A data replication strategy for document-oriented NoSQL systems

Khaoula Tabet*

Laboratory of Mathematics, Informatics, and Systems (LAMIS),
University of Laarbi Tebessi,
Tebessa 12002, Algeria
Email: tabetkhaoula@gmail.com
*Corresponding author

Riad Mokadem

Institut de Recherche en Informatique de Toulouse (IRIT),
Paul Sabatier University, 118,
Route de Narbonne 31062,
Toulouse, France
Email: riad.mokadem@irit.fr

Mohamed Ridda Laouar

Laboratory of Mathematics, Informatics, and Systems (LAMIS),
University of Laarbi Tebessi,
Tebessa 12002, Algeria
Email: ridda_laouar@yahoo.fr

Abstract: Cloud providers aim to maximise their profits while satisfying tenant requirements, e.g., performance. The relational database management systems face many obstacles in achieving this goal. Therefore, the use of NoSQL databases becomes necessary when dealing with heterogeneous workloads and voluminous data. In this context, we propose a new data replication strategy that balances the workload of nodes and dynamically adjusts the number of replicas while the provider profit is taken into account. Result analysis shows that the proposed strategy reduces the resource consumption, which improves the provider profit while satisfying the tenant performance requirement.

Keywords: cloud environment; NoSql databases; data replication; provider profit; performance.

Reference to this paper should be made as follows: Tabet, K., Mokadem, R. and Laouar, M.R. (2019) 'A data replication strategy for document-oriented NoSQL systems', *Int. J. Grid and Utility Computing,* Vol. 10, No. 1, pp.53–62.

Biographical notes: Khaoula Tabet is currently a PhD Student in Computer Science at the University of Tebessa, Tebessa, Algeria. Her main research interests are related to information systems, cloud computing, big data, urban management, and other related topics.

Riad Mokadem is currently an Associate Professor in Computer Science at Paul Sabatier University, Toulouse, France, and a Member of the IRIT laboratory. His main research interests are query optimisation in large-scale distributed environments, data replication and database performance. He has been a Member of Organisation Committees of several conferences, mainly DEXA'11 that was held in Toulouse. Recently, he was invited as a Guest Co-editor for a special issue on '*Elastic Data Management in Cloud Systems*' in *the International Journal on Computer Systems Science & Engineering (IJCSSE)*.

Mohamed Ridda Laouar is a Full Professor in Computer Science at the University of Tebessa, Algeria. He received his PhD in Industrial and Human Computer Science at the University of Valenciennes, France, in 2005. His research areas include information systems, decision support systems, e-library systems, cloud computing and other related topics. He has contributed to journals such as *Hi Tech Library and Human Systems Management*. He is the Editor of *IJIST* journal and different proceedings of conferences such as of *ICIST* and *ICSENT*.

1 Introduction

The elastic management of voluminous and heterogeneous data is one of the most important research areas in cloud computing (Hameurlain and Mokadem, 2017). Companies have turned to cloud environments to host their applications and databases. They expect cloud providers to keep a certain agreed upon of set Service Level Objectives (SLO), e.g., performance, defined in a Service Level Agreement (SLA) (Stantchev and Schröpfer, 2009), a contract between a provider and its tenants, while the ‘pay as you go’ model is considered (Armbrust et al., 2010).

Data replication is an important technique that ensures availability and fault tolerance while the system performances are improved. It consists of having many copies of the same data across multiple servers (Tabet et al., 2017). Data replication is frequently used in: (i) Database Management Systems (DBMS) (Perez et al., 2010), (ii) parallel and distributed systems (Loukopoulos et al., 2005; Benoit et al., 2008), (iii) mobile systems (Tu et al., 2006) and (vi) large scale systems including P2P (Xhafa et al., 2012) and data Grid systems (Mansouri et al., 2014). However, the existing proposed strategies are not adopted in cloud systems. They aim to obtain the best performance without taking into account the profit of cloud providers that aim to maximise their profits. This motivates the proposition of several data replication strategies in cloud systems, (Wei et al., 2010; Sakr and Liu, 2012; Xue et al., 2015; Tos et al., 2016). Relational database management systems face many obstacles in achieving those needs. Therefore, the using of NoSQL databases becomes necessary over the last few years, especially when dealing with voluminous and heterogeneous data. One of the most known NoSQL systems is MongoDB (Chodorow, 2013) which is considered as the leading document store.

To the best of our knowledge, most of existing works in the literature focus on obtaining high performances by using different data replication strategies that are used through useful data placement and load balance. Some works have focused their attention on the improvement of auto-sharding, load balance, and auto-scaling algorithms, e.g., (Liu et al., 2012; Gu et al., 2015; Mohamed, 2015). Other works analyse the differences of Mongo’s data model, query and replication model with relational database management systems, e.g., (Lima et al., 2016; Mansouri and Asadi, 2014). Many of these works are trying to take a closer look to performance by comparing capabilities of MongoDB with other NoSQL systems, e.g., performance comparison between MongoDB and Cassandra in (Haughian et al., 2016). Results show that MongoDB provides better performance. However, most of these works focus only on the improvement of system performance. For example, all replicas are read-only and are updated only from the master node in the master-slave replication model. This can cause problems when a master node fails down. To overcome this problem and guarantee the system performance, the replica set model was proposed. It overcomes the fail of a primary node by auto scheduling. Then, it offers an automatic system load balance which achieves performance (Goel and Buyya, 2007). However, replication costs, SLA violation and provider profit issues are neglected.

In this paper, we propose a new data replication strategy for MongoDB NoSQL databases. The proposed strategy aims to ensure both system performance for the tenant and a profitability for the cloud provider. A replication of a document is considered only if two conditions are satisfied simultaneously: (i) the estimated response time of a tenant query, before its execution, is greater than a response time threshold (RespT) agreed in SLA and then, (ii) if this replication occurs, the provider should have an economic profit.

