



HAL
open science

A data replication strategy with tenant performance and provider economic profit guarantees in cloud data centers

Riad Mokadem, Abdelkader Hameurlain

► **To cite this version:**

Riad Mokadem, Abdelkader Hameurlain. A data replication strategy with tenant performance and provider economic profit guarantees in cloud data centers. *Journal of Systems and Software*, 2020, 159, pp.110447. 10.1016/j.jss.2019.110447 . hal-02878939

HAL Id: hal-02878939

<https://hal.science/hal-02878939>

Submitted on 23 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Open Archive Toulouse Archive Ouverte



OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in: <https://oatao.univ-toulouse.fr/26278>

Official URL:

<https://doi.org/10.1016/j.jss.2019.110447>

To cite this version:

Mokadem, Riad  and Hameurlain, Abdelkader  *A data replication strategy with tenant performance and provider economic profit guarantees in cloud data centers.* (2020) *Journal of Systems and Software*, 159. 110447. ISSN 0164-1212.

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

A data replication strategy with tenant performance and provider economic profit guarantees in Cloud data centers

Riad Mokadem*, Abdelkader Hameurlain

Institut de Recherche en Informatique de Toulouse (IRIT Laboratory), Paul Sabatier University, Toulouse 31860, France

A B S T R A C T

Meeting tenant performance requirements through data replication while ensuring an economic profit is very challenging for cloud providers. For this purpose, we propose a data Replication Strategy that satisfies Performance tenant objective and provider profit in Cloud data centers (RSPC). Before the execution of each tenant query Q , data replication is considered only if: (i) the estimated Response Time of Q (RT_Q) exceeds a critical RT threshold (per-query replication), or (ii) more often, if RT_Q exceeds another (lower) RT threshold for a given number of times (replication per set of queries). Then, a new replica is really created only if a suitable replica placement is heuristically found so that the RT requirement is satisfied again while ensuring an economic profit for the provider. Both the provider's revenues and expenditures are also estimated while penalties and replication costs are taken into account. Furthermore, the replica factor is dynamically adjusted in order to reduce the resource consumption. Compared to four other strategies, RSPC best satisfies the RT requirement under high loads, complex queries and strict RT thresholds. Moreover, penalty and data transfer costs are significantly reduced, which impacts the provider profit.

Keywords:

Cloud systems
Databases
Data replication
Cost model
Economic profit
Performance

1. Introduction

1.1. Motivation

Data replication is a well known technique that consists in storing multiple copies of data, called replicas, at multiple nodes. It aims to increase data availability, reduce bandwidth consumption and achieve fault-tolerance. Data replication has been commonly used in traditional systems such as P2P (Spaho et al., 2015) and data grid systems (Tos et al., 2015). In such systems, a replication strategy needs to determine *what* to replicate? *When* to create/remove replicas? *Where* to place them? and *How many* replicas to create? (Ranganathan and Foster, 2001). However, most of the proposed replication strategies in the above systems are difficult to adapt to clouds since they aim to obtain better system performance without taking into account the *economic cost* of replication. In fact, creating as many replicas as possible in cloud systems cannot be economically feasible since it can result in wasteful resource utilization and reduced provider profit. Hence, data replication strategies in cloud systems should also achieve goals such as:

- (i) Providing a reliable Quality of Service (QoS) by satisfying a Service Level Agreement (SLA), a legal contract between a cloud provider and its tenants, i.e., customers (Buyya et al., 2009). Mainly, an SLA contains one or several tenant Service Level Objectives (SLO), i.e., requirements, to be satisfied by the provider.
- (ii) A dynamic adjustment of resources that the provider rents to its tenants, according to the 'pay as you go' pricing model (Armbrust et al., 2010).

Performance guarantees, e.g., response time (RT), are often not offered by cloud providers as a part of the SLA because of the heterogeneous workloads in cloud systems, e.g., Google Cloud SQL¹ only provides downtime and error guarantees without an RT guarantee. Thus, satisfying performance can often conflict with the goal of obtaining a maximum economic benefit at minimal operating costs (Lang et al., 2014). However, data replication strategies in such systems should consider the 'tenant performance/provider profitability' trade-off, especially when they are proposed for OLAP (Online Analytical Processing) applications as we do here. In other terms, the consistency management is not the focus of this work.

Most of the proposed replication strategies in the literature focus on ensuring a specific tenant objective, e.g., availability

* Corresponding author.

E-mail addresses: mokadem@irit.fr (R. Mokadem), hameurlain@irit.fr (A. Hameurlain).

¹ <https://cloud.google.com/sql/docs/>.

(Wei et al., 2010), reliability (Li et al., 2017), low latency (Ma and Yang, 2017), data durability (Liu et al., 2018), security (Ali et al., 2018) and energy efficiency (Boru et al., 2015) when other strategies balance among different objectives, e.g., availability and load balancing (Long et al., 2014). On the other hand, there are a number of strategies (Liu et al., 2018; Bonvin et al., 2011; Xiong et al., 2011; Mansouri et al., 2017; Edwin et al., 2017; Mansouri and Javidi, 2018) and replication frameworks (Pu et al., 2015) that aim to satisfy the tenant’s objectives while reducing the cost of replication, e. g., data storage and/or data transfer costs, between Data Centers (DCs). Some of them (Gill and Singh, 2016; Lin et al., 2013) are mentioned cost-aware although the considered cost of replication is not necessarily an economic cost. It is regarded as an assigned budget value for DCs in Gill and Singh (2016) and modelled in terms of time in Lin et al. (2013). In this context, only some strategies (Wu et al., 2013; Zeng et al., 2016; Tos et al., 2017; Casas et al., 2017; Liu and Shen, 2017; Mansouri and Buyya, 2019) model the replication cost and the provider profit as monetary costs while satisfying a tenant RT SLO. Furthermore, they often only consider a per-query replication that immediately responds to any query load or popularity change, but may generate higher RTs and increase overhead costs.

1.2. Proposal

We propose a data Replication Strategy that satisfies both Performance and minimum availability tenant objectives while ensuring an economic profit for the provider in Cloud data centers (RSPC). We consider a set of Virtual Machines (VMs) scattered over heterogeneous DCs, themselves distributed over different regions. Each VM has its own allocated storage volume and computational resources. Throughout this paper, a node refers to a VM.

Ensuring a given minimum availability SLO (SLO_{AV}) consists in initially creating, across regions, a minimum number of replicas for each data set, e.g., a database (DB) relation or an HDFS data block (Cheng et al., 2012), then maintaining them (Wei et al., 2010). Dealing with the performance guarantee, we focus on the RT metric. In this context, most of data replication strategies in the literature, e.g., (Mansouri et al., 2017), are based on data locality in order to reduce data access time, i.e., a replica is deployed at the user’s node or at the node very near to it (Ranganathan and Foster, 2001). In order to reduce the bandwidth consumption, RSPC benefits from the Network Bandwidth (NB) locality (Park et al., 2004), i.e., a replica of a required remote data d_r is placed at a node having a larger NB toward the node requiring d_r .

Before the execution of each tenant query Q requiring distributed data over a set of nodes within a region, the RT of Q (RT_Q) is estimated in order to check whether the RT objective (SLO_{RT}) is satisfied or not. The RSPC replica management deals with the following issues:

- A. A new replica is created only if: (a) SLO_{RT} is not satisfied. Then, (b) the provider must have an economic profit when a replica placement node that satisfies SLO_{RT} is found.
 - a. SLO_{RT} is not satisfied when RT_Q exceeds a given critical RT threshold (RT_{SLO_PQ}), i.e., replication per- query, or (more often) when RT_Q exceeds another (lower) RT threshold (RT_{SLO_PSQ}) for a given number of times, i.e., replication per set of queries. Hence, the creation of a new replica is avoided whenever RT_Q exceeds a non-critical threshold. The RT_Q estimation is based on a proposed cost model that takes into account the parallel execution of DB queries. It considers several factors, e.g., NB, query complexity, user’s access patterns and query arrival rate that impact query performance while concurrent queries are processed.

- b. In order to check whether the replication is profitable for the provider, its revenues and expenditures are also estimated before the execution of Q in a multi-tenant context. Within the proposed economic cost model, penalties resulting from an SLA breach are also factored as well as the replication cost. A penalty management algorithm is also proposed to reduce these penalties.
- B. A replica placement heuristic is also proposed. It aims to find, within a reduced search space, a suitable placement *NodeP* for receiving a new replica in order to satisfy SLO_{RT} again, i.e., $RT_Q < RT_{SLO_PSQ}$, in a profitable way. In order to determine what and where to replicate, the proposed heuristic starts by identifying the resource bottleneck that causes the SLO_{RT} unsatisfaction:
 - a. Q or $Q_p \subseteq Q$ could require a remote d_r through a low NB, which generates a data transfer bottleneck. Hence, *NodeP* should have a better NB to the node requiring d_r , or,
 - b. An overloaded node (requiring a local data d_l) could execute Q/Q_p that could not satisfy SLO_{RT} . Hence, Q/Q_p should be executed on a less loaded *NodeP* that also receives a replica of d_l .
- C. A dynamic adjustment of the replica factor, i.e., number of replicas, is also considered. Adding a new replica occurs in order to satisfy SLO_{RT} , which avoids penalties. On the other hand, unnecessary replicas are compressed and subsequently removed when SLO_{RT} is satisfied over time or according to the changes in the user’s access patterns. This reduces the consumption of resources, which increases the provider’s profits.

For evaluation purposes, we based on TPC-H queries and compared RSPC to four other strategies with regard to several metrics, including the average RT and the average replica factor for both uniform and skewed data distributions, the impact of the number of VMs, the query arrival rate and the query complexity on performance, the number of SLA violations and resource expenditures of the provider. RSPC not only provides acceptable RTs that satisfy SLO_{RT} but, takes into account the provider economic profit, especially under high workloads, strict RT thresholds and complex queries. It responds better to changes in the user’s access patterns. Replication overhead costs are also reduced since most of replications are performed per set of queries. Moreover, RSPC generates fewer expenses, e.g., penalties and data transfer costs.

Overall, the contributions of this paper are: (i) we propose a replica creation per query or (more often) per set of queries in order to simultaneously satisfy SLO_{AV} and SLO_{RT} in a profitable way, (ii) the NB hierarchy based heuristic aims to quickly find an acceptable replica placement, (iii) the replica factor is dynamically adjusted, and (iv) the replication cost as well as penalties are factored into the proposed economic cost model. The rest of this paper is organized as follows: Section 2 gives some cloud characteristics and the considered architecture. Section 3 presents the RSPC replica management (replica creation, replica placement and replica factor adjustment). Section 4 deals with a query RT estimation. Section 5 describes the proposed economic cost model. Section 6 shows the experimental results. Section 7 analyzes the related work. Finally, Section 8 concludes the paper and gives some future work.

2. Background and architecture overview

2.1. General context

Several tenants may simultaneously place their queries in the cloud. Thus, adopting a virtualization technology, through the creation of VMs on a physical machine, reduces the amount of re-

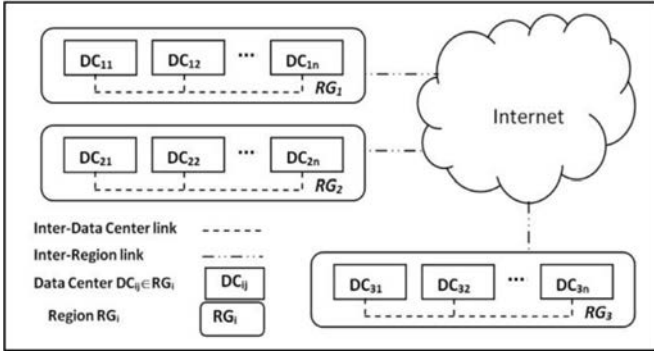


Fig. 1. An example of a distributed data center architecture.

sources required to execute each tenant query. In this context, scaling out, that we consider in this paper through data replication, results in less provisioning of resources than scaling up, i.e., adding resources within a physical machine (Hwang et al., 2016). Hence, an elastic resource management is critical to minimize operating cost while ensuring tenant SLOs, e.g., SLO_{RT} . These resources, e.g., replicas, are allocated so that a provider and its tenants agree on an SLA (Hameurlain and Mokadem, 2017), e.g., RT of tenant's queries should be less than a threshold defined in the SLA. Mainly, an SLA includes: (a) one or several SLOs, (b) a validity period, (c) a billing period (BP) during which the provider rents services to its tenants, (d) an agreed monetary amount paid by the tenant to the provider for the processing of its queries during a BP, and (e) an agreed monetary penalty amount paid by the provider to its tenant in case of breach of the SLA (Da Silva et al., 2012).

2.2. Architecture overview

Some companies consider the transferring of all data to a single DC/cluster when executing a tenant query. This generates a significant data transfer. Given that users as well as data are scattered across the globe, cloud systems should be deployed across multiple DCs covering large geographical areas.^{2,3} Some recent solutions, e.g., (Cooper et al., 2008; Ardekani and Terry, 2014; Tos et al., 2016), model a two level hierarchy, i.e., a region is composed of a single DC, while optimizing the cross-region data transfer consumption. However, links between DCs are heterogeneous (Gupta et al., 2014). Then, a NB hierarchical model is more realistic. In this paper, we consider within a region, several DCs that communicate through an intermediate NB. This leads to a system topology with three levels: regions, DCs and nodes that host data.

