Frontiers of Information Technology & Electronic Engineering
www.zju.edu.cn/jzus; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

A locality-based replication manager for data cloud

Reza SOOKHTSARAEI[†‡1], Javad ARTIN[1], Ali GHORBANI[2], Ahmad FARAAHI[1], Hadi ADINEH[1]

([1]*Department of Computer Engineering and Information Technology, Payame Noor University, Tehran, Iran*)

([2]*Department of Industrial Engineering, Faculty of Engineering, Maziar University, Noor, Iran*)

[†]E-mail: reza.sookhtsaraei@gmail.com

Received Nov. 10, 2015; Revision accepted Feb. 16, 2016; Crosschecked Nov. 14, 2016

Abstract: Efficient data management is a key issue for environments distributed on a large scale such as the data cloud. This can be taken into account by replicating the data. The replication of data reduces the time of service and the delay in availability, increases the availability, and optimizes the distribution of load in the system. It is worth mentioning, however, that with the replication of data, the use of resources and energy increases due to the storing of copies of the data. We suggest a replication manager that decreases the cost of using resources, energy, and the delay in the system, and also increases the availability of the system. To reach this aim, the suggested replication manager, called the locality replication manager (LRM), works by using two important algorithms that use the physical adjacency feature of blocks. In addition, a set of simulations are reported to show that LRM can be a suitable option for distributed systems as it uses less energy and resources, optimizes the distribution of load, and has more availability and less delay.

Keywords: Data cloud, Replication, Graph, Locality replication manager (LRM)
http://dx.doi.org/10.1631/FITEE.1500391 **CLC number:** TP393

1 Introduction

Cloud computing provides computational resources and scalable storage using the Internet (Armbrust *et al*., 2009; Creeger, 2009; Dikaiakos *et al.*, 2009). The users can use the services at any location by cloud computing. This method is similar to using public services such as water, gas, and telephones which are used by the public according to their needs (Buyya *et al*., 2009).

Because these days different sciences use a large amount of data, the flexibility and clarity of cloud services can be used for data management and more effective use of data-based functions. In many sciences, the capacity of data is shown in terabytes and petabytes. Currently, the replication of data is used as an effective technique for managing the efficiency of such spacious data. The replication of data has many advantages such as more access to data, less delay in access, and increased availability.

To obtain an efficient replication of data, two important questions should be answered. The first question is how many copies should be made from each data to satisfy the needs of the system. The larger the number of copies, the more the space and the energy the system needs for storage and use, respectively. On the other hand, a fixed number of copies are not a suitable option to obtain an effective replication of data. For instance, cloud storage systems such as the Hadoop distributed file system (HDFS) (Shvachko *et al*., 2010), Amazon S3 (Amazon, 2008), and Google file system (GFS) (Ghemawat *et al.*, 2003) all use three copies of data by assumption. It was shown by Li *et al.* (2011) that, generally, a fixed number of copies may not be the best way to arrive at the solution. The second question is about the place where the copies are stored, in other words, where each copy should be placed to perform the duties faster and to ensure that the load

‡ Corresponding author
ORCID: Reza SOOKHTSARAEI, http://orcid.org/0000-0002-6444-3201

is distributed in a balanced way. These two questions constitute the problem of data replication, for which an efficient method is suggested in this paper.

Although many studies have been conducted on data replication (Ranganathan and Foster, 2001; Ghemawat *et al*., 2003; Lamehamedi *et al.*, 2003; Wei *et al*., 2010; Li *et al*., 2011; Nukarapu *et al.*, 2011; Choi and Youn, 2012), very few have examined the physical adjacency of blocks for one single inquiry. Many studies have focused on parameters such as availability, quick response, and effective function. However, to reach the optimum values of these parameters, the issue of the physical location of data used by one user should be taken into account. Moreover, previous studies have ignored the number of copies to be stored, which is an important issue. The larger the number of copies of one dataset in the system, the more the resources (such as storage capacity and energy) used. Therefore, fewer copies should be made in the system to avoid waste of resources as much as possible, and also to ensure that the above-mentioned parameters are not affected greatly.

Based on what has been mentioned above, a locality-based replication manager is presented for data-based functions in cloud storage. This replication manager, which is henceforth called the locality replication manager (LRM), takes into account not only the needs of the quality-of-service (QoS) function, but also the physical locality of data blocks in one inquiry, to reach more optimum replication parameters such as higher availability and faster response. The main goals of LRM can be summarized as follows: (1) decreasing the number of copies, (2) increasing the availability, (3) satisfying the needs of QoS, and (4) decreasing the operational expenses such as the storage space and energy.

2 Related studies

Replication has been extensively researched in the World Wide Web (WWW) (Qiu *et al.*, 2001), peer-to-peer networks (Aazami *et al*., 2004), as well as ad hoc and sensor networks (Intanagonwiwat *et al*., 2000; Tang *et al*., 2008). Recently, it has reappeared as a new topic of research with the emergence of distributed systems at a large scale such as the grid (Dabrowski, 2009) and the cloud (Bonvin, 2009).