Dealing with the MongoDB query response time estimation, we take into account the most important parameters that impact the query execution, e.g., data size, number of shards, I/O and Network Bandwidth (NB). When data replication is considered, only popular data, i.e., having a high access frequency during a period of time, that are situated in overload nodes are replicated. Thus, the workload of nodes is balanced, which affects the system performance. The proposed replication strategy is also based on a geographical (Goel and Buyya, 2007) and NB level (Park et al., 2003) localities in order to place new replicas closer to data consumers and reduce the communication costs respectively, when replicating data. Furthermore, the number of replicas is adjusted dynamically in order to reduce the resource consumption, i.e., the least popular data replicas are removed. Finally, in order to take into account the provider profit, we estimate the revenues and expenses of the provider when dealing with data replication. The provider should have a real profit when considering this replication.

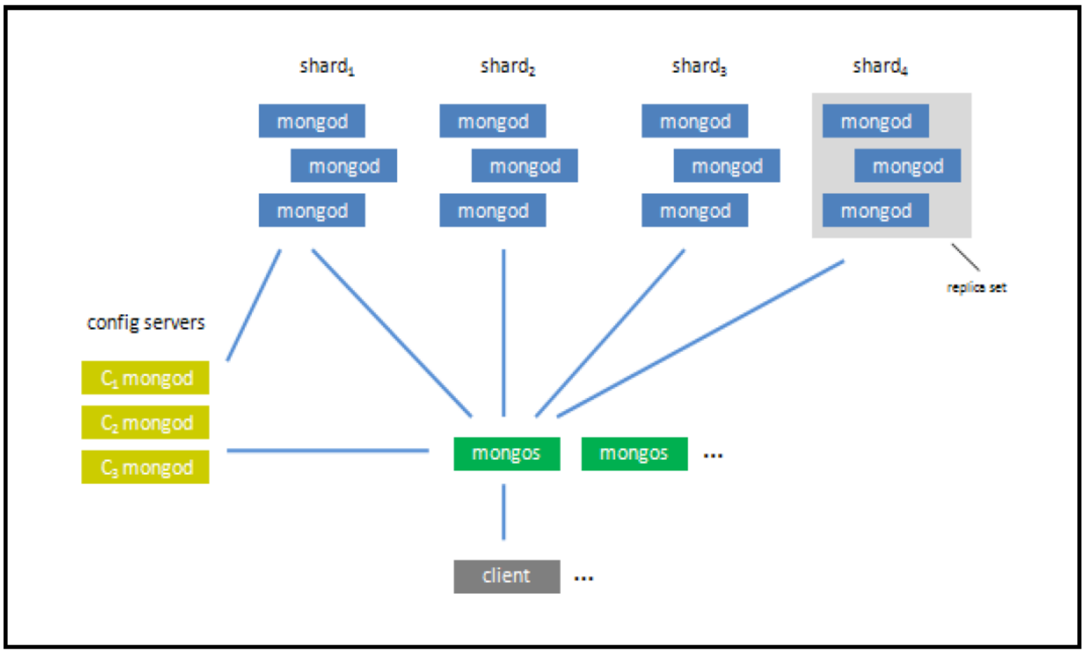
In order to validate the proposed strategy, we compared it to the already existing replication strategy in MongoDB. The results show that the proposed strategy provides better response times while the provider profit is taken into account. The organisation of the rest of the paper is as follows: Section 2 provides a summary of replication mechanisms in MongoDB. Section 3 details of the proposed replication strategy. Section 4 contains the performance evaluation of the proposed strategy. Finally, we conclude the paper and give some future work.

2 Background

2.1 MongoDB

MongoDB is an open source document-oriented NoSQL database system developed by 10gen in 2007 (Membrey et al., 2010). It manages collections of BSON documents, provides high performance, high availability, easy scalability through replication and auto-sharding. The main concept in MongoDB is the documents that present the unit of data for MongoDB. It is equivalent to a row in a RDBMS and a collection can be equivalent to a relation (Membrey et al., 2010).

Figure 1 Replication and sharding in MongoDB



There are different replication mechanisms in MongoDB. We distinguish two models: the Master/Slave and the Replica set replication models. The main difference between them is that a replica set has the ability to an automatic failover when primary node is unavailable by electing new one from existing secondary nodes situated in the same cluster. In a master-slave replication model, all replicas are read-only and updated only from the master node after changes, which can cause problems. When the master node fails down, there is no possibility to write new data. Thus, the most suitable model for replication in MongoDB is the replica set model.

In addition to the replication mechanism, MongoDB uses another mechanism called sharding. Sharding is the process of splitting data uniformly across clusters to parallelise the access of data. But, it cannot ensure fault tolerance. In case a shard fails down, the data stored on that shard become unavailable. At the same time, the system is still able to operate even if some parts of data are missing. Therefore, we combine sharding with replication in the proposed strategy because some systems require high availability, and high fault tolerance such in the new generation of cloud-based computing platforms. By replicating each shard, in case a replica fails down, another replica can replace it. Then, the system becomes more failure tolerant. Moreover, a replica set's different members have the same documents, while different shards have different documents (Rao and Govardhan, 2013) as shown in Figure 1.