Let $RG = \{RG_1, \dots, RG_i, \dots, RG_q\}$ with $(1 \leq i \leq q)$ be a set of q geographical regions connected via the Internet without a direct link between them (Fig. 1). Thus, the NB capacity between geographical regions is not abundant and expensive. Each region contains heterogeneous DCs. We use $DC = \{DC_{i1}, \dots, DC_{ij}, \dots, DC_{in}\}$ with $(1 \leq j \leq n)$ to denote a set of n DCs within a region RG_i . The NB between these DCs is more abundant and cheaper compared to the first level. Finally, within each DC_{ij} , we use $N = \{N_{ij1}, \dots, N_{ijk}, \dots, N_{ijm}\}$ with $(1 \leq k \leq m)$ to denote a set of m heterogeneous nodes, i.e., VMs (Fig. 2). They are connected by a high NB that is even cheaper than the NB between DCs within a region.

As a real scenario, we consider a cloud system composed of distributed DCs over three countries (RG_1 , RG_2 and RG_3) as shown in Fig. 1. DCs within RG_i are located in different cities. Then, each DC_{ij}

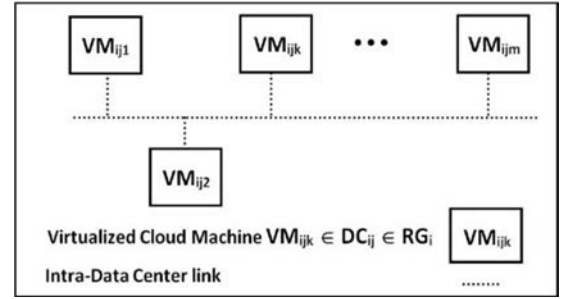


Fig. 2. An example of a data center DC_{ij} .

contains a number of VMs with a storage disk volume and computational resources for each VM_{ijk} , i.e., N_{ijk} . This corresponds to a shared-nothing architecture (Ozsu and Valduriez, 2011). When a tenant query is received in RG_i , the manager of RG_i coordinates its execution across nodes $\in RG_i$ and keeps track of important information such as the number of replicas and their locations. Then, a replica manager within each DC is responsible for creating/deleting replicas. It updates the RG_i manager whenever a replica is created/deleted.

3. The proposed replication strategy

The proposed RSPC strategy aims to satisfy tenant's SLO_{AV} and SLO_{RT} in a profitable way. Below, we present the RSPC replica management, including: replica creation (When to replicate?), replica placement (What and where to replicate?) and replica factor adjustment (How many replicas to create?) while ensuring an economic profit for the provider.

3.1. Replica creation

RSPC replica creation is considered only in the following scenarios:

- (i) Initially, when satisfying a minimum availability objective SLO_{AV} , and
- (ii) When the RT objective (SLO_{RT}) is not satisfied. This is checked by estimating the RT of a tenant query Q (RT_Q) before its execution. Then, a replication may be necessary to satisfy SLO_{RT} again. The replication decision relies on: (a) a cost model that estimates RT_Q and (b) an economic cost model that estimates the provider profit (Q_Profit) when executing Q while considering a replica creation.

3.1.1. Replication for satisfying SLO_{AV}

Authors in Wei et al. (2010) affirm that too many replicas may not increase availability. Thus, ensuring a given SLO_{AV} corresponds to maintaining a minimum number of replicas for each data set d , e.g., triplication is considered in HDFS (Cheng et al., 2012). As shown in Algorithm 1 (line 2), RSPC replicates initially each data set, e.g., a DB relation, ($Repl_Fact_Min$) times. $Repl_Fact_Min$ corresponds to the minimum number of replicas that satisfies SLO_{AV} . Thus, the number of replicas of d ($Repl_Fact_d$) should always be superior to $Repl_Fact_Min$.

In order to satisfy SLO_{AV} , each data set $d \in N_{ijk}$ within $DC_{ij} \in RG_i$ is replicated initially on: (a) a node $N_{ijk'} \in DC_{ij}$ ($k' \neq k$), with $N_{ijk'}$ the least busy node $\in DC_{ij}$, (b) a node $\in DC_{ij'} \in RG_i$ (with $j' \neq j$) with $DC_{ij'}$ a neighbour DC of DC_{ij} , and (c) a node across each region RG_i ($i' \neq i$).

3.1.2. Replication for satisfying SLO_{RT}

Suppose that a cloud provider receives thousands of (read-only) tenant queries. Suppose also that a given tenant query is executed

² Microsoft DCs. <http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx>.

³ Google DC Locations. <http://www.google.com/about/datacenters/inside/locations/>.

Algorithm 1
RSPC Replica Creation.

Input: RT_Q , RT_SLO_PSQ , RT_SLO_PQ , $NSetQ$, $Repl_Fact_Min$, $Repl_Fact_d$. Initially, $NQ=0$.
Output: Replica creation on $NodeP$.

1. **Begin**
- // Initial SLO_{AV} satisfaction
2. **While** $(\exists d / Repl_Fact_d < Repl_Fact_Min)$ **then** $\{d$ replica creation $\}$
- // SLO_{RT} satisfaction
3. **Before** each query Q execution
4. **If** $(RT_Q < RT_SLO_PSQ)$ **then** No Replica creation;
5. **Else if** $(RT_Q < RT_SLO_PQ)$ **then**
6. **if** $(NQ < NSetQ)$ **then** No replica creation; $NQ++$;
7. **else** \rightarrow **replication per set of queries**
- if** $(NodeP$ is found / $(RT_Q < RT_SLO_PSQ \& Q_Profit > 0)$
8. **then** $\{New$ replica creation on $NodeP$; $NQ=0\}$
9. **Else** // $RT_Q > RT_SLO_PQ \rightarrow$ **per-query replication**
10. **if** $(NodeP$ is found / $(RT_Q < RT_SLO_PSQ \& Q_Profit > 0)$
11. **then** $\{New$ replica creation on $NodeP$; $NQ=0\}$
12. **}**
13. Q execution
14. **End**

on an overloaded node or requires a remote access to distributed data sets. This may increase the RT of such a query.

The proposed RSPC strategy estimates whether SLO_{RT} is satisfied or not before the execution of each tenant query Q . For this aim, RT_Q is estimated. It corresponds to the estimated elapsed time from the initiation to the completion of Q (Ozsu and Valduriez, 2011). RT estimation is investigated in a further section. The estimated RT_Q value determines whether:

- (i) There is no replication, i.e., no replication is required. It means that SLO_{RT} is satisfied. It occurs if $RT_Q < RT_SLO_PSQ$ (line 4 in Algorithm 1). RT_SLO_PSQ is an agreed RT threshold defined in the SLA. Its value is previously negotiated between the provider and each tenant. It depends on the needs of applications or on the tenant's willingness to pay more for a strict RT threshold (requiring more resources to the provider) or a little less for a relaxed RT threshold (requiring less resources to the provider), or
- (ii) A replication may be triggered. In this case, two replication types are possible: per-query replication or replication per set of queries:

Per-query replication. If the estimated RT_Q is superior to RT_SLO_PQ , a data replication is immediately considered (line 9 in Algorithm 1). RT_SLO_PQ is a critical (maximum) query RT threshold defined in the SLA, with $RT_SLO_PQ > RT_SLO_PSQ$. It is previously established by the provider in order to limit the penalties paid to the tenant. This replication is called 'per-query' since a replication is considered when the estimated RT of a single query exceeds RT_SLO_PQ .

Replication per set of queries. Managing data replication in a per-query way may increase RTs, especially with a high query arrival rate. In order to reduce the replication overhead generated by repetitive replications, RSPC most often deals with a replication per set of queries. In this case, a data replication is considered if the estimated RT_Q belongs to $[RT_SLO_PSQ, RT_SLO_PQ]$ for a given number of times, i.e., a number of past queries were also in this interval. By this way, we avoid creating a new replica each time RT_Q exceeds a non-critical threshold (RT_SLO_PSQ). Hence, the replication per set of queries is triggered more often than the per-query replication.

In Algorithm 1, SLO_{RT} is still satisfied if no more than $NSetQ$ queries have an estimated $RT \in [RT_SLO_PSQ, RT_SLO_PQ]$ (line 6). A replication per set of queries is considered if $RT_Q \in [RT_SLO_PSQ, RT_SLO_PQ]$ for the $NSetQ^{th}$ time (line 7). NQ corresponds to the number of queries for which the RT was estimated in this interval.

3.1.3. Ensuring the provider profit

Even if a replication is considered (per-query or per set of queries), a replica is really created only if a node that could receive a new replica is found so that SLO_{RT} is satisfied again, i.e., $RT_Q < RT_SLO_PSQ$. Furthermore, this replication must be profitable for the provider (lines 7 and 10 in Algorithm 1). For this aim, the economic benefit of the provider (Q_Profit) is also estimated before deciding to replicate (before the execution of Q). This leads to the estimation of the provider's revenues ($Q_Revenues$) and expenses ($Q_Expenses$) as shown in Formula (1).

$$Q_Profit = Q + Revenues - Q_Expenses \quad (1)$$

Ensuring profitability for the provider when executing Q means that its revenues amount must be superior to its expenses amount when several tenants are served. Estimation of these costs is investigated in a further section.

To summarize, a replica creation occurs: (i) initially, when satisfying SLO_{AV} , or, (ii) before the execution of each query Q , a replica creation is considered if SLO_{RT} is not satisfied. It occurs if: $(RT_Q > RT_SLO_PQ)$, i.e., replication per-query, or, if $(RT_Q \in [RT_SLO_PSQ, RT_SLO_PQ])$ when the estimated RTs of $NSetQ-1$ past queries were also in this interval), i.e., replication per set of queries. However, a new replica is really created only if a replica placement $NodeP$ is found so that $(RT_Q < RT_SLO_PSQ)$ is satisfied again and $Q_Profit > 0$. Table 1 summarizes the parameter notations used in this paper.

3.2. Replica placement heuristic

Without an efficient replica placement, replicas may be distributed in an unbalanced way. In consequence, some nodes may contain more replicas than they can support, which generates an overload. Replica placement among nodes has been proven to be NP-hard (Kumar et al., 2014).

Let's assume $QEP: \langle Q, NEP \rangle$ a given Query Execution Plan that is provided for each tenant query Q with $Q: \{Q_0, Q_1, \dots, Q_p, \dots, Q_{n-1}\}$ a set of n operators, e.g., joins, and a Node Execution Plan $NEP: \{N_{ijk}, N_{ijk'}, \dots, N_{ij'k}\}$ with $(j \neq j' \& k \neq k')$ a set of nodes within a same region RG_i . NEP includes both nodes that execute Q and nodes that store all data sets, e.g., relations, required by Q and their existing replicas. Suppose that QEP does not satisfy SLO_{RT} . Thus, a replica placement $NodeP$ should be selected to receive a new replica in order to satisfy SLO_{RT} in a profitable way. For this aim, $NodeP$ should have low load, enough storage space and sufficient NB to serve the new replica.

Table 1
Summary of parameter notations.

Parameter abbreviation	Meaning
N_{ijk} or VM_{ijk}	Virtual machine $\in DC_{ij} \in RG_i$
DC_{ij}	A data center $DC_{ij} \in RG_i$
RG_i	A region RG_i
SLO_{AV}	Minimum availability objective
SLO_{RT}	Response time (RT) objective
RT_Q	The estimated RT of Q
RT_SLO_PQ	Critical RT threshold (per-query repl. is considered each time $RT_Q > RT_SLO_PQ$)
RT_SLO_PSQ	Lower (non-critical) RT threshold (repl. per set of queries is considered if $RT_Q \in [RT_SLO_PSQ, RT_SLO_PQ]$ for the $NSetQ^{th}$ time)
Q_p	An operator \subseteq a tenant query Q
$NodeP$	The selected placement node for a new replica
$NB_{ijk,NodeP}$	Available network bandwidth (NB) between N_{ijk} and $NodeP$
$Load_{ijk}$	Estimated load in N_{ijk}
$Repl_Fact_d$	Replica factor for data set d
$Repl_Fact_Min$	Minimum replica factor for satisfying SLO_{AV}
QEP	A given query execution plan for Q
NEP	Nodes $\in RG_i$ executing Q & storing data sets required by Q and their existing replicas
BP	Billing period
Q_Profit	Estimated monetary profit for the execution of Q
$Q_Revenues$	Estimated monetary revenues for the execution of Q
$Q_Expenses$	Estimated monetary expenses for the execution of Q
RT_{EQ}	The RT effectively measured when the provider executes Q
T_i	A tenant T_i
Ti_Amount	The provider's revenue received from T_i for executing a number $\#Q$ of queries
Pen_RT	A penalty (monetary amount) paid by the provider to T_i following an SLA breach

Identifying the best placement node requires visiting all nodes, which can lead to an overload. Instead, we propose a replica placement heuristic that provides an appropriate, i.e., an acceptable but not the best, replica placement $NodeP$. It is selected within the region RG_i that contains NEP . We are not looking outside RG_i as in [Tos et al. \(2017\)](#). As a result, the search space is significantly reduced. Let S_{dr} be the size of a remote $d_r \in NodeR$ required by $Q_p \subseteq Q$, S_{dl} the size of a local data set d_l required by Q_p executed on $NodeO$, $S(NodeP)$ the available storage capacity on $NodeP$, $Load_{ijk}$ the load estimation on N_{ijk} provided by a workload manager within DC_{ij} , $NB_{NodeR,ijk}$ the NB from N_{ijk} to $NodeR$ and Q_Profit the estimated provider profit when a new replica creation is considered.

RSPC replica placement starts by identifying the bottleneck resource that generates the unsatisfaction of SLO_{RT} . Two possible scenarios can cause a bottleneck: (i) a bandwidth bottleneck is generated by a transfer of required remote data d_r ($\in NodeR$) through a low NB when processing Q_p . Then, satisfying SLO_{RT} consists in finding $NodeP$ with a better NB to the node that requires d_r when $NodeP$ receives a replica of d_r or, (ii) a node that executes Q_p is overloaded despite only local data d_l are required on this node. Then, Q_p should be executed on a less loaded node $NodeP$, which will also receive a new replica of d_l . Below, identifying $NodeP$ is detailed for each scenario.

