.) in tele-informatics (Royal Institute of Technology, Stockholm, Sweden), and M.Sc. degrees in business economics (University of Grenoble, France), pure mathematics (University Geneva, Switzerland), and computer science (University of Geneva). He is an Associate Professor of networking and telecommunications at the Concordia Institute of Information Systems Engineering (CIISE), Concordia University, Montreal, Canada, where he holds a Canada Research Chair in End-User Service Engineering for Communication Networks. He is also an Adjunct Professor at Telecom Sud-Paris in France and at the University of Western Cape in South Africa. In the past he has worked in industry for almost a quarter of a century and has held several senior technical positions at LM Ericsson in Sweden and Canada (e.g. expert, principal engineer, senior specialist). In the past he has also served as IEEE Communications Society distinguished lecturer, Editor-In-Chief of IEEE Communications Magazine and Editor-In-Chief of IEEE Communications Surveys & Tutorials.