2.2 Cloud architecture

Dealing with the considered cloud architecture, we consider a topology, i.e., composed of several geographical Data Centres (DCs). Let DC a set of data centres such as $DC = \{DC_1, DC_2,..., DC_n\}$, linked together through a high NB. Each data centre Dc_i contains a set N of m nodes realised as Virtual Machines (VMs) with $N = \{N_{i1}, N_{i2},... N_{im}\}$, as shown in Figure 3. Each node resides on physical hosts and has specific characteristics related to computing, storage and memory capacities, and network connectivity in order to achieve the execution of submitted queries. This leads to a system topology with two levels: DCs and nodes that host data. DCs and nodes are interconnected via hierarchy network bandwidths. DCs are connected via the internet with a low NB when the links between nodes across Dc_i have moderately higher NB compared to the first level.

3 Proposed data replication strategy

An existing data replication mechanism is already supported by MongoDB. It allows to having an automatic failover and loading balance while the main purpose is to obtain a high fault tolerance in case of a primary (master) node crash. In this paper, we propose a novel MongoDB data replication strategy in order to obtain better performance while maximising the provider profit.

The aim of any data replication strategy is to determine *what* data should be replicated, *when* a replica should be created/deleted, *how* many replicas to create and *where* to place a new replica (Mokadem and Hameurlain, 2015). In cloud environments, another issue is considered. A proposed replication strategy should also be profitable to the provider while tenant objectives are satisfied. We deal with all these issues in the next subsections.

3.1 When to replicate

The decision of the replication event depends on the verification of two conditions: (i) a response time is greater than a response time threshold and (b) a replication should be profitable to the provider.

Before the execution of each query Q, we estimate the response time of *Q* (*RespQ*) and compare it with the response time threshold *RespT* defined in *SLA*. In case *RespQ* is greater than *RespT*, some required datasets may be required to be replicated. As long as the provider is still estimated to be profitable, replication event is triggered as shown in the replica decision algorithm (Algorithm 1).

Algorithm 1 Replication decision algorithm

```
Algorithm: Replication decision
Begin
1:  RespQ ← estimated response time of a MongoDB query Q
2:  If (RespQ > RespT) then
3:  Select data concerned by replication
4:  NFound ← Node that holds the new replica / (RespQ < RespT)
5:  p ← Estimated profit by placement of new replica
6:  if (NFound≠∅) and (p > 0) then replica placement
End
```

3.1.1 Response time estimation

The provider is based on response time estimation in order to decide whether to replicate data or not to meet the tenant requirement.

The parallelisation of tasks is one of the most important characteristics when using cloud computing. We benefit from several existing relational database studies, e.g., (Özsu and Valduriez, 2011; Tos, 2017), to propose a response time estimation model that is suitable for our proposed strategy. Estimating a query response time is based on resource consumption when executing this query. Traditionally, we distinguish two types of parallelism: intra-operator and inter-operator parallelism in order to process large amount of data and to improve performance.

In this paper, we focus on response time estimation for a query that have no dependent operations. In fact, most of NoSQL systems do not use the join operator. Precisely, classical MongoDB search queries are without joins. In consequence, we focus on the intra-operator parallelism that consists in the parallel execution of several operators executed on several cloud sites. Then, the result consists in the combination of the generated results from each site. Recall also that we are not discussing here how the sharding is done in MongoDB (Özsu and Valduriez, 2011). There may be response variation between the execution of an operator on different shards because of individual hardware resources, memory, CPU etc.

Suppose that a tenant query *Q* is submitted to a cloud that contains several data centres, each one containing a number of nodes. Assume now that *Q* is delegated to a given node that should return a result to the tenant.

Q may be executed on several nodes and multiple shards. Let *Q* a tenant query that consists of several sub-queries $Q=\{Q_1, Q_2,...,Q_i,...,Q_k\}$ that are executed in parallel on *k* shards. Let Q_i a sub-query that requires a collection C_n. Suppose that the collection C_n is splitted into *l* chunks as $C_n\{C_{n,1}\ C_{n,2},..., C_{n,1}\}$, with replicas on *j* sites. In this case, we refer to this sub-query by $Q^j_{i,k}$.

The estimated response time of the submitted query is calculated as the sum of: (i) the longest estimated time it takes for *Q* $Q^j_{i,k}$, (ii) transfer time T_{Tr} for transferring data between shards. The intermediate results are transferred from all the concerned shards to a final destination and (iii) produce time T_{pr} for producing the results, e.g., an union operator is applied for grouping horizontally fragmented collections, and outputs, e.g., storing the results. The response time for the execution of the query *RespQ* is shown in the following equation (21):

$$RespQ = Max\,k\left[Resp\left(Q^j_{i,k}\right)\right] + T_{Tr} + T_{Pr} \tag{1}$$

In order to calculate the response time of the operator, i.e., executed on each shard in parallel $Resp\left(Q^j_{i,k}\right)$, it is necessary to evaluate the amount of time contributed by computing resources, including CPU, disk I/O and communication times as shown in the following equation:

$$\begin{aligned} Resp\left(Q^j_{i,k}\right) = Max\left[T^{CPU}\left(Q^j_{i,k}\right) + T^{I/O}\left(Q^j_{i,k}\right)\right] \\ + T^{Com}\left(Q^j_{i,k}\right) \end{aligned} \tag{2}$$

Estimated CPU time: The time related to CPU depends on how much time is spent when processing an operator, i.e., view as a number machine instructions executed on different shards for a unit amount of data. The estimation of a CPU time for the sub-query $Q^j_{i,k}$ is depicted in equation (3).

$$T^{CPU}\left(Q^j_{i,k}\right) = T^{CPU} * \#Inst * (1+a) \tag{3}$$

with T_{CPU}, is a time of a CPU instruction, and $a > 0$ a weighting factor, including CPU hardware capabilities, e.g., the processing rate and caching capabilities on a node that influences query engine performance.