3.2.1. Scenario 1: SLO_{RT} not satisfied because of a remote data transfer bottleneck

This situation occurs when Q_p needs d_r that is received from a remote $NodeR$ with a low NB (line 2 in [Heuristic 1](#)). We start by identifying $NodeT \in NEP$ in DC_{ij} , which has the least NB to repatriate d_r from $NodeR$ (line 3), i.e., $NB_{NodeT,NodeR} = \min_{ijk}(NB_{ijk,NodeR})$ with $N_{ijk} \in NEP$. Then, $NodeP$ should be identified in order to receive a new replica of d_r . The NB is checked between each $N_{ijk} \in DC_{ij}$ ($N_{ijk} \neq NodeR$) and $NodeT$. The node having the best NB to $NodeT$ is selected to be $NodeP$, i.e., $NB_{NodeT,NodeP} = \max_{ijk}(NB_{NodeT,ijk})$ (line 4). It should have enough storage to store a replica of d_r . Then, if (i) $RT_Q < RT_SLO_PSQ$ (line 5) and (ii) the provider estimates that it has a profit while considering this replication (line 12), d_r is replicated on $NodeP$ (line 13) and Q is executed on the updated NEP (now, NEP also contains $NodeP$). Otherwise, the process is repeated with other remote data required by other nodes $\in NEP$ (line 6).

3.2.2. Scenario 2: SLO_{RT} not satisfied because of an overloaded node

This situation occurs when SLO_{RT} is not satisfied because some nodes are overloaded (line 7 in [Heuristic 1](#)). We start by identifying the most loaded (busy) node $NodeO \in NEP$ that holds a local data set d_l when executing Q_p (line 8), i.e., $Load_{NodeO} = \max_{ijk}(Load_{ijk})$. Suppose that $NodeO \in DC_{ij}$. The next step consists in identifying the less loaded node $\in DC_{ij}$ (line 9), i.e., $Load_{NodeP} = \min_{ijk}(Load_{ijk})$, so that it could receive a replica of d_l , i.e., $S(NodeP) > S_{dl}$. It could also execute Q_p so that $RT_Q < RT_SLO_PSQ$ (line 10). If SLO_{RT} is not yet satisfied, the process is repeated with other nodes $\in NEP$ and so on (line 11). On the other hand, if $NodeP$ is found so that SLO_{RT} is satisfied and the provider has an economic benefit, a replica of d_l is created on $NodeP$ (now, $NodeP \in NEP$) Finally, Q is executed on NEP .

In both scenarios, if $NodeP$ is not found or replication is not profitable, Q is executed on the initial NEP and penalties are paid by the provider to its tenant.

3.3. Replica factor adjustment

A static over-provisioning of replicas constitutes a naive solution, which would mostly result in resource over-utilization and lower provider revenues ([Bonvin et al., 2011](#)). We have seen that RSPC deals with an incremental replication. As long as it is necessary and profitable, replicas are created by one in order to satisfy SLO_{RT} . However, creating a new replica consumes additional resources, e.g., NB and storage, which increases the expenses of the provider. Then, the provider should avoid unnecessary replications. As an example, replicas of unpopular data in [Cheng et al. \(2012\)](#) are erased in order to save storage resource consumption while ensuring fault tolerance. In this paper, we provide a dynamic replica factor adjustment in order to minimize resource consumption while satisfying SLO_{RT} .

Instead of a periodic replica factor adjustment as in [Tos et al. \(2017\)](#), the replica factor adjustment in RSPC is detected through the RT estimation as shown in [Heuristic 2](#). It occurs when SLO_{RT} is satisfied over time. It is considered when RT_Q is far below RT_SLO_PSQ , i.e., $RT_Q \leq \beta \times RT_SLO_PSQ$, with $\beta < 1$ (its value is established by the provider). In this case, it means that some nodes are under loaded or some replicas are less accessed and therefore unnecessary.

Heuristic 1

RSPC replica placement.

Input: Set of $N_{ijk} \in NEP \in RG_i$, $Q_p \subseteq Q$ requires remote $d_r (\in NodeR)$ and/or local $d_l (\in NodeO$ executing $Q_p)$. Initially, we suppose that **SLO_{RT} is not satisfied** & $NEP' = NEP$.

Output: $NodeP$ (replica placement node).

1. **Begin**
 2. **If** ($d_r \neq \emptyset$) **then** // Remote data transfer bottleneck
 3. {Find $NodeT (\in NEP)$ that requires $d_r / NB_{NodeT, NodeR} = \text{Min}_{ijk}(NB_{ijk, NodeR})$ }
 4. Find $NodeP \in DC_{ij} / NodeP \notin NEP \ \& \ NB_{NodeT, NodeP} = \text{Max}_{ijk}(NB_{NodeT, ijk})$ }
 5. **if** ($(NodeP \neq \emptyset) \ \& \ S(NodeP) > S_{dl} \ \& \ (RT_Q < RT_SLO_PSQ)$) **then**
{ $NEP \leftarrow NEP - NodeR + NodeP$; goto 12}
 6. **else** { $NEP \leftarrow NEP - NodeT$;
 - if** $NEP = \emptyset$ **then** { $NEP = NEP'$; goto 14} **else** goto 3}}
 7. **Else** //Overloaded NodeO
 8. {Find $NodeO (\in NEP)$ that requires $d_l / Load_{NodeO} = \text{Max}_{ijk}(Load_{ijk})$ }
 9. Find $NodeP \in DC_{ij} / NodeP \notin NEP \ \& \ Load_{NodeP} = \text{Min}_{ijk}(Load_{ijk})$ }
 10. **if** ($(NodeP \neq \emptyset) \ \& \ S(NodeP) > S_{dl} \ \& \ (RT_Q < RT_SLO_PSQ)$) **then**
{ $NEP \leftarrow NEP - NodeO + NodeP$; goto 12}
 11. **else** { $NEP \leftarrow NEP - NodeO$; **if** $NEP = \emptyset$ **then** { $NEP = NEP'$; goto 14} **else** goto 8}}
 12. **If** ($Q_Profit > 0$) **then** // Provider profit check
 13. d_r (d_l respect.) is replicated from $NodeR$ ($NodeO$ respect.) to $NodeP$
 14. Q execution on NEP
 15. **End**
-

Heuristic 2

RSPC unnecessary replica removal.

Input: NEP , $Q_p \subseteq Q$, β , RT_SLO_PSQ , RT_Q , d (data set required by Q_p), $Repl_Fact_Min$, $Repl_Fact_d$.

Output: N_Remov the selected node hosting the replica to be removed.

1. **Begin**
 2. **If** ($RT_Q \leq \beta \times RT_SLO_PSQ$) **then**
 3. {Find $d \in N_Remov \in NEP / d$ is the least popular data set
 4. **If** ($Repl_Fact_d > Repl_Fact_Min$) **then**
 5. {Compress a replica of $d \in NEP$ (can be deleted after T);
 6. $Repl_Fact_d = Repl_Fact_d - 1$ }
 7. **else** goto 3 & repeat with another data set required by Q_p }
 8. **End**
-

Data popularity constitutes an important parameter considered by many strategies, e.g., the most requested data are replicated in Ananthanarayanan et al. (2011). A popularity of a data set d is determined by analyzing the access to d from users during a unit of time (Ranganathan and Foster, 2001). Concerning the selection of unnecessary replicas, RSPC selects the replica of the least popular data set d required by Q_p . Replicas of unpopular data waste the storage resource and generate considerable bandwidth costs (Liu et al., 2018).

Suppose that d is stored in $N_Remov \in NEP$ (line 3). Then, the least accessed replica (of d) is selected within NEP . In RSPC, we do not remove this replica since it might be accessed again in the near future. We propose to compress this replica as in Liu et al. (2018) to avoid recreating it later. After that, if it is not accessed during a given period of time T (set by the provider), it is permanently deleted (line 5). However, deleting a given replica is not systematically done since SLO_{AV} should also be satisfied, i.e., $Repl_Fact_d \geq Repl_Fact_Min$.

Also, some important data may have higher priority of obtaining relatively more replicas. A good solution is to assign a weight for each data and then create a large number of replicas for the weighted data. We defer this issue to future work.

4. Query response time estimation

Although several approaches are possible for storing and processing data, e.g., in NoSQL systems, the success of commercial relational clouds, e.g., (Amazon Relational Database Service (RDS), 2019), proves that relational DBMSs are still useful in the cloud era.

Let's assume that a cloud receives DB tenant queries. Let's also assume that a DBMS query optimizer provides our strategy with a near-optimal QEP before the execution of each tenant query Q

within a region RG_i . It provides the location of all required relations and their replicas within RG_i .

RSPC checks whether the given QEP satisfies SLO_{RT} based on the estimation of RT_Q . For this aim, the costs of all resources required for the execution of Q are estimated as given in Formula (2) (Tomov et al., 2004). Thus, the costs of processing Q (CPU_Q_Cost), data access Input/Output (IO_Q_Cost) and data transfer between nodes ($Transf_Q_Cost$) are estimated before the execution of Q .

$$RT_Q = \text{MAX}[(CPU_Q_Cost + IO_Q_Cost), Transf_Q_Cost] \quad (2)$$

We assume that CPU and I/O resource consumptions do not occur simultaneously on a single node. We also assume that a query execution can start before the transfer of all data, e.g., as soon as a significant amount of data such as a page becomes available locally. We benefit from several studies in distributed DBMS, e.g., (Tomov et al., 2004), dealing with a query RT estimation. In what follows, we estimate the RT of a DB query when considering the above costs. Then, we highlight some important factors that impact this estimation.

4.1. RT estimation of a DB query

RT estimation of a single query with no dependent operations, i.e., no joins, is relatively simple. In contrast, estimating the RT of a query with multiple dependent operations such as joins is more difficult (Tomov et al., 2004). In what follows, we estimate the RT of a relational multi-join DB query. We consider the join operator in its simple hash version. A hash table is built in memory through the *Build* operator. Then, the join result is produced through the *Probe* operator (Ozsu and Valduriez, 2011) as shown in Fig. 3.

Definitions and assumptions. Let $QEP: \langle Q, NEP \rangle$ with $Q: \{Q_0, Q_1, \dots, Q_{n-1}\}$ a set of n join operators, i.e., Q is the join result of

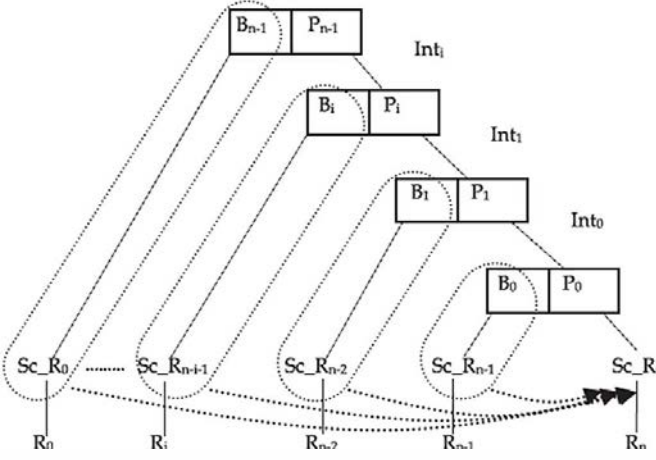


Fig. 3. Example of a dependency graph for a right-deep query tree.

$(n + 1)$ relations (R_0, R_1, \dots, R_n) . NEP is a set of nodes scattered over DCs within a region RG_i . It contains nodes that execute operators of Q and also all nodes that hold the required relations and their replicas. For ease of presentation, we assume that each node has a large enough memory to hold hash tables. We also assume that the effect of bucket overflow is believed not to affect the processor performance. Recall that a DBMS catalog stores statistical information about DB relations that we based on when estimating RT_Q . We benefit from some studies, e.g., (Tomov et al., 2004), to obtain an accurate estimate of the selectivity.

Let NT_i be the number of tuples of R_i , S_i the size of a R_i tuple (in bytes). I_{read}/I_{write} the #inst to read/write a data page from/to a disk, P_{Size} the number of tuples per page, I_{tuple} the cost of extracting a tuple from a page in memory, $I_{Build/Probe}$ the number of CPU instructions each tuple needs for relation building/probing, NT_{Bi} the build relation size (number of tuples) of the i^{th} join and NT_{Pi} the probing relation size of this join. Let also T_{CPU} be the duration of a CPU instruction.

Example of a DB query. We focus on a right-deep query tree as shown in Fig. 3, i.e., $Q = (R_0 \text{ join } (R_1 \text{ join } (R_2 \text{ join } (\dots \text{ join } (R_{n-1} \text{ join } R_n) \dots)))$). It provides the best potential for exploiting parallelism. Also, the size of a *Build* can be more accurately predicted since the cardinality estimates are based on predicates applied to base relations.

RT estimation. All scans of relations Sc_{R_i} followed by *build* operators B_i (enclosed by dotted lines in Fig. 3) are independent. Hence, hash tables can be built in parallel (independent parallelism) in order to produce the different B_i . As a result, only the longest path ($Sc_{R_{n-1}}$ followed by B_i) is taken into account when estimating the CPU cost of building hash tables. This justifies the using of *MAX* in the first part of Formula (3) (Hsaio et al., 1994). Then, we add the required cost for executing n probe operators that constitutes a pipeline chain. Each probe operator P_i consumes the output of the corresponding B_i . It begins after the end of P_{i-1} (NT_{Pi} is the size of the $(i-1)$ th join operation, except $NT_{P0} = NT_n$ in Fig. 3). The estimated CPU time when executing n joins is as follows:

$$\begin{aligned}
 CPU_Q_Cost = & \left[\text{MAX}_{i=0}^n \left(\left(\frac{NT_i}{P_{Size}} \times I_{read} \right) + (NT_i \times I_{tuple}) \right. \right. \\
 & \left. \left. + (NT_{Bi-1} \times I_{Build}) + \sum_{i=0}^{n-1} (NT_{Pi} \times I_{Probe}) \right) \right] \\
 & \times T_{CPU} \times (1 + \alpha + L) \quad (3)
 \end{aligned}$$

with $\alpha > 0$, a weighting factor, including hardware capabilities, e.g., the processing rate, the average I/O disk throughput and caching

capabilities on different N_{ijk} . Let L be the average load factor that takes into account the load in each N_{ijk} concerned by the execution of Q . We discuss the load on N_{ijk} in the next subsection. Dealing with left-deep and bushy query trees requires only some adjustments, e.g., in a left-deep query tree, NT_{Bn} is equal to the size of the last join operator ($NT_{P_{n-1}}$) except the first *Build* (equal to NT_n).

$$IO_Q_Cost = \sum_{i=0}^n \left(\frac{NT_i \times S_i}{P_{Size}} \right) \times t_{pio} \quad (4)$$

When executing Q , the required relations are read from the disk of the nodes hosting these relations. Whether remote or local data are concerned, I/O resources are consumed. The time required for reading depends on the size of the data read from the disk. Let us deal with the cost of reading $(n + 1)$ relations. Let t_{pio} be the disk service time per page. The estimated I/O cost of Q is given in Formula (4).

The execution of Q also requires data transfer between different nodes. Then, the NB available between nodes is taken into account. Let NB be the average NB (bytes/s) between the nodes $\in NEP$ involved in the execution of Q . Here, each intermediate relation Int_i (resulting from a Probe P_i) is always sent to the node that executes the next join (except the last P_{n-1} that produces the final result). The NB consumption also depends on the amount of data transferred. Let S_{Int} be the size (in bytes) of Int_i tuple. The estimated data transfer cost is shown in Formula (5).

$$Transf_Q_Cost = \left[\sum_{i=0}^{n-1} (NT_{Pi} \times S_i) + (NT_n \times S_n) \right] / NB \quad (5)$$

4.2. Important factors for the RT estimation

Different factors are taken into account when estimating the RT of Q . We have seen that data size and NB (between nodes within a region in RSPC) constitute important factors that significantly impact the RT estimation. In what follows, we also consider the workload on each node that executes $Q_p \subseteq Q$.

$$Load_{ijk} = \sum_{p=1}^{AR_{ijk}} (CPU_Q_p_Cost) \times (1 + C) \quad (6)$$

Let $Load_{ijk}$ be the estimated workload on a given N_{ijk} . It takes into account the arrival rate AR on N_{ijk} (AR_{ijk}) that corresponds to the number of concurrent Q_p executed on N_{ijk} by unit of time. Thus, $Load_{ijk}$ depends on the sum of computing costs of all Q_p executed or awaiting execution on N_{ijk} as shown in Formula (6) with $C > 0$, the query complexity factor in N_{ijk} . It corresponds to the number of executed joins on N_{ijk} , i.e., the number of hash tables on N_{ijk} . The average of all $Load_{ijk}$ (in all nodes concerned by the execution of Q) corresponds to the average load factor L introduced in Formula (3). $CPU_Q_p_Cost$ corresponds to the computing cost estimation of Q_p . Its estimation (in s) depends on the number of instructions $\#Inst$ required for processing Q_p as shown in Formula (7).

$$CPU_Q_p_Cost = \#Inst \times T_{CPU} \times (1 + a) \quad (7)$$

5. Management of provider economic costs: economic cost model

A cloud provider aims to generate profits when executing the tenant's queries while meeting tenant SLO requirements. For this aim, a replication can be triggered in order to satisfy SLO_{RT} and thus avoid the payment of penalties to its tenants. In RSPC, a new replica is created only if the estimated monetary incomes received

by the provider ($Q_Revenues$) are superior to the estimated monetary expenses ($Q_Expenses$) required for executing Q . In what follows, we propose a provider economic cost model that estimates $Q_Revenues$ and $Q_Expenses$ when executing Q in a multi-tenant context.

5.1. Estimation of the provider's revenues and expenditures

5.1.1. Provider's revenues

In order to improve their profits, providers implement a resource sharing among multiple tenants by consolidating various tenant's applications on a single system. Thus, multiple tenants are run on a same physical server (Long et al., 2014), e.g., a tenant can share a DBMS with another in the context of databases (Sousa et al., 2018). In return, each tenant pays the rent of resources to the provider according to the 'pay as you go' model, i.e., a tenant only pays what it consumes (Armbrust et al., 2010).

$$Q_Revenues = \left(\sum_1^n (T_i_Amount) / \#Q \right) + Rent_XaaS \quad (8)$$

In the proposed economic cost model, the provider serves several tenants and could receive two different amounts as shown in Formula (8):

- (i) The service amount (T_i_Amount) received from each tenant T_i served by this provider for the resources allocated, which increases the economic provider profit. Some commercial providers, e.g., Google Cloud, charge a monthly rental of resources (for subscribers) when others charge for a shorter period, e.g., one hour with Amazon, or for a given number of queries. We consider pricing for a given number of queries $\#Q$, e.g., 0.6\$ per 1000 queries. Then, it is possible to calculate the expected income per-query ($Q_Revenues$). Recall that when executing a tenant query, the provider could create one or more replicas in order to satisfy SLO_{RT} . Therefore, a tenant is not billed for the number of replicas created when its query is executed.
- (ii) The probable average price of a service credit ($Rent_XaaS$) received when the provider rents a given $XaaS$ service to another provider (Serrano et al., 2016).

Increasing the number of tenants through resource sharing decreases the per-tenant performance but reduces the provider overall operating cost (Long et al., 2014). Thus, it is desirable to increase the number of tenants in condition that the available resources are sufficient to meet the tenants' objectives, e.g., SLO_{RT} . With the same pricing policy for all tenants, the optimal number of tenants to serve is equivalent to the largest number of tenants that the provider can serve while satisfying the SLA.

5.1.2. Provider's expenditures

The provider has to pay the operating cost of each server executing Q or storing/transferring the data required for that execution. Thus, the provider's expenditures correspond to the total price of all these resources. The estimate of these expenses is performed before the execution of Q . It is given in Formula (9) that deals with the following denotations: Let RT_{Q_p} be the estimated RT needed to execute $Q_p \subseteq Q$. Let $\#VM$ be the number of required nodes that execute operators of Q . Let CPU_Cost be the CPU cost by million instructions or by unit of time, e.g., one hour in Amazon's cloud, for using a node allocated to execute Q_p .

The cost of replication is included to the provider's expenses. We take into account the cost of estimating both the RT and the provider's profit ($Estim_Cost$). Also, NB costs are consumed during the data transfer and storage costs are consumed at the destination at each replica creation. As the number of replicas increases

as the provider's expenditures increase. This is why RSPC considers an elastic resource management through a dynamic replica factor adjustment as described above. It permits to reduce the provider's expenditures. Let S_{dl} be the size of each stored data set. Let S_{dr} be the size of each transferred data set including the new replicas created. $\#D$ corresponds to the number of required data sets including their replicas. The storage of each of them has a monetary cost $Stor_Cost_{ijk}$ according to the prices applied in each DC_{ij} . The network cost $Netw_Cost$ for the access/transfer to/of r' remote data is also considered. It corresponds to the average cost of data transfers when executing Q . Obviously, it includes the cost of data transfer when creating a new replica.

$$Q_Expenses = \sum_{p=1}^{\#VM} (RT_{Q_p} \times CPU_Cost) + \sum_{l=1}^{\#D} (S_{dl_Cost_{ijk}}) + \sum_{r=1}^{r'} (S_{dr} \times Netw_Cost) + Estim_Cost + Avg_Past_Pen_RT + XaaS_Cost + Inv_Cost \quad (9)$$

Penalties paid by the provider to its tenants are also factored into the economic cost model as shown in Formula (9). This corresponds to the probable amount paid from the provider to T_i when one/several SLO is/are not satisfied. Despite the provider takes necessary precautions, e.g., replication, in order to avoid the payment of a penalty, there may be some queries that do not satisfy SLO_{RT} . Indeed, we rely on the average penalty cost per query $Avg_Past_Pen_RT$ paid by the provider to its tenants in the previous BP. Also, the provider must take into account the price of the investments, e.g., material, software licenses and power/energy costs (Inv_Cost). Finally, when the provider leases a given $XaaS$ service from another provider, its expenses could include the price of the $XaaS$ rental ($XaaS_Cost$).

5.2. Penalty management

When the provider executes Q with an effective RT greater than RT_SLO_PSQ , i.e., without creating a replica that satisfies SLO_{RT} , a violation of the SLA is recognized. It is computed as a penalty amount paid by the provider to its tenant. Here, we only focus on the SLO_{RT} violation.

Let RTE_Q be the RT effectively measured when the provider executes Q . As shown in the RSPC penalty management (Algorithm 2), when $RTE_Q < RT_SLO_PSQ$, no penalty is paid (line 2). Otherwise, a provider pays a penalty Pen_RT to its tenant (line 4). Pen_RT is also defined in the SLA and corresponds to the penalty amount paid each time RTE_Q exceeds RT_SLO_PSQ . In order to minimize penalties, we proceed as follows: when $RTE_Q \in [RT_SLO_PSQ, RT_SLO_PQ]$, the provider accepts to pay Pen_RT (after trying to avoid them through data replication). However, each time ($RTE_Q > RT_SLO_PQ$) is verified, i.e., per-query replication is triggered, the provider also pays Pen_RT while incrementing the value of V_Nber that corresponds to the number of times that RTE_Q exceeds RT_SLO_PQ .

Frequent SLA violations are damaging to the image of the provider. Hence, SLA violations should be reduced as much as possible. We propose to limit the number of critical SLA violations, i.e., $V_Nber < Critical_V_Nber$, especially when the total volume of data ($Data_Vol$) significantly increases compared to the DB volume agreed in the SLA (Vol_SLO) as shown in line 6 of Algorithm 2. If these two thresholds, i.e., $Critical_V_Nber$ and Vol_SLO , are reached simultaneously, the provider can renegotiate the value of RT thresholds (on the rise) with its tenant as well as the service amount paid by the tenant.

Reducing the number of SLA violations reduces penalty costs. When analyzing formulas (1) and (9), it is clear that reducing

Algorithm 2
RSPC penalty management.

Input: RTE_Q , RT_SLO_PSQ , RT_SLO_PQ , Pen_RT , $Data_Vol$, Vol_SLO , $Critical_V_Nber$. Initially, $V_Nber=0$.
Output: Pen_RT paid from the provider to its tenant T_i .

1. Begin
2. if $RTE_Q < RT_SLO_PSQ$ then no penalty
3. else
4. {if $(RTE_Q < RT_SLO_PQ)$ then {Provider pays Pen_RT to T_i }
5. else {Provider pays Pen_RT to T_i ; V_Nber++ }
6. if $((V_Nber == Critical_V_Nber)$ and $(Data_Vol > Vol_SLO))$ then
7. {Negotiating new RT_SLO_PSQ /new RT_SLO_PQ ; $V_Nber= 0$ }
8. End

as much penalty cost as possible without increasing the cost of replication, as we do here, improves the economic profit for the provider.

6. Experimental analysis

In order to evaluate the cost-effectiveness of RSPC, we compare its performance alongside four replication strategies proposed for cloud systems: (i) the Cost-effective Dynamic Replication Management strategy (CDRM) (Wei et al., 2010), (ii) the PErformance and Profit oriented data Replication strategy (PEPR) (Tos et al., 2017), (iii) the Dynamic Cost-aware Re-replication and Re-balancing strategy (DCR2S) (Gill and Singh, 2016) and (iv) the Dynamic Popularity aware Replication Strategy (DPRS) (Mansouri et al., 2017). In CDRM, a replica is placed on the node with the lowest blocking probability in order to reduce data access skew, which improves the load balance. A blocking probability is calculated on each VM. However, CDRM does not consider the economic aspects and SLA satisfaction as a decision criteria. PEPR benefits from the NB hierarchy, like RSPC, to reduce the NB consumption and takes into account the provider profit when a new replica is created. However, it only deals with a per-query replication. DCR2S aims to create a replica for a data set if its popularity exceeds a threshold. Through the concept of knapsack, replicas are re-replicated from higher-cost DCs to lower-cost DCs in order to reduce the cost of replication. Load balancing of resources is neglected and the replication cost is fixed in advance (within a given budget). Finally, DPRS replicates only the top 20% of frequently accessed data on the best locations according to the number of users' interests, free storage space and site centrality. Although the download time is reduced through the using of parallel downloading, it does not profit from the NB hierarchy.

6.1. Experimental setup and benchmark description

CloudSim (Calheiros et al., 2010), a popular and an open source cloud computing simulation tool, is used to simulate DCs. We simulated a cloud with 3 regions as shown in Fig. 1. Within each region, we simulated 10 DCs. Then, 1000 heterogeneous VMs are implemented in each DC. We have extended CloudSim to support data replication, query placement and some important requirements: (i) each VM has storage, memory and computing capacity and (ii) hierarchical NB capabilities and latencies are simulated between VMs, DCs and regions. For resource characteristics, we based on Barroso et al. (2018) to realistically represent a typical Cloud environment. Economic concepts are also taken into account: (i) a monetary pricing is defined for each resource in accord with Google Cloud, AWS and Microsoft Azure⁴ prices, (ii) a tenant is

⁴ Amazon S3 Pricing. <https://aws.amazon.com/s3/pricing/>. Azure storage pricing. <https://azure.microsoft.com/en-us/pricing/details/storage/>. Azure data transfer pricing. <https://azure.microsoft.com/en-us/pricing/details/data-transfers/>. Google Cloud pricing <https://cloud.google.com/compute/pricing?hl=fr>. March 2019

Table 2
Configuration parameters.