Estimated input / output time: *Q* may require document d_l from a local node N_{ip} ($1 \le i \le m$, $1 \le p \times n$) and/or from a number *r'* of remote document d_r distributed on remote nodes (d_r may constitute intermediate results). Let Sd_l the size of total local documents (in bytes) required on N_{ip}, Sd_r the size of remote documents (∈ remote node) required for the *Q* processing (in bytes), IO_r and IO_{ip} the average I/O disk throughputs on a remote node and N_{ip}, respectively (in bytes/s) including randomly and sequentially access to a page. Hence, the estimated I/O cost of $Q^j_{i,k}$ (in sec) on N_{ip} that require local documents and *n* remote documents is given by:

$$T^{I/O}\left(Q^j_{i,k}\right) = \sum\nolimits_{r=1}^{n} Sd_r / IO_r + Sd_l / IO_{ip} \tag{4}$$

Estimated communication time: When a MongoDB query can require remote documents, the communication costs should be taken into consideration. This represents the time spent when transferring documents between nodes. We need to estimate the migration time of remote documents d_r into the local node. Let NB_{ip} the average NB to N_{ip} (in bytes/s). The estimation of a transfer time for $Q^j_{i,k}$ that requires *n* remote documents d_r is depicted in equation (5).

$$T_{Com}\left(Q^j_{i,k}\right) = \sum\nolimits_{r=1}^{n} sd_r / NB_{ip} \tag{5}$$

3.1.2 Provider profit estimation

Most of existing replication strategies in the cloud aim to achieve better performance without considering the provider profit (Tos et al., 2016). In our strategy, replication of a collection is considered only if a provider has an economic profit. In this context, we need to estimate both revenues and expenditures of the provider when executing Q that deals with data replication. From the provider point of view, the profit for executing Q (Profit_Q) is estimated as shown in equation (6).

$$Profitq = Revenues_Q - Expenses_Q \qquad (6)$$

All providers aim to maximise revenues paid by tenants. Usually, the purpose of any cloud provider is to obtain much more gain, i.e., profit. This justifies the parallel query execution. Also, expenditures of the provider must be minimised. But in case of an SLA breach, the provider pays a penalty to the tenant. Hence, an interesting challenge consists to avoid the penalty cost paid by the provider to the tenant for not satisfying a given SLO. Then, the profitability of cloud providers is based on the minimisation of expenses caused by the agreements violation.

Revenue estimation: A tenant is not billed for the number of replicas required when its query is executed, i.e., this process is transparent for the tenant. However, a tenant has to periodically pay the provider for the computing time, operational cost and resources allocated when processing its query.

As proposed in Tos (2017), a per query revenue estimation depends on the maximum query arrival rate Max_AR and the duration of the Billing Period (BP), already specified in the SLA. It also depends on the amount of rent (Rent) that tenant pays to the cloud provider for the services acquired during a BP.

Expenditures estimation: Evaluating a query Q generates operational costs that are impacted by different parameters. Let *Cost_t* be a cost by unit of time u for using a node allocated to the Q evaluation (Sousa and Machado, 2012), *T_Q* the estimated total time needed to evaluate Q, *Nodes_Nber* the number of the required nodes when evaluating Q during u, *Netw_cost* the cost of the NB usage including data migration from one node to another when creating a replica, *Stor_cost* the cost required to store replicas in disks. On the other hand, the provider pays a penalty to the tenant in case of an SLA breach. Hence, provider expenditures should include these penalties. In formula (7), the penalties paid from the provider to this tenant when one/several SLO_i is/are not satisfied is also considered (Tos et al., 2017). The estimation of the provider expenditures when evaluating Q (Expense_Q) is shown in formula (7).

$$\begin{aligned} Expense_Q = {} & \sum I^{Nodes\_Nber}\left(T\_Q * Cost\_t\right) \\ & + Netw\_cost + Stor\_cost + penalties \end{aligned} \qquad (7)$$

3.2 What data should be replicated?

In case we found that an estimated response time of Q (RespQ) is greater than RespT, we have to select data that should be concerned by the replication. In the proposed strategy, we based on the access frequency of each dataset during a period of time, from the arrival time to actual time. This consists to select the dataset that has the highest popularity degree. According to Mansouri and Asadi (2014), the access frequency of each file is analysed by considering the number of times the file was accessed during a given time interval. Here, the file with a greater value of Access Frequency is selected as the popular file. The Popularity Degree (PD) of a dataset d is to be calculated using formula (8) (Sun et al., 2012).

$$PD_d = \sum_{t_i = t_s}^{t_p}\left(an_k\left(t_{i,} t_{i+1}\right) * w\left(t_{i,} t_p\right)\right) \qquad (8)$$

when t_s, t_p, present respectively the start time and the present tim, $\text{an}_k(t_i, t_{i+1})$ the number of access to d during the interval $[t_i, t_{i+1}]$ and $w(t_i, t_p)$ a time-based forgetting function (w) with values within the interval [0, 1]. It is used to calculate the PD of a block of data at the present time t_p basing on access frequency at the start time t_s. When analysing the data popularity, only the popular dataset is replicated. We only replicate data that popularity exceeds PDT, with the *PDT* the PD threshold agreed on SLA. Therefore, it is not necessary to create replicas for all datasets, especially for less accessed files.