Parameter	Value
Number of regions (m)	3
Number of DCs within a region (n)	10
Number of VMs within a DC (q)	1000
Average (Avg) size of a relation	700 Mb
Avg. available inter-region BN (delay respect.)	500 Mb/s (150 ms respect.)
Avg. available inter-DC BN (delay respect.)	1Gb/s (50 ms respect.)
Avg. available intra-DC BN (delay respect.)	8 Gb/s (10 ms respect.)
Average size of a relation	800 Mb
Average VM processing capability	1500 MIIPS
Average storage capacity/ VM	10 Gb
Billing Period (BP) duration	10 min
#queries/ BP	[3000, 48,000]
RT_SLO_PQ	180s
RT_SLO_PSQ	{50, 100, 150}s
Provider revenues/ 1000 queries	{1, 0.8, 0.6}\$
Average Stor_Cost	0.15\$/Tb
Intra-DC Netw_Cost	0.0005\$/Gb
Intra region Netw_Cost	0.002\$/Gb
Inter region Netw_Cost	0.07\$/ Gb
Average CPU_Cost	1\$/10 ⁹ MI
Penalty/ violation (P_RT)	0.0025\$
NSetQ	10
Repl_Fact_Min	4

charged for a given number of queries (here #Q=1000) and (iii) an SLO_{RT} violation is computed as a penalty amount.

We based on the TPC-H data generation program with skew.⁵ Two data distributions are considered: (i) uniform distribution that provides a naive baseline and (ii) non uniform, here zipf, distribution, Breslau et al. (1999) that is designed to react to data popularity and models unconstrained accesses from an independent population such as Internet users. The zipf factor (z) that controls the degree of skew is set to 1. The arrival rate of DB queries follows a Poisson distribution. Our experiments dealt with 3000, 12,000, 30,000 and 48,000 queries during a BP. The broker assigns cloudlets (associated to queries) to randomly selected VMs to access relations themselves distributed on randomly selected VMs. We considered a subset of TPC-H⁶ queries {Q4, Q10 and Q8} for analytical purposes. These queries have different level of complexity {1, 3 and 7 joins respectively} when a right-deep query plan is pre-determined for each query. We call them simple, medium and complex queries respectively. We simulated a parallel execution of queries launched simultaneously by several tenants. A read-only DB relation constitutes the granularity of replication. We dealt with a simulation since it allows us to directly control some parameters in order to understand their individual impact on performance, e.g., query arrival rate and system configuration variations. Table 2 describes the main parameters used in our experiments.

⁵ <https://www.microsoft.com/en-us/download/details.aspx?id=52430>. March 2019.

⁶ TPC-H benchmark specification, 2019, [online]: <http://www.tpc.org/tpch/>.

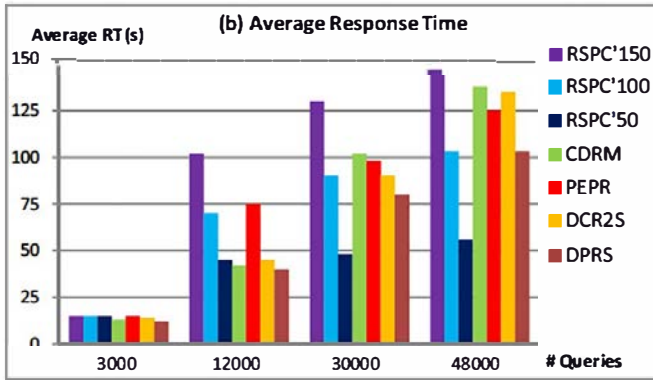
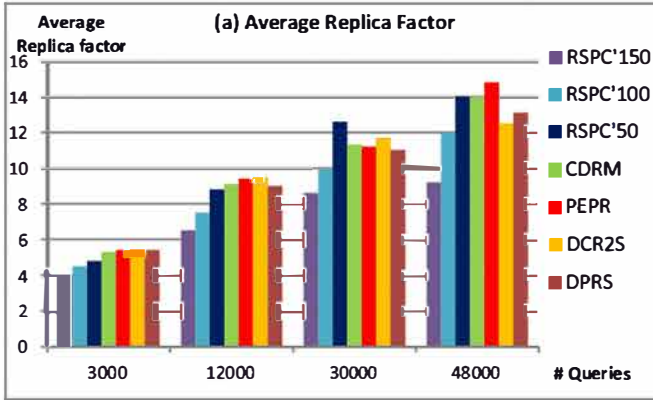


Fig. 4. (a) Replica factor and (b) RT with a uniform distribution.

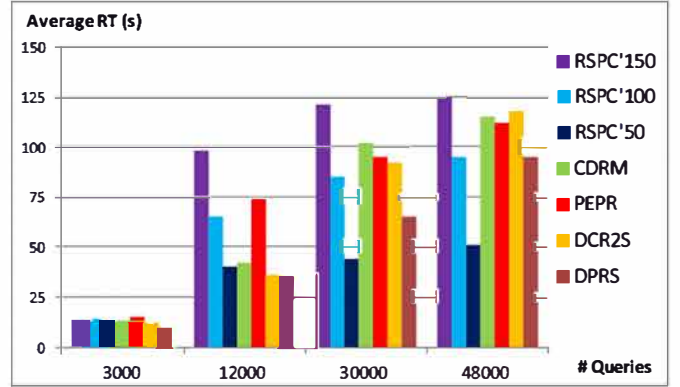


Fig. 5. Average RT for a zipf distribution.

most important number of replicas in order to satisfy SLO_{RT} while a less number of replicas are created with RSPC'150. CDRM considers only the access frequency and does not create more replicas when a load balance is achieved. DCR2S creates additional replicas in order to improve RT since the budget is not reached.

A larger number of queries (48,000 queries that correspond to 80 queries/s) increases the replica factor, as shown in Fig. 4(a). RSPC'50, RSPC'100 and RSPC'150 require only some additional replica creations. RSPC'150 does not exceed the RT threshold while RSPC'100 and RSPC'50 RTs slightly exceed the corresponding RT_{SLO_PSQ} . Overloaded VMs are blocked for receiving new queries in CDRM and popular data are updated with DPRS. This generates replica creations outside the region receiving the queries. Only a few replicas are created with DCR2S. Once the cost of replication exceeds the budget, the knapsack algorithm tries to optimize the cost of replication by re-replicating to lower cost DCs. However, load balancing is not taken into account and replicating outside the local region does not decrease the replication cost. Hence, DCR2S, DPRS and especially CDRM generate a significant RT increase. PEPR generates an important RT even it creates more RT replicas. This confirms that repeated per-query replications generate an important overhead while most of replications in RSPC are provided per set of queries. Recall that the RSPC's replica factor includes the replicas (here 4) initially created to satisfy SLO_{AV} .

We also measure the average RT for the compared strategies with a zipf distribution as shown in Fig. 5. The replica factor is proportional to the data popularity. RTs of DPRS, RSPC and DCR2S are less important than RTs obtained in Fig. 4(b) (around 11%, 9% and 7% respectively). DCR2S takes into account the data popularity variation by assigning different weights to data accesses. DPRS is dynamically adapted to the users' preferences while RSPC maintains replicas for the most popular data. No improvements are observed with CDRM that creates replicas based on load balancing. It is also the case with PEPR that periodically removes replicas regardless of the account data popularity.

6.2.2. Replica factor adjustment

Fig. 6 shows the average replica factor obtained with the compared strategies and when a significant decrease is observed in the number of queries during a BP. 24,000 queries are submitted in the first half of the BP (80 queries/s in average), followed by 1500 queries during the second half (5 queries/s in average). At the end of the BP, the replica factor of all strategies is decreased. All strategies aim is to reduce the consumption of resources as the workload decreases. However, RSPC removes more replicas than CDRM, DCR2S, and DPRS due to the RSPC's dynamic replica factor adjustment. This proves that CDRM, DCR2S and DPRS continue to use some of the previously created replicas even with reduced workloads. Here, unnecessary replicas will not receive any further

6.2. Experimental results

We have measured (during a BP) the following metrics: (i) the average replica factor and the average measured query RT with different distributions, (ii) the replica factor adjustment, (iii) the impact of the query arrival rate, the system configuration, and the query complexity on performance, (iv) the number of SLA violations and (v) the provider resource consumption and its expenditures.

RSPC experiments dealt with different values of RT_{SLO_PSQ} from a more strict one (50s) to a more relaxed one (150s) with an intermediate moderate value (100s). The provider revenues depend on these thresholds as shown in Table 2. We denote the RSPC strategy under these thresholds by RSPC'50, RSPC'150 and RSPC'100 respectively. The critical RT_{SLO_PQ} is set to 180s. These values are defined based on preliminary experiments.

6.2.1. Average response time and replica factor

Fig. 4(a) and (b) show the average RT and the average replica factor obtained with the compared replication strategies when the data distribution is uniform. We deal with queries that randomly include simple, medium and complex queries.

When a low number of queries are submitted during a BP, e.g., less than 12,000 queries, PEPR presents the most important replica factor since the replica decision is considered at the per-query level, i.e., each time RT of a query exceeds the RT threshold (100s here). The replica factor in CDRM, DCR2S and especially DPRS is more important compared to RSPC (in its three options). In fact, RSPC does not replicate data when SLO_{RT} is satisfied while CDRM aims to balance the workload between different nodes by creating more replicas. DPRS provides the best RT while RSPC'150 provides the most important RT. When the number of data access increases, the RT also increases. When 30,000 queries are submitted during a BP (50 queries/s), RSPC'50 presents the best RT. It creates the

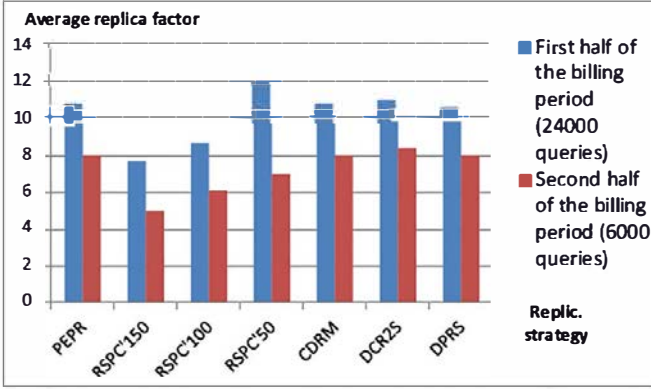


Fig. 6. Replica factor adjustment.

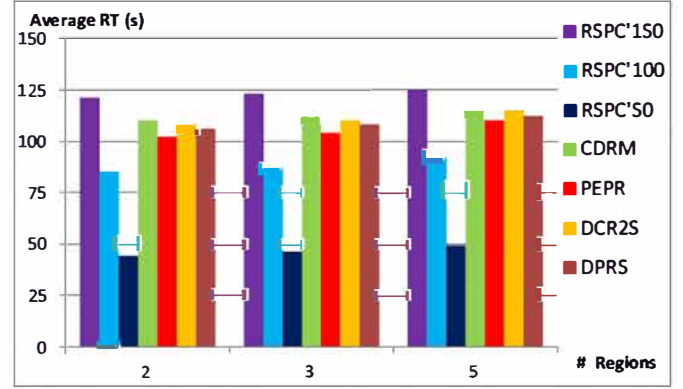


Fig. 8. Impact of the number of regions on performance.

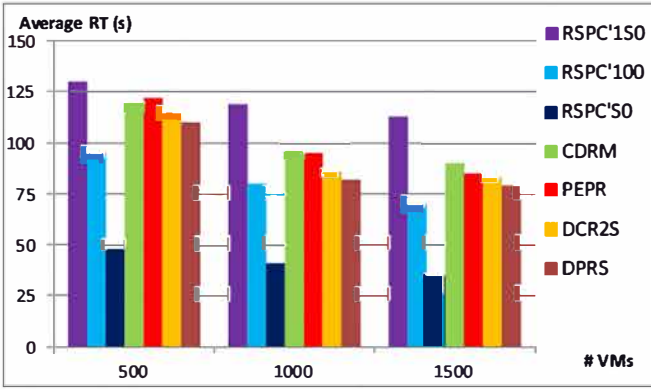


Fig. 7. Impact of the number of VMs per DC on performance. .

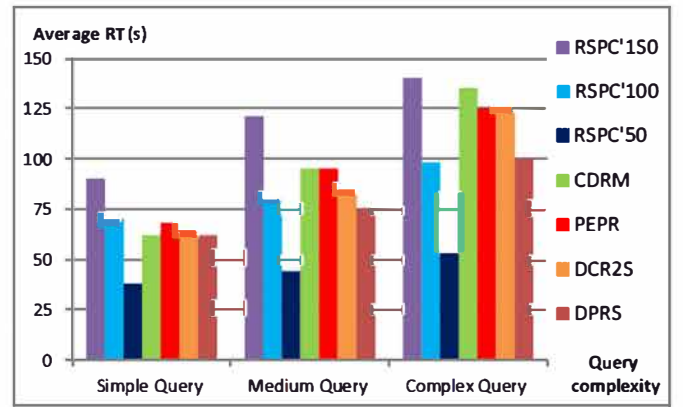


Fig. 9. Impact of query complexity on average RT.

queries during a certain period before deleting them. We defer the replica compression to future work.

6.2.3. Impact of the system configuration on RT

Fig. 7 shows the impact of the number of VMs per DC on performance when 30,000 queries are submitted during a BP. As the number of VMs per DC increases, the average RT is reduced with all strategies since VMs are less overloaded. With 1000 VMs per DC, the average RT decreases around 20% with CDRM and DCR2S, and around 25% with DPRS. This is due to the additional creation of replicas within a local region. With even more VMs per DC (1500 VMs), DCR2S and CDRM do not create many other replicas since the budget threshold is already reached with DCR2S and the load balance is achieved with CDRM. Also, the popular data are already replicated with DPRS. The average RT obtained with RSPC'50 (RSPC'150 respectively) decreases around only 15% (8% respectively). Only some additional replicas are created since the SLORT is satisfied for most queries.

With the increase on the number of regions (the total number of VMs within the system remains unchanged), the average RTs obtained with all strategies are slightly more important as shown in Fig. 8. With RSPC'50, fewer VMs are available for creating replicas and for executing the same number of queries while no replicas are created across regions. In contrast, more replicas are created across regions with CDRM, PEPR, DPRS and DCR2S.

6.2.4. Impact of query complexity on RT

We analyze the impact of query complexity on performance when 30,000 queries are submitted during a BP. We deal with simple, medium and complex queries as shown in Fig. 9. Experimenting with simple queries means that 80% of them are simple and so on.

With simple queries, CDRM, DCR2S and DPRS provide almost similar RTs while RSPC'100 and RSPC'150 generate longer RTs. However, RSPC (in its three options) provides RTs lower than the respective RT_{SLO_PSQ} . With medium complexity queries, the compared strategies provide longer RTs. RSPC'50 creates more replicas since the average RT is not far from RT_{SLO_PSQ} . With complex queries, some VMs are overloaded. We observe increasing RTs with all strategies. Nodes are clearly blocked for receiving new replicas in a local region with CDRM. Then, inter-region data transfers are required when creating new replicas with CDRM as in DPRS. Creating other replicas with DCR2S is not possible since the budget is reached while additional overheads are generated with the PEPR per-query replication. Then, RTs of DPRS, PEPR, DCR2S and CDRM increase significantly. In contrast, the average RT with RSPC'50 slightly exceeds RT_{SLO_PSQ} while the average RTs with RSPC'100 and especially RSPC'150 are below RT_{SLO_PSQ} .

6.2.5. Analysis of SLA violations

SLA violation analysis can be very useful when the RT of some queries far exceeds the average RT. Fig. 10 shows a relationship between the number of submitted queries and the number of SLA violations during a BP with a zipf distribution. We assume that a penalty amount is paid with PEPR, CDRM, DCR2S and DPRS when the RT exceeds 100s.

When 12,000 queries are submitted during a BP, PEPR generates the most important number of SLA violations as a result of higher RTs while other strategies generate almost the same number of SLA violations. In fact, most SLA violations are observed during the initial replica configuration, i.e., first queries during the BP. The aim is to avoid SLA violations through data replication. With 30,000 queries, RSPC'100 and RSPC'150 generate the least num-

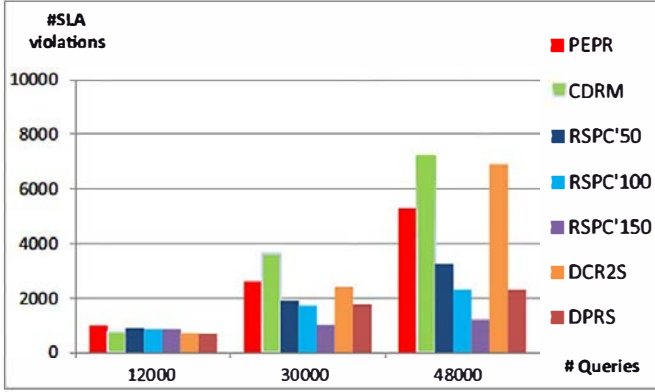


Fig. 10. Number of SLA violations.

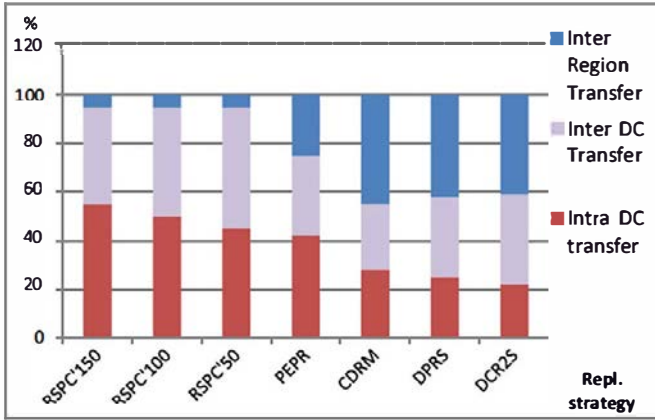


Fig. 11. Total Data transfer with respect to the NB hierarchy.

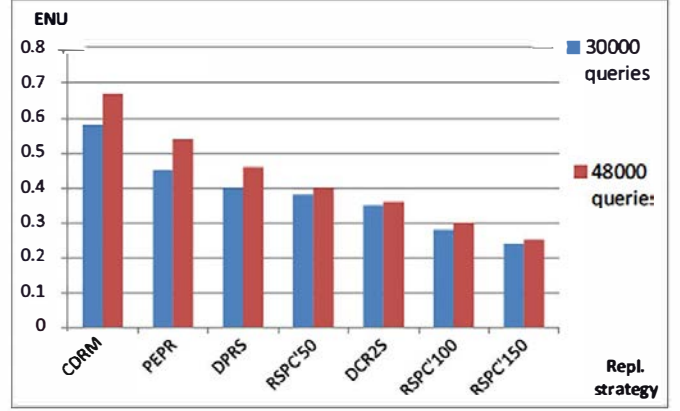


Fig. 12. Effective Network Usage (ENU).

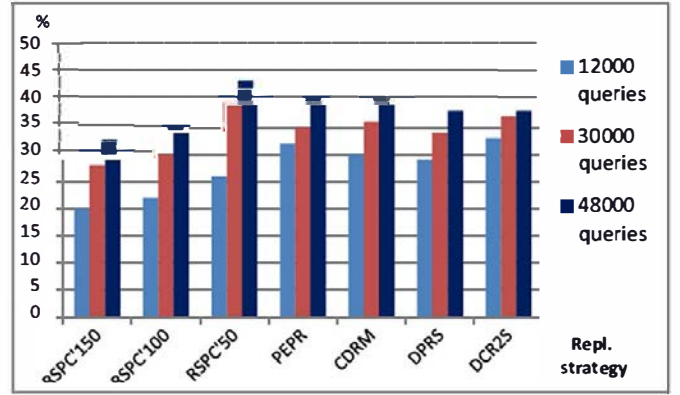


Fig. 13. Storage resource usage.

ber of SLA violations while more SLA violations are observed with PEPR and CDRM. The average RT with CDRM and PEPR is close to RT_{SLO_PSQ} but most of RT values exceed this value. With 48,000 queries, PEPR, CDRM and DCR2S generate even more SLA violations while the number of SLA violations with RSPC and DPRS increases slowly. The number of SLA violations with CDRM is 6 times (2.2 times respectively) more important than those generated by RSPC'150 (RSPC'50 respectively).

6.2.6. Provider's NB and storage consumptions

Fig. 11 shows the NB consumption required by the compared strategies while considering the NB hierarchy. In RSPC, the majority of data transfers are performed in the intra-region level. Inter-region data transfers are only performed during initial replications to satisfy SLO_{AV} . In contrast, inter-region transfers are more frequent with DCR2S, DPRS, PEPR and especially CDRM. This highly impacts RTs since inter-region links are slower.

$$ENU = (\#d_r + \#Repl) / \#d_l \quad (10)$$

We measure the Effective Network Usage (ENU) that shows the efficiency of the NB usage. It is calculated through Formula (10) (Cameron et al., 2003). $\#d_r$ indicates the number of times that a node reads a relation from a remote node. $\#Repl$ corresponds to the number of replications and $\#d_l$ is the number of times that a node reads a relation locally. The value of ENU is between 0 and 1. A lower ENU value indicates that the replication strategy is successful in sorting data in the proper locations while the NB is utilized more efficiently.

The obtained ENU values with the compared strategies are shown in Fig. 12 when 30,000 and 48,000 queries are submitted during a BP. Compared to DPRS, PEPR and especially CDRM,

RSPC'50 has a reduced ENU (around 5%, 8% and 20% respectively). The required data are available in local regions with RSPC. RSPC'100 and especially RSPC'150 do not create new replicas as long as SLO_{RT} is satisfied, which results in fewer data transfers. Compared to RSPC'50, the ENU of DCR2S is lower by 3% since replicas are not created when the budget is reached and only some replicas are re-replicated. More queries (48,000 queries are submitted during a BP) increases the ENU value obtained with CDRM, PEPR and DPRS (9, 8 and 5% respectively) since more replicas are created outside the local regions, while the number of replicas slowly increases with RSPC, e.g., the ENU value of RSPC'150 increases by only 2%.

We also measure the storage consumption required by the compared strategies. Fig. 13 shows the storage resource percentage used in the system. When only 12,000 queries are submitted during a BP, DCR2S, CDRM, PEPR and DPRS consume more storage resources since they create more replicas. Most of the time, it is not necessary to create additional replicas with RSPC since the RT is less than RT_{SLO_PSQ} , especially with RSPC'150. With 30,000 queries, RSPC'50 consumes more storage resources since more replicas are created in order to satisfy SLO_{RT} . With a more important number of queries, less than 5% more storage resources are required by RSPC'50 and DCR2S since only some replicas are created compared to other strategies.

6.2.7. Provider's monetary expenditures

Fig. 14 shows the total of the provider's monetary expenditures when executing the tenants' queries during a BP according to the resources prices listed in Table 2. The provider's expenses for the execution of a query are obtained according to Formula (9). They include storage, network, CPU, replication and penalty costs. On the

Table 3
A comparative analysis of some metrics considered by some existing replication strategies.

Repl. strategy	Year	Repl. type	Reduced RT	Availability	Load balancing	Fault tolerance	Consistency	Energy consumption	Parallelism	Repl. cost consideration	Monetary profit
SPC	-	Dynamic	+	+	+	-	-	-	+	NB, Storage	+
DRM (Wei et al., 2010)	2010	Dynamic	-	+	+	-	-	-	+	NB	-
ACS (Abu-Libdeh et al., 2010)	2010	Dynamic	-	+	+	+	-	-	+	Switch vendors	+
carce (Bonvin et al., 2011)	2011	Dynamic	-	+	-	-	-	-	-	Storage, NB	-
martSla (Xiong et al., 2011)	2011	Dynamic	+	+	+	-	-	-	+	NB, Storage	+
repSky (Bessani et al., 2011)	2011	Dynamic	+	+	+	+	-	-	+	Switch vendors	+
ipSim. (Zeng and Bharadwaj, 2014)	2013	Static	+	+	+	+	-	-	+	Storage	+
panstore (Wu et al., 2013)	2013	Dynamic	+	+	-	-	-	-	-	NB	+
ADR (Lin et al., 2013)	2013	Dynamic	+	+	+	-	-	-	-	Storage	-
WORD (Kumar et al., 2014)	2014	Dynamic	+	+	+	+	-	-	-	#machines	-
form (Long et al., 2014)	2014	Static	+	+	+	-	-	-	-	Energy	-
nerg (Boru et al., 2015)	2015	Dynamic	+	+	+	-	+	+	-	NB	-
EPR (Tos et al., 2017)	2016	Dynamic	+	+	-	-	-	-	+	NB, Storage	+
XR2S (Gill and Singh, 2016)	2016	Dynamic	+	+	-	-	-	-	+	NB, budget	-
S-QoS (Zeng et al., 2016)	2016	Dynamic	+	+	-	-	-	-	+	NB	+
imorm (Edwin et al., 2017)	2017	Dynamic	+	+	+	-	-	-	-	Storage	-
IPRS (Mansouri et al., 2017)	2017	Dynamic	+	+	+	-	+	+	+	Storage, NB	+
MCR (Liu et al., 2018)	2018	Dynamic	+	+	+	-	-	-	+	NB, Storage	-
MRS (Sun et al., 2018)	2018	Dynamic	+	+	+	+	-	-	-	#overload	+
DR (Mansouri and Iavdli, 2018)	2018	Dynamic	+	+	+	-	-	-	-	Storage, NB	-

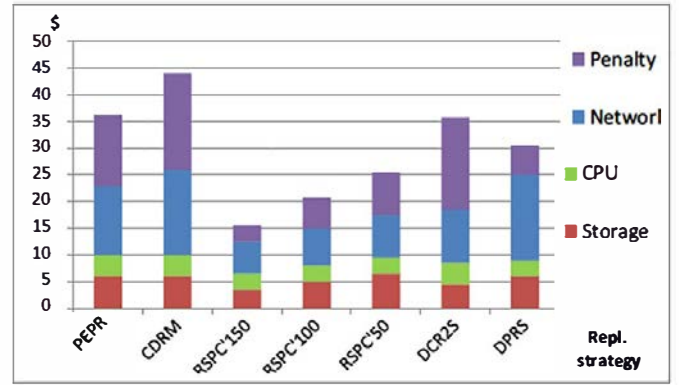


Fig. 14. Total provider's monetary expenditures.

other hand, tenants are billed for the 48,000 queries executed by the provider during a BP. With RSPC*50, RSPC*100 and RSPC*150, the provider revenue corresponds to 48\$, 35\$ and 29\$ respectively in accordance with the prices of executing a number of queries (# $Q=1000$) as indicated in Table 2. Obviously, the provider profit is obtained by subtracting the expenditures from the incomes as shown in Formula (1).