Algorithm 2 Selection of a dataset concerned by replication

```
Algorithm: Data selection
Begin
1: $RespQ_{ip}$ ← Estimated response time of Q on $N_{ip}$
2: If ($RespQ_{ip}$ >RespT) then
3: if Load($N_{ip}$)> LoadT then
4:   { Select data set d situated in $N_{ip}$ where $PD_d$ >=$Max_{PD}$
5:     Find appropriate node to place replicas
6:     Trigger replication}
End
```

The second criterion that we based on is the node workload *Load.* We consider the replication of data situated on overload nodes. Besides some particular data are accessed much more frequently than others, the load on nodes that hosts data will be augmented which can affect the system performance. To overcome this problem, we select data situated on an overload node and then, we place them in another node that is not overloaded. A workload of a particular node presents the amount of work, i.e., required to do. The definition of a node being overloaded can be given as follows: for each node N_{ip}, we can have its load denoted as $Load(N_{ip})$, that depends on both computing and network capacities of N_{ip}. In case we found that $Load(N_{ip})$ is bigger than *LoadT*, we declare that the current node N_{ip} is overloaded. LoadT is the threshold load agreed in SLA as a result, we replicate sets of frequently used data in this node in order to decrease the workload of N_{ip} as shown in Algorithm 2.

3.3 Where to place new replicas

When the estimated response time for a query, submitted on $N_{ij} \in DC_i$, is greater than *RespT*, we select data to be replicated such as *RespQ* < *RespT*. For this aim, we need to find the appropriate node to hold the new replica as shown in Algorithm 3. We profit from the NB level locality to place replicas in nodes that have a good NB with N_{ij}. In consequence, the search of the replica placement node *NFound* should be done in the current DC. The selected node should also have enough storage space and not overloaded. This constitutes the first condition. The second condition consists in the fact that the provider should also have a profit when replicating data on this node. The profit of the provider is checked before executing a query that uses the concerned replica. Finally, if there is a real profit, the new replica is really placed on *NFound*.

Algorithm 3 Replica placement algorithm

```
Algorithm: Replica placement
Begin
Let DC= {$DC_1$, $DC_2$, .....$DC_n$}, Let N={$N_{i,1}$ $N_{i,2}$ ...... $N_{i,m}$} $\in$ $DC_i$,
Let $RespQ_{ip}$ the estimated response time on $N_{i,p}$
Suppose that $RespQ_{ip}$ > RespT& the data set to be replicated is selected
Begin
1: found $\leftarrow$ Flase
2: for (k=1, k<m, k++)      //k!=p
3:   begin
4:      if ($RespQ_{ik}$ < RespT) Then NFound= $N_{ik}$
5:      found $\leftarrow$ true; Goto14
6:   end
7:   if (found=False) then
8:      begin
9:        Search of $DC_k$ with the best bandwidth to DCi
          //suppose that $N_{kk'}$ nodes of $DC_k$
10:       if ($RespQ_{kk'}$ < RespT) & ($N_{kk'}$.freespace> $S_d$) &
                       ($LoadN_{kk'}$ <LoadT) then
              {found $\leftarrow$ True; NFound= $N_{kk'}$ Goto14}
11:     Else Search of $DC_k$ with the best storage_price
12:         if ($RespQ_{kk'}$ < RespT) & ($N_{kk'}$.freespace > $S_d$ ) &
                       ($LoadN_{kk'}$ <LoadT) then
             {found $\leftarrow$ True; NFound= $N_{kk'}$ Goto14}
13.   end
14:  if found= False then Goto 17
15: If (NFound $\neq \emptyset$) then Estimate $Profit_Q$
16:  If (Profit >0) then Place the new replica on NFound
End
```

If one of these two conditions is not satisfied for all nodes on the current DC, the search is done in other DCs so that we find the appropriate node. Firstly, we search the DC that has the best bandwidth to the DC that holds the dataset to be replicated. If the response time objective is not satisfied, the search is applied on the DC that offers the best storage prices. Of course, the searched node should have the storage possibility to hold a new replica and it should not be overloaded, i.e., the load should be less than *LoadT*.

3.4 Replica number adjustment

Cloud providers aim to satisfy the tenant requirements while maximising its benefits. For this aim, it should minimise expenditures when executing the tenant queries. We consider an elastic resource management that consists to remove all unrequired resources, i.e., replicas that are not required should be removed. This reduces the provider expenditures and then, increases the provider profit.

Let Q a submitted tenant query in DC_i. For each node N_{ij}, we save the history of data (replicas) access. When the estimated RT on N_{ip}, is significantly slower than *RespT*, i.e., $RespQ_{ip} << RespT$, we estimate the popularity degree of each chunk of data, e.g., a replica *Ri*. We select replicas that have PD lower than *PDT*. Then, we delete it to gain more storage space as shown in the replica deletion algorithm (Algorithm 4).

Algorithm 4 Replica deletion algorithm

```
Algorithm: Replica deletion
Begin
1: $RespQ_{ip}$ $\leftarrow$ Estimated RT of Q submitted in $DC_i$
2: if ($RespQ_{ip}$ << RespT) then
3:    For each replica in $DC_i$ do
4:       Analyze and record recent history of data access
5:      Calculate $PD_{Ri}$ of a replica Ri
6:      If ($PD_{Ri}$ <PDT) then Delete replica Ri
End
```

4 Performance evaluation

4.1. Simulation environment

In order to evaluate the impact of our proposed replication model on the performance of MongoDB, we used the Yahoo Cloud Serving Benchmark (YCSB), i.e., an open source development tool for evaluating the performance of different cloud data stores including Nosql systems, e.g., MongoDB (Abubakar et al., 2014).

Table 1 Machine specifications and settings

Setting	*Value*
OS	Ubuntu 16.04 LTS
Word length	64-bits
RAM	8GB
Hard disk	1TB
CPU speed	1.70GHz
YCSB version	0.5.0
Core	5

First, we evaluate and explore the performance of MongoDB when using the already existing replication mechanism. Then, we apply our proposed replication strategy in order to compare the obtained results. This evaluation is done in a single tenant environment, where its specifications and settings are indicated in Table 1.

4.2 Evaluation of existing MongoDB data replication strategy with different scenarios

For benchmarking purposes, we have to define a set of workloads that are used on MongoDB. In the context of YCSB, a workload contains a set of core workloads that define a basic benchmark for cloud systems. In our experiments, we used six different workloads, each one consists of 1,000 operations on 1KB records (10 fields, 100bytes each with default key) already loaded into a database. Table 2 shows the description of the tested workloads.

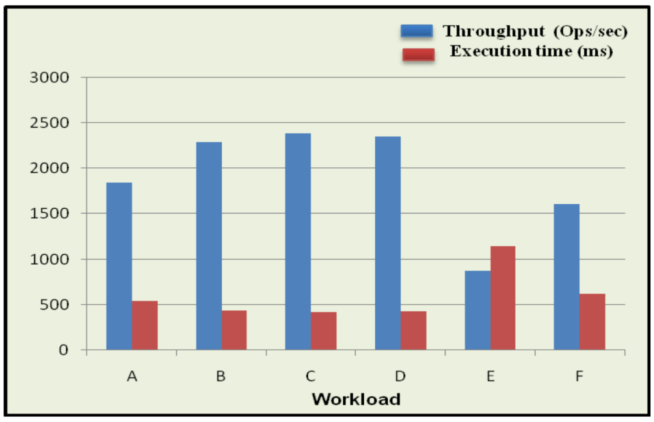
Table 2 Workloads description

Workload	*Operations*
A	50% Read, 50% Write
B	95% Read, 5% Write
C	100% Read
D	95% Read, 5% insert
E	95% Scan, 5% insert
F	50% Read, 50% RMW

Two metrics are measured: throughputs (in operations/s) and total execution time (in ms). Besides the throughput can be defined as a measurement that is used to determine the performance of a database system, it presents the amount of transactions produced over time during a test. On the other hand, the response time is the amount of time from the moment that a tenant sends a query until the application indicates that the query has completed with all results (http://searchnetworking.techtarget.com).

Figure 2 represents the obtained results when experimenting with each workload described in Table 2. We remark that the performance of MongoDB is more efficient when executing read only operations (workload C). This strengthens our choice of applying our strategy with read only operations, especially since the aim is to improve performances through data replication. Hence, we evaluate the proposed replication strategy when dealing with read only operations. This is why our strategy is proposed for OLAP applications.

Figure 2 Query execution results under different scenarios



4.3 *Result analysis of the proposed strategy*

Our main aim is to show that the proposed strategy improves the system performance while the provider profit is taken into account. We used *Mongo java* API to interact with MongoDB. Then, we added.jar files to *netbeans* IDE project (version 8.0.2) to have a *java Mongo* connection. After the implementation of the proposed replication strategy, first experiments consist to submit parallel read queries with several simulated workloads. We note the obtained execution time with the reading of 10, 100, 1,000, 10,000, 100,000 and 1,000,000 records in a collection. Recall that with the proposed strategy, the replication event is triggered only if the estimated response time is bigger than a response time threshold. We specified a response time threshold as 5,500ms for each query. Then, the replication should be profitable for the provider. Throughout the following experiments, we deal with two important metrics: (i) the response time and (ii) the number of replications. Recall that the response time is the amount of time from the moment that a tenant sends a query until the production of the first results (Özsu and Valduriez, 2011).

Figure 3 Impact of the proposed strategy on MongoDB performances

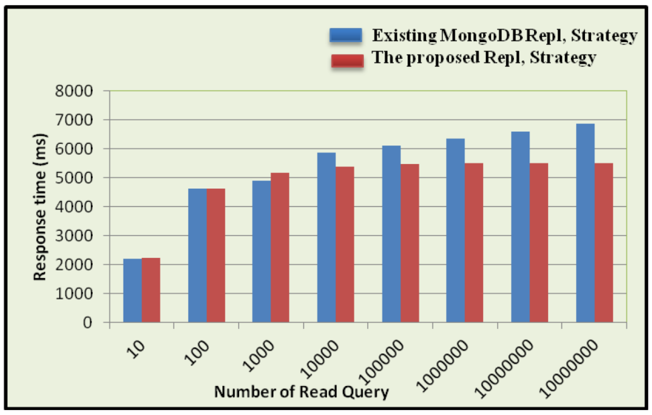
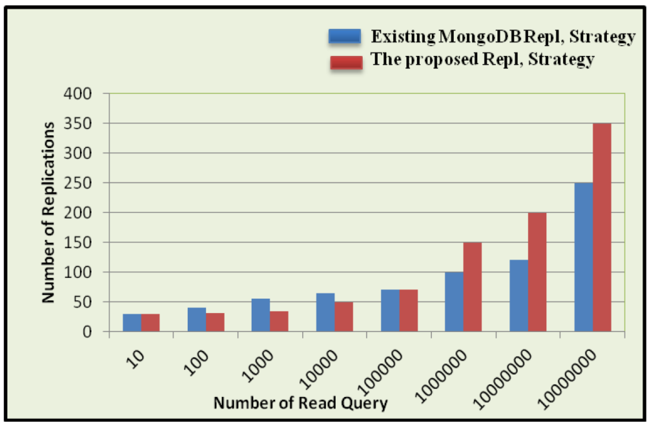


Figure 3 shows the variation of the response time with the two compared replication strategies while varying the workload. With a reduced number of queries, a slightly more important number of replicas are created by the existing replication strategy in MongoDB. But in contrast, the compared strategy doesn’t take into consideration the profits of the provider. Fewer replications are required with our strategy since the response time is inferior to the response time threshold. Then, there is a marked disparity when the workload increases. Gradually, when we have more loads, subsequently more replications occur with our strategy in order to avoid the SLA breach. This is due to the fact that a replication is triggered each time the estimated response time is greater than a response time threshold. This responds to the aim of cloud providers. In fact, the provider profit should be maximised through the minimisation of SLA violation while the SLO response time is satisfied.