Almost the same processing cost is observed when experimenting with the compared strategies. As an example, the overhead generated by the RT estimation in RSPC is almost similar to that generated by calculating the blocking probability in CDRM. RSPC generates a significant data transfer save since the majority of replicas are created within a region. On the other hand, more storage consumption is required with RSPC*50 since it creates more replicas to satisfy SLO_{RT} . However, the storage is relatively cheap compared to the cost of data transfer. Also, much more penalties are paid with CDRM and DCR2S while the majority of expenses with DPRS and CDRM concerns the data transfer. In contrast, fewer penalties are paid with RSPC, especially with RSPC*150 since the average RT does not exceed RT_{SLO_PSQ} . RSPC*150 generates 3 times less monetary expenses than CDRM. With equal revenue, CDRM generates less profit for the provider than RSPC*100.

When comparing the profits generated by different options of RSPC, we deduce from Fig. 14 that RSPC*50, RSPC*100 and RSPC*150 generate a profit of 23\$, 15\$ and 12\$ respectively. Although RSPC*50 generates more monetary profits, RSPC*150 consumes fewer resources.

6.3. Discussion

With a reduced number of queries, CDRM, DCR2S and especially DPRS achieve a slightly better RT than RSPC (in its 3 options). It is due to the larger number of replicas created by these strategies, but without taking into account the provider profit. At the price of having a slightly higher RT but under SLO_{RT} satisfaction, RSPC generates less provider expenditure costs.

As the number of queries increases, VMs become busier. The same effect is observed with more complex queries. RSPC considers a replica creation only if SLO_{RT} is not satisfied. With CDRM, satisfying only a load balancing objective is not sufficient to ensure SLO_{RT} . DPRS creates additional replicas based on the data popularity change when these replicas are created outside the region that receives queries. This is not the case with DCR2S due to the limited initial budget and re-replication is not sufficient to balance the workload. On the other hand, per-query replication in PEPR generates an overhead that affects performance, while most replications are provided per set of queries in RSPC. With regard to the consideration of user behaviour changes, common in cloud environments, RSPC as well as DCR2S and DPRS generate best average RTs with

a zipf distribution. Regarding penalty costs, RSPC'50 creates more replicas to satisfy SLO_{RT} , which reduces penalty costs compared to CDRM, DCR2S and PEPR strategies. Furthermore, more unnecessary replicas are removed with RSPC through the dynamic replica factor adjustment, which reduces the provider expenditure costs. As a result, the provider's profits are increased.

With respect to resource consumption, RSPC benefits from the NB hierarchy, which reduces the data transfer consumption while PEPR, DPRS and mainly CDRM require inter-region data transfers. Although RSPC'50 requires more storage consumption, storage costs are cheaper than data transfer costs. Finally, whereas RSPC'150 generates higher RTs while satisfying SLO_{RT} , it consumes fewer resources while the profit of the provider is less important. Nevertheless, more resources are available. Thus, they can be used to generate additional profit for the provider when serving additional tenants.

7. Related work

Most of the replication work in the literature (Milani and Navimipour, 2016; Malik et al., 2015) classified data replication strategies in cloud systems into static (Zeng and Bharadwaj, 2014) and Dynamic (Bui et al., 2016) strategies while other work (Tabet et al., 2017) classified them into provider-centric (Da Silva et al., 2012) and consumer-centric (Limam et al., 2019; Zhao et al., 2015) strategies. RSPC is considered as a dynamic strategy since replicas of each data set are created, placed and maintained dynamically according to the user's access patterns. Also, we deal with the provider-centric approach that attempts to ensure the provider profit while satisfying tenant's SLOs. A classification according to the achieved objective function is also proposed in Mokadem and Hameurlain (2015). Once grouped together, these objective functions better address the issues of these strategies. Here, RSPC is based on both data and NB localities. Below, (i) we analyze the consideration of the *tenant performance/provider profit* trade-off by some existing strategies, (ii) we present some relevant techniques used by replication strategies, (iii) we present some replication strategies considered for multiple cloud providers, and (iv) we provide some examples of recent replication frameworks.

(i) Many replication strategies considered the replication costs when satisfying the performance SLO. The EIMORM strategy (Edwin et al., 2017) balanced among availability, load balancing and cost of replication when deciding to replicate. In Bonvin et al. (2011), the performance SLO is satisfied by a geographically-diverse placement of replicas in an economically efficient way. However, a high communication cost is observed. Furthermore, like most of the proposed strategies in the literature, the costs of replication and provider profit are not modelled as monetary costs. In Zeng et al. (2016), the number of replicas and their placement depend on the trade-off between performance and monetary cost in each node. However, the load balancing is not considered. In Casas et al. (2017), the replica factor is incremented as long as the monetary cost of an application does not exceed its upper limit. The work of Liu and Shen (2017) aims to minimize the payment cost of customers through a resource reservation pricing model while satisfying tenant's SLO, e.g., latency. The PEPR strategy (Tos et al., 2016) that was extended in Tos et al. (2017) aimed to satisfy SLO_{RT} while ensuring the provider profit. However, only a per-query replication is considered and the management of penalties is not considered. In RSPC, a new replica is created only if the provider has a monetary profit. Furthermore, the monetary costs of replication and penalties are factored into the provider's expenditures.

(ii) Some data replication strategies are based on techniques that achieve specific tenant SLOs. Examples of these techniques are: compression (Liu and Shen, 2016a) for data durability, or for

reducing replication bandwidth (Xu et al., 2015), multi-failure resilient scheme (Liu and Shen, 2016b) for enhancing availability, de-duplication (Nicolae, 2015) for reducing data transfer, prefetching (Mansouri and Javidi, 2018), data migration (Mansouri and Buyya, 2019), parallel downloading (Mansouri et al., 2017), data mining (Hamrouni and Charrada, 2015), supervised learning (Bui et al., 2016), overheating similarity of nodes (Sun et al., 2018), partitioning (Zhou and Fan, 2017) for ensuring performance and fragmentation for optimal security (Ali et al., 2018). On the other hand, many corporations, e.g., Facebook, as well as many replication strategies (Bui et al., 2016) are based on the erasure coding technique rather than/in addition to data replication. Data are encoded and expanded with redundant data stored across different locations to tolerate possible failures or outages (Abu-Libdeh et al., 2010). However, data encoding/decoding may generate an overhead. In this context, authors in HyRD (Mao et al., 2016) relied on data replication to store small files and on erasure code technique to store large files on multiple providers. A general rule is to use replication for data objects that are active, i.e., warmer, and to use erasure coding for data objects that are colder, i.e., inactive. Then, the more emphasis one places on read performance (storage efficiency respectively), the greater the advantage of replication (erasure coding respectively). RSPC relies on the compression technique. Replicas of unpopular data are compressed instead of being permanently deleted to avoid creating new replicas (again) for data that will become popular again later.

(iii) Although most of the replication strategies cited above have been proposed for a single cloud provider, other replication strategies are deployed across multiple cloud providers (Wu et al., 2013; Mansouri and Buyya, 2019; Abu-Libdeh et al., 2010; Bessani et al., 2011). SpanStore (Wu et al., 2013) spans DCs across multiple cloud providers. Pricing discrepancies are exploited in order to minimize costs when considering fault tolerance and latency requirements. Authors in Liu and Shen (2017) and Mansouri and Buyya (2019) focused on the costs of Get/Put operation of data sets. The resource price difference is exploited in Mansouri and Buyya (2019) to minimize the monetary cost of replication with the assumption that the workload on objects is known in advance. The work of Liu and Shen (2017) avoids, as in Abu-Libdeh et al. (2010), the vendor lock-in problem, i.e., a tenant may not be free to switch from one provider to another. The proposed strategy in Chen et al. (2014) is mainly designed for providing a fault-tolerance objective when the monetary cost of repair is reduced compared to the erasure code technique. By replicating each data set across regions in order to satisfy SLO_{AV} , RSPC also provides a high level of fault tolerance that is not the focus of this paper. Also, the pricing difference among providers is not exploited in RSPC as replication decision criteria. RSPC is considered for a single cloud provider that operates distributed DCs in a multi-tenancy context. However, the difference in resource prices within different DCs is taken into account when estimating the provider's expenses.

(iv) Some replication frameworks have been proposed for data analytical solutions (Pu et al., 2015; Wu et al., 2013; Ardekani and Terry, 2014; Kloudas et al., 2015). Tuba (Ardekani and Terry, 2014) provided a geo-replicated key-value store with an automatically re-configuration of its replica set. However, an exhaustive search is applied to enumerate all possible placements of replicas. Authors in Kloudas et al. (2015) considered an optimization formulation of data placement, which is not scalable in large-scale systems. Authors in Pu et al. (2015) used the query frequency and data access statistics when placing data in order to reduce the bandwidth cost between DCs. RSPC also uses statistics when estimating the size of intermediate results in order to estimate the RT of read-only queries. However, RSPC does not search the optimal configu-

ration. We consider a replica placement heuristic that deals with a reduced search space.

For a more advanced comparison between these strategies, we have identified, in Table 3, a non-exhaustive list of some data replication strategies with respect to some important metrics. A lot of strategies have been proposed for cloud systems. However, there is not a single one that ensures all the tenant objectives while considering the economic aspects of clouds. The main differences between RSPC and most of the mentioned strategies are summarized as follows: (i) SLO_{AV} and SLO_{RT} are simultaneously satisfied while ensuring a monetary economic profit for the provider, (ii) the replica decision is based on the estimation of both RT and provider's profit while taking into account some important parameters, e.g., query complexity and query arrival rate, (iii) a replica placement is heuristically found when the NB hierarchy reduces the NB consumption, (iv) the replica factor is dynamically adjusted and (v) both penalty and replication costs are factored to the proposed economic cost model.

8. Conclusion and future work

We propose RSPC, a dynamic data replication strategy that simultaneously satisfies tenant's SLO_{AV} and SLO_{RT} while ensuring a profit for the provider. Maintaining a minimum number of replicas, across regions, for each data set permits to satisfy SLO_{AV} . Dealing with the SLO_{RT} satisfaction, the provider offers its tenants an RT threshold as a performance guarantee instead of an optimal performance. Through a proposed cost model, the RT of each tenant query Q is estimated before its execution within a region. A replica creation is considered only if SLO_{RT} is not satisfied. Often, it is considered per set of queries. Then, a new replica is really created in a balanced way only if a suitable replica placement node is heuristically found so that SLO_{RT} is satisfied again while generating an economic profit for the provider. Both provider's revenues and expenses are estimated through a proposed economic cost model in a multi-tenant context. The replication costs as well as penalties are factored into this model. Moreover, the replica factor is dynamically adapted to changes in the workload and user's access patterns.

We compare performance of RSPC alongside four other strategies when executing OLAP TPC-H queries. RSPC aims to just satisfy SLO_{RT} while taking into account the provider's profit. Moreover, it best satisfies SLO_{RT} under important query arrival rates, strict RT thresholds and complex queries, and adapts better to changes in the user's access patterns. Hence, penalty costs are reduced. Data transfer costs are also reduced due to the NB hierarchy, which impacts the provider profit. Furthermore, replicating per set of queries significantly reduces the generated replication overhead.

As a future work, a balancing can be made between the number of tenants and performance in order to improve the provider profit. In this context, we plan to prove that serving an optimal number of tenants through the 'pay as you go' model while satisfying SLO_{RT} results in an optimal profit for the provider. This constitutes a rationale behind the design of the proposed strategy. The replica creation/deletion decisions could also take into account the log of past queries in order to predict which replicas should be replicated/deleted in advance. RSPC could also be evaluated when using data compression/de-duplication to further reduce resource consumption. Further, we project to extend our strategy to be operational in a multi-provider cloud environment and compare its performance alongside other strategies based on techniques such as the erasure code based redundancy technique. Finally, we plan to implement RSPC in a real cloud environment.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