Figure 4 shows the response time values obtained when experimenting with the two data replication strategies. With a small number of queries, we obtained a similar response time with both strategies. The response time of the proposed strategy is better when dealing with more than 100 parallel read queries. With a high load, it is clear that our replication strategy produces less important response times. Response time values of the proposed strategy should be inferior to the response time threshold. Otherwise, a data replication is triggered. This explains why we have a more important replications with the proposed strategy. Hence, we have better response time values with the proposed strategy. On the other hand, each time the compared replication strategy exceeds the response time threshold we have a SLA violation. In consequence, the provider should pay a penalty to the tenant when there is no penalty to pay with the proposed strategy. In consequence, the provider profit is maximised.

Furthermore, an elastic replica management is considered. The proposed strategy avoids unnecessary replications by removing replicas when these replicas are not required. This decreases both the storage and bandwidth consumptions while the already existing replication in MongoDB consumes more resources, i.e., more replicas.

Figure 4 Number of replications



5 Related work

In the existing MongoDB balance algorithm, data are splitted and distributed unevenly among different shards. However, data distribution will directly affect the system performance (Hamrouni et al., 2015). In order to solve this problem, the authors in Liu et al. (2012) propose an improved algorithm named Frequency of Data Operation (FODO) that takes into consideration the load of servers and the frequency of data operations. The proposed algorithm can balance the data among shards, which improves the cluster's operations (read and write) performances. Gu et al. (2015) analyse different existing mechanisms of replication already used in MongoDB. The authors gave a profound explanation and description of the two used models of replication: Master/slave and replica-set models. An interesting study (Lima et al., 2016) has been conducted to evaluate the impact of biological data replication on MongoDB and Cassandra. Some setting in the cluster and replication factors can certainly affect the results. Results showed an improvement in the data availability, but a loss in system performance and the impact of the replication factor on the execution time was higher. Haughian et al. (2016) evaluate and examined the impact on performance when using or not the replication under different levels of workloads and consistency. They experimented with MongoDB and Cassandra NoSQL data stores. Using benchmarking experiment's results, they affirm that replication and consistency have a direct impact on system performance. Tauro et al. (2013) present a comprehensive analysis of different existing NoSQL data stores, including MongoDB, and their features. Basing on the data model, querying, and replication model, this research provides knowledge for tenants to choose the appropriate NoSQL database.

When analysing different existing propositions that deal with replication mechanisms in MongoDB, we remark that most of them focus their attention on evaluating and benchmarking replication. Further researches gave a performance comparison between MongoDB and other popular NoSQL systems, as well as relational database management systems. However, no one of them draw attention to the replica management in MongoDB when used as the back-end by a cloud service provider. Furthermore, they neglect the provider profit. In this paper, we dealt with replica management in MongoDB in order to ensure both system performance and profitability of the cloud provider.

6 Conclusion

We propose a new data replication strategy for MongoDB, a document oriented NoSQL database Engine. The main aim of the proposed strategy is to satisfy the performance requirement for the tenant while the provider profit is taken into account. The replication is triggered only if the estimated response time of a submitted tenant query Q is greater than a response time threshold defined in the SLA. Then, the replication must be profitable for the provider when a replication is considered. We based on the estimation of: (i) the response time of Q before its execution, and (ii) both revenues and expenditures of the provider when considering data replication. Then, the most popular data is replicated on the less loaded node. Furthermore, existing replicas that have the lowest popularity are dynamically removed in order to decrease the resource consumption which increases the provider profit. We evaluated the proposed strategy through an experimental evaluation. We compared the actual existing replication strategy in MongoDB with our proposed strategy in term of response time and number of triggered replication. The obtained results confirm that the proposed strategy provides better results when considering simultaneously the tenant response time satisfaction and the provider economic profitability. As a future work, we intend to extend the proposed replica creation decision process by accepting to lose some profit when the provider executes queries of an important tenant within a multi-tenant environment. This should be advantageous for the provider within a long period with a condition that the provider will make again a profit with the same tenant afterwards. Also, we plan to implement the proposed strategy in a real cloud environment.

References

Abubakar, Y., Adeyi, T.S. and Auta, I.G. (2014) 'Performance evaluation of NoSQL systems using YCSB in a resource austere environment', *Performance Evaluation*, Vol. 7, No. 8, doi:10.5120/ijais14-451229.

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D, Rabkin, A., Stoica, L. and Zaharia, M. (2010) 'A view of cloud computing', *Communication of the ACM*, Vol. 53, No. 4, pp.50–58.

Benoit, A., Rehn-Sonigo, V. and Robert, Y. (2008) 'Replica placement and access policies in tree networks', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 19, No. 12, pp.1614–1627.

Chodorow, K. (2013) *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*, O'Reilly Media, Inc., O'Reilly.

Goel, S. and Buyya, R. (2007) 'Data replication strategies in wide-area distributed systems', *Proceedings of the Enterprise Service Computing: From Concept to Deployment*, pp.211–241.

Gu, Y., Wang, X., Shen, S., Ji, S. and Wang, J. (2015) 'Analysis of data replication mechanism in NoSQL database MongoDB', *Proceedings of the IEEE International Conference on Consumer Electronics-Taiwan*, pp.66–67.

Hameurlain, A. and Mokadem, R. (2017) SPECIAL ISSUE, *Elastic Data Management in Cloud Systems*, pp.261–262.

Hamrouni, T., Hamdeni, C., Ben Charrada, F. (2015) 'Impact of the distribution quality of file replicas on replication strategies', *Journal Networking Computing*, Vol. 56, pp.60–76.

Haughian, G., Osman, R. and Knottenbelt, W.J. (2016) 'Benchmarking replication in cassandra and mongodb nosql datastores', *Proceedings of the International Conference on Database and Expert Systems Applications*, International Publishing, Springer, pp.152–166.

Liu, Y., Wang, Y. and Jin, Y. (2012) 'Research on the improvement of MongoDB Auto-Sharding in cloud environment', *Proceedings of the 7th IEEE International Conference on Computer Science & Education*, pp.851–854.

Lima, I., Oliveira, M., Kieckbusch, D., Holanda, M., Walter, M.E. M., Araújo, A. and Lifschitz, S. (2016) 'An evaluation of data replication for bioinformatics workflows on NoSQL systems', *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*, pp.896–901.

Loukopoulos, T., Lampsas, P. and Ahmad, I. (2005) 'Continuous replica placement schemes in distributed systems', *Proceedings of the 19th annual international conference on Supercomputing*, pp.284–292.

Mansouri, N. and Asadi, A. (2014) 'Weighted data replication strategy for data grid considering economic approach', *International Journal of Computer Electrical Automation Control Information Engineering*, Vol. 8, pp.1336–1345.

Mansouri, Y., Azad, S.T. and Chamkori, A. (2014) 'Minimizing cost of K-replica in hierarchical data grid environment', *Proceedings of the IEEE 28th International Conference on Advanced Information Networking and Applications*, pp.1073–1080.

Membrey, P., Plugge, E. and Hawkins, T. (2010) *The Definitive Guide to MongoDB: the noSQL Database for Cloud and Desktop Computing*, Springer.

Mohamed, H.H.H. (2015) *A New Auditing Mechanism for Open Source NoSQL Database a Case Study on Open Source MongoDB Database*, PhD Dissertation, Universiti Utara Malaysia.

Mokadem, R. and Hameurlain, A. (2015) 'Data replication strategies with performance objective in data grid systems: a survey', *International Journal of Grid and Utility Computing*, Vol. 6, No. 1, pp.30–46.

Özsu, M.T. and Valduriez, P. (2011) *Principles of Distributed Database Systems*, Science & Business Media, Springer.

Park, S.M., Kim, J.H., Ko, Y.B. and Yoon, W.S. (2003, December) 'Dynamic data grid replication strategy based on Internet hierarchy', *Proceedings of the International Conference on Grid and Cooperative Computing*, Springer, Berlin, Heidelberg, pp.838–846.

Perez, J.M., García-Carballeira, F., Carretero, J., Calderón, A. and Fernández, J. (2010) 'Branch replication scheme: a new model for data replication in large scale data grids', *Future Generation Computer Systems*, Vol. 26, No. 1, pp.12–20.

Rao, B.R.M. and Govardhan, A. (2013) 'Sharded Parallel Mapreduce in Mongodb for online aggregation', *International Journal of Engineering and Innovative Technology*, Vol. 3, No. 4, pp.119–127.

Sakr, S. and Liu, A. (2012) 'Sla-based and consumer-centric dynamic provisioning for cloud databases', *Proceedings of the IEEE 5th International Conference on Cloud Computing (CLOUD)*, pp.360–367.

Stantchev, V. and Schröpfer, C. (2009) 'Negotiating and enforcing QoS and SLAS in grid and cloud computing', *Advances in Grid and Pervasive Computing*, pp.25–35.

Sousa, F.R. and Machado, J.C. (2012) 'Towards elastic multi-tenant database replication with quality of service', *Proceedings of the IEEE/ACM 5th International Conference on Utility and Cloud Computing*, IEEE Computer Society, pp.168–75.

Sun, D.W., Chang, G.R., Gao, S., Jin, L.Z. and Wang, X.W. (2012) 'Modeling a dynamic data replication strategy to increase system availability in cloud computing environments', *Journal of Computer Science and Technology*, Vol. 27, No. 2, pp.256–272.

Tabet, K., Mokadem, R., Laouar, M.R. and Eom, S. (2017) 'Data replication in cloud systems: a survey', *International Journal of Information Systems and Social Change*, Vol. 8, No. 3, pp.17–33.

Tauro, C.J., Patil, B.R. and Prashanth, K.R. (2013) 'A comparative analysis of different nosql databases on data model, query model and replication model', *Proceedings of the International Conference on Emerging Research in Computing, information, Communication and Applications*, Elsevier Publications, pp.14–25.

Tos, U., Mokadem, R., Hameurlain, A., Ayav, T. and Bora, S. (2016) 'A performance and profit oriented data replication strategy for cloud systems', *Proceedings of the IEEE Conferences Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pp.780–787.

Tos, U., Mokadem, R., Hameurlain, A., Ayav, T. and Bora, S. (2017) 'Ensuring performance and provider profit through data replication in cloud systems', *Cluster Computing*, pp.1–14.

Tos, U. (2017) *Réplication de Données Dans les Systèmes de Gestion de Données à Grande échelle*, PhD Dissertation, Université de Toulouse, Université Toulouse III-Paul Sabatier.

Tu, M., Li, P., Xiao, L., Yen, I.L. and Bastani, F.B. (2006) 'Replica placement algorithms for mobile transaction systems', *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 7, pp.954–970.

Wei, Q., Veeravalli, B., Gong, B., Zeng, L. and Feng, D. (2010) 'CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster', *Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER)*, pp.188–196.

Xhafa, F., Kolici, V., Potlog, A.D., Spaho, E., Barolli, L. and Takizawa, M. (2012) 'Data replication in P2P collaborative systems', *Proceedings of the IEEE 7th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pp.49–57.

Xue, M., Shen, J. and Guo, X. (2015) 'Replica placement in cloud storage based on minimal blocking probability', *Proceedings of (CENet'15)*, Shanghai, China.