- Abu-Libdeh, H., Princehouse, L., Weatherspoon, H., 2010. Racs: a case for cloud storage diversity. *ACM SoCC*.
- Ali, M., Kashif, B., Khan, U., Bhardwaj, V., Keqin, L., Albert, Z., 2018. DROPS: division and replication of data in cloud for optimal performance and security. In: *IEEE Transaction on Cloud Computing*, pp. 303–315.
- Amazon Relational Database Service (RDS), 2019. https://aws.amazon.com/rds/?nc1=h_ls
- Ananthanarayanan, G., Agarwal, S., Kandula, S., Greenberg, A., Stoica, I., Harlan, D., Harris, E., 2011. Scarlett: coping with skewed content popularity in mapreduce clusters. In: *Int. Conf. on Computer system (EuroSys)*, pp. 287–300.
- Ardekani, M.S., Terry, D.B., 2014. A self-configurable geo-replicated cloud storage system. In: *OSDI'14, Broomfield*, pp. 367–381.
- Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, L., Zaharia, M., 2010. A view of cloud computing. *Commun. ACM* 53 (4), 50–58.
- Barroso, L.A., Holze, U., Ranganathan, P., 2018. The Datacenter as a Computer: Designing Warehouse-Scale Machines. In: *The Datacenter as a Computer: Designing Warehouse-Scale Machines*, p. 189.
- Bessani, A., Correia, M., Quresma, B., André, F., Sousa, P., 2011. DepSky: dependable and secure storage in a cloud-of-clouds. In: *Int. Conf. on Computer systems, EuroSys'11*, pp. 31–46.
- Bonvin, N., Papaioannou, T.G., Aberer, K., 2011. Autonomic SLA-driven provisioning for cloud applications. In: *In 11th Int. Symp. on Cluster, Cloud and Grid Computing*, pp. 434–443.
- Boru, D., Kliazovich, D., Granelli, F., Bouvry, P., Zomaya, A.Y., 2015. Energy-Efficient data replication in cloud computing datacenters. *Cluster Comput.* 18 (1), 385–402.
- Breslau, L., Cao, P., Fan, L., Phillips, G., Shenker, S., 1999. Web caching and Zipf-like distributions: evidence and implications. In: *Proc. of IEEE INFOCOM'99, New York, USA*, pp. 126–134.
- Bui, D.M., Hussain, S., Huh, E.N., Lee, S., 2016. Adaptive replication management in HDFS based on supervised learning. *IEEE Trans. Knowl. Data Eng.* 28 (6), 1369–1382.
- Buyya, R., et al., 2009. Cloud computing and emerging it platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Gen. Comp. Syst. Elsevier* 25 (17).
- Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R., 2010. CloudSim: a toolkit for modeling and simulation of cloud computing env. & evaluation of resource provisioning algorithms. *Softw. Pract. Exp.* 41 (1), 23–50.
- Cameron, D.G., Carvajal-schiaffino, R., Paul Millar, A., Nicholson, C., Stockinger, K., Zini, F., 2003. UK grid simulation with OptrSim. *e-Science All-Hands Meeting*.
- Casas, I., Taheri, J., Ranjan, R., Wang, L., Zomaya, A.Y., 2017. A balanced scheduler with data reuse and replication for scientific workflows in cloud computing. *Fut. Gener. Comput. Syst.* 74, 168–178.
- Chen, H., Hu, Y., Lee, P., Tang, Y., 2014. NCCloud: a network coding-based storage system in a cloud-of-clouds. *IEEE Trans. Comput.* 63 (1), 31–44.
- Cheng, Z., Luan, Z., Meng, Y., Xu, Y., Qian, D., Roy, A., Zhang, N., Guan, G., 2012. ERMS: an elastic replication management system for HDFS. In: *IEEE Int. Conf. on Cluster Computing Workshops (CLUSTER)*, pp. 32–40.
- Cooper, B.F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobson, H., Puz, N., Weaver, D., Yerneni, R., 2008. Pnuts: yahoo!'s hosted data serving platform. In: *Int. conf. of Very Large DataBases (VLDB)*, pp. 1277–1288.
- Da Silva, T., Nascimento, M., Macedo, J., 2012. Towards non-intrusive elastic query processing in the cloud. In: *Int. workshop on Cloud data management, Hawaii, USA. ACM*, pp. 9–16.
- Edwin, E.B., Umamaheswari, P., Thanka, M.R., 2017. An efficient and improved multi-objective optimized replication management with dynamic and cost aware strategies in cloud computing data center. *Cluster Comput.* 1–10.
- Gill, N.K., Singh, S., 2016. A dynamic, cost-aware, optimized data replication strategy for heterogeneous Cloud data centers. *Future Gen. Comp. Syst.* 65, 10–32 Elsevier.
- Gupta, A., Yang, F., Govig, J., et al., 2014. Mesa: geo-replicated, near real-time, scalable data warehousing. *PVLDB* 7 (12), 1259–1270.
- Hameurlain, A., Mokadem, R., 2017. Elastic data management in cloud systems. *Guest editors. Special Issue on J. Comput. Syst. Sci. Eng.* 32 (4).
- Hamrouni, T., Charrada, F., 2015. A data mining correlated patterns-based periodic decentralized replication strategy for data grids. *J. Syst. Softw.* 110, 10–27.
- Hsaio, H.I., Chen, M.S., Yu, P.S., 1994. On parallel execution of multiple pipelined hash-joins. In: *Proc. of ACM-SIGMOD, Minneapolis, MN, May 24–27*, pp. 185–196.
- Hwang, K., Bai, X., Shi, Y., Li, M., Chen, W., Wu, Y., 2016. Cloud performance modeling with benchmark evaluation of elastic scaling strategies. *Trans. Par. Distr. Systems* 27 (1), 130–143.
- Kloudas, K., Mamede, M., Preguiça, N., Rodrigues, R., 2015. Pixida: optimizing data parallel jobs in wide-area data analytics. In: *Proc. of VLDB'15*, pp. 72–83.

- Kumar, K.A., Quamar, A., Deshpande, A., Khuller, S., 2014. SWORD: workload-Aware data placement and replica selection for cloud data management systems. *VLDB J.* 23 (6), 845–870 Special Issue paper.
- Lang, W., Shankar, S., Patel, J., Kalhan, A., 2014. Towards multi-tenant performance SLOs. *IEEE Trans. Knowl. Data Eng.* V. 26 (6), 702–713.
- Li, R., Hu, Y., Lee, P., 2017. Enabling efficient and reliable transition from replication to erasure coding for clustered file systems. *IEEE Trans. Parallel Distrib. Syst.* 28 (9), 2500–2513 (2017).
- Limam, S., Mokadem, R., Belalem, G., 2019. Data replication strategy with satisfaction of availability, performance and tenant budget requirements. *J. Cluster Comput.* <https://doi.org/10.1007/s10586-018-02899-6>.
- Liu, J.-W., Chen, C.-H., Chang, J.-M., 2013. Qos-aware data replication for data-intensive applications in cloud computing systems. *IEEE Trans Cloud Comput.* 1 (1), 101–115.
- Liu, G., Shen, H., 2017. Minimum-cost cloud storage service across multiple cloud providers. *IEEE/ACM Trans. Netw.* 25 (4), 2498–2513.
- Liu, J., Shen, H., 2016b. A low-cost multi-failure resilient replication scheme for high data availability in cloud storage. *IEEE 23rd Int. Conf. on High Performance Computing.*
- Liu, J., Shen, H., 2016a. A popularity-aware cost-effective replication scheme for high data durability in cloud storage. In: *IEEE Int. Conf. on BigData, Washington D.C.*, pp. 384–389.
- Liu, J., Shen, H., Narman, H.S., Lin, Z., Li, Z., 2018. Popularity-aware multi-failure resilient and cost-effective replication for high data durability in cloud storage. *Trans. Parallel Distrib. Syst.*
- Long, S.-Q., Zhao, Y.-L., Chen, W., 2014. MORM: a multi-objective strategy for cloud storage cluster. *J. Syst. Architect.* 60 (2), 234–244.
- Ma, K., Yang, B., 2017. Stream-based live data replication approach of in-memory cache. *Concurr. Comput.* 29 (11), 1–9.
- Malik, S.R., Khan, S.U., Ewen, S.J., et al., 2015. Performance analysis of data intensive Cloud systems based on data management and replication: a survey. *Distrib. Parallel Databases* 1–37.
- Mansouri, N., Javidi, M.M., 2018. A new prefetching-aware data replication to decrease access latency in cloud environment. *J. Syst. Softw.* 144, 197–215.
- Mansouri, N., Kuchaki Rafsanjani, M., Javidi, M.M., 2017. DPRS: a dynamic popularity aware replication strategy with parallel download scheme in cloud environments. *Simul. Model. Theory* 77, 177–196.
- Mansouri, Y., Buyya, R., 2019. Dynamic replication and migration of data objects with hot-spot and cold-spot statuses across storage data centers. *J. Parallel Distrib. Comput.* 126, 121–133.
- Mao, B., Wu, S., Jiang, H., 2016. Exploiting workload characteristics and service diversity to improve the availability of cloud storage systems. *IEEE Trans. Parallel Distrib. Syst.* 27 (7), 2010–2021.
- Milani, A., Navimipour, N.J., 2016. A comprehensive review of the data replication techniques in the cloud environments: major trends & future directions. *J. Netw. Comput. Appl.* 64, 229–238.
- Mokadem, R., Hameurlain, A., 2015. Data replication strategies with performance objective in data grid systems: a survey. *Int. J. Grid Utility Comput.* 6 (1), 30–46.
- Nicolae, B., 2015. Leveraging naturally distributed data redundancy to reduce collective I/O replication overhead. In: *IEEE Int. conf. on Parallel Distrib. Process. Symp.*, pp. 1023–1032.
- Ozsu, M.T., Valduriez, P., 2011. *Principles of Distributed Database Systems*, third ed. Springer, New York.
- Park, S.M., Kim, J.H., Ko, Y.B., Yoon, W.S., 2004. Dynamic data grid replication strategy based on Internet hierarchy. In: *Grid and Cooperative Computing*, pp. 838–846.
- Pu, Q., Ananthanarayanan, C., Bodik, P., Kanadula, S., Akella, A., Bahl, P., Stoica, I., 2015. Low latency geo-Distributed data analytics. In: *SIGCOMM*, pp. 421–434.
- Ranganathan, K., Foster, I., 2001. Identifying dynamic replication strategies for a high performance data grid. In: *Int. Workshop on Grid Computing*, pp. 75–86.
- Serrano, D., Bouchenak, S., Kouki, Y., et al., 2016. SLA guarantees for cloud services. *Future Gener. Comput. Syst.* 54, 233–246.
- Sousa, F.R.C., Moreira, L.O., Costa Filho, J.S., Machado, J.C., 2018. Predictive elastic replication for multi-tenant databases in the cloud. *Concurr. Comput.* 30 (16).
- Spaho, E., Barolli, A., Xhafa, F., Barolli, L., 2015. P2P data replication: techniques and applications. In: *Modeling and Proc. for Next Generation Big-Data Technologies*. Springer, pp. 145–166.
- Sun, S.Y., Yao, W., Yong Li, X., 2018. DARS: a dynamic adaptive replica strategy under high load Cloud-P2P. *Fut. Gener. Comput. Syst.* 78, 31–40.
- Tabet, K., Mokadem, R., Laouar, M.R., Eom, S., 2017. Data replication in Cloud systems: a survey. *Int. J. Inf. Syst. Serv. Sect.* 8 (3), 7–33.
- Tomov, N., Dempster, E., Williams, M.H., Burger, A., Taylor, H., King, P.J.B., Broughton, P., 2004. Analytical response time estimation in parallel relational database systems. *Parallel Comput.* 30 (2), 249–283 ISSN 01678191.
- Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., Bora, S., 2016. A performance and profit oriented data replication strategy for Cloud systems. In: *IEEE Conf. on Cloud and Big Data Computing, CBDCOM, France*, pp. 780–787.
- Tos, U., Mokadem, R., Hameurlain, A., Ayav, T., Bora, S., 2017. Ensuring performance and provider profit through data replication in cloud systems. *Cluster Comput.* 21 (3), 1479–1492.
- Tos, U., Mokadem, R., Hameurlain, A., Tolga, A., Bora, S., 2015. Dynamic replication strategies in data grid systems: a survey. *J. Super Comput.* V. 71 (11), 4116–4140.
- Wei, Q., Veeravalli, B., Gong, B., Zeng, L., Feng, D., 2010. CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster. In: *Proc. of the IEEE Int. Conf. on Cluster Computing (CLUSTER)*, pp. 188–196.
- Wu, Z., Butkiewicz, M., Perkins, D., Katz-Basset, E., Madhyastha, H.V., 2013. Spanstore: cost-effective geo-replicated storage spanning multiple cloud services. In: *SOSP*, pp. 292–308.
- Xiong, P., Chi, Y., Zhu, S., Moon, H.J., Pu, C., Hacigumus, H., 2011. Intelligent management of virtualized resources for database systems in cloud environment. In: *Proc. of Int. Conf. of Data Engineering (ICDE)*, pp. 87–98.
- Xu, L., Andrew, P., Sudipta, S., Li, J., Ganger, G.R., 2015. Reducing replication bandwidth for distributed document databases. *SoCC '15*.
- Zeng, L., Xu, S., Wang, Y., Kent, K., Bremner, D., Xu, C., 2016. Toward cost-effective replica placements in cloud storage systems with qos-awareness. *Software* 47 (6), 813–829.
- Zeng, Z., Bharadwaj, V., 2014. Optimal metadata replications and request balancing strategy on Cloud data centers. *J. Parallel Distrib. Comput.* 74 (10), 2934–2940.
- Zhao, L., Sakr, S., Liu, A., 2015. A framework for consumer-centric SLA management of cloud-hosted databases. *IEEE Trans. Serv. Comput.* 8 (4), 534–549.
- Zhou, J., Fan, J., 2017. JPR: exploring joint partitioning and replication for traffic minimization in online social networks. *IEEE 37th Int. Conf. on Distributed Computing Systems*.

Riad Mokadem is currently an Associate Professor in Computer Science at Paul Sabatier University, Toulouse, France, and a member of the IRIT laboratory. His current research interests include query optimization in large-scale distributed environments, data replication and database performance. He is involved in several International conferences and journals. In 2017, Riad was invited as guest co-Editor for a special issue on 'Elastic Data Management in Cloud Systems' in the International Journal on Computer Systems Science & Engineering (IJCSSE).

Abdelkader Hameurlain is a full professor in Computer Science at Paul Sabatier University (IRIT Lab.) Toulouse, France. His current research interests include query optimization in parallel and large scale distributed environments, and mobile databases. Prof. Hameurlain has been the general chair of the Int. Conf. on Database and Expert Systems Applications (DEXA'02, DEXA'11, DEXA'17 and DEXA'18). Also, he was invited to give talks at many int. conferences (e.g. DEXA'09, IIWAS'12, COMP'13, BDAS'16 and ADBIS'19) and has also been guest editor of several international journals. He is a co-editor in Chief of the Int. Journal "Transactions on Large-Scale Data- and Knowledge-Centered Systems" (LNCS, Springer).