Data replication techniques can be classified in two groups: static and dynamic. In static replications (Ghemawat *et al*., 2003; Rahman *et al*., 2006; Shvachko *et al*., 2010), the number of copies and the nodes of their hosts are predetermined. In dynamic replication (Tang and Xu, 2005; Chang and Chang, 2008; Lei *et al.*, 2008; Dogan, 2009; Li *et al.*, 2012), the number and place of copies are determined based on the users' resource needs, and changes in the pattern of their access in a smart way. Among the disadvantages of dynamic replications, there are difficulties of data collection from all nodes in a complex infrastructure like the cloud, while maintaining the compatibility of the data in such an environment.

Ghemawat *et al*. (2003) presented a static algorithm for distributed replication that takes into account three important factors in decision making. First, in this algorithm, some service providers are selected to host the copies, in which the storage space is smaller than the average storage space used in the whole system. Second, fewer copies of data are made on such service providers. The third factor takes into account the distribution of load. In other words, service providers are selected in such a way that the copies are distributed all over the rack.

Li *et al.* (2011) presented a new dynamic economical algorithm for data replication called cost-effective incremental replication (CIR) for the cloud. In this algorithm, the technique of increasing replication has been used to reduce the number of copies and provide the required assurance. The cost of storing the data, especially when the data needs to be stored for a short time or when the data needs a low level of assurance, is remarkably reduced. Wei *et al*. (2010) considered a dynamic algorithm for distributed replication called cost-effective dynamic replication management (CDRM), which focuses on the relationship between the availability and the number of copies. In this algorithm, the number of copies is kept as a minimum for a certain amount of availability. Wei *et al*. (2010) presented six different replication algorithms for three different patterns of access, aiming at reducing the time of access and the use of bandwidth. These six algorithms are as follows: no replication or caching, best client, cascading replication, plain caching, caching plus cascad-

ing replication, and quick extension. Nukarapu *et al.* (2011) presented a data replication algorithm called the centralized dynamic replication algorithm (CDRA) which reduces the overall time of accessing the file by taking into account the limited space for caching in grid sites.

Based on CDRA, a distributed algorithm was presented for caching, where the grid sites react quickly to the condition of the site and make smart decisions for caching, which are easily adapted in a distributed environment like the data grid. This method can reduce the overall delay of access to the data file by half the size of the optimum solutions.

The limitation of this method is that it takes into account only the cost of access. Choi and Youn (2012) presented a dynamic synthetic protocol that, in a very efficient way, synthesizes the grid network with the tree structure. By doing this, it creates a flexible topology based on the three parameters of height and depth of the tree and the number of slips in each node. In this method, the behaviors of reading/writing and writing/writing are easily recognized to keep the compatibility of the data. Li *et al.* (2012) introduced a reliable and economical data management mechanism, which reduces the copies in the system by controlling the active copies and simultaneously ensures that the needs of the data are me. The caching space used is also reduced. Tang and Xu (2005) showed a way for placing the data items in the best service provider.

Considering this method of replication, each client can refer to the closest service center to access his/her data. In addition, two exploratory algorithms were presented to gradually omit and add data copies, in which the QoS of each enquiry was also taken into account. In addition to the solutions that optimize the number of copies and their place, there are some solutions in replication that focus on optimization by updating the copies (Tu *et al.*, 2007). The most important drawback of such solutions is considering a limited set of effective parameters in replication. Moreover, they simply optimized the efficiency of the system by ignoring the energy used in the data centers. Hassan *et al.* (2009) presented a complementary method to reach an optimum for replication. Their study deals with the delay of the system, its storage capability, and the level of certainty, but did not pay attention to the energy used in the system.

This paper introduces a transaction manager called the locality replication manager (LRM), which not only takes into account important factors in replication but also uses the locality in the requested blocks to optimize the factors.

3 System model and purposes of optimization

In this section, the model of the system and the derived formulas are explained.

3.1 Model of the system

This storage consists of some clusters that share the data and other resources in an efficient way. The main component of these resources is the distributed file system. The Google File System (GFS), Hadoop Distributed File System (HDFS), and Amazon Simple Storing Service (S3) are three well-known instances. In this paper, the architecture of HDFS, (Fig. 1), is used for replication management. We assume that each file consists of some blocks that have been distributed in the data nodes of this file system. We use the 'name node' as a coordinator in replication management. Henceforth, we recognize the node as the coordinator of the elements (Fig. 2).

Locality replication manager (LRM): it is the replication manager referred to in this study. The main duty of LRM is to receive the inquiries of the users, collect the condition of data nodes in the cluster, and finally decide the best host to place the blocks. LRM performs these duties in coordination with its other components. In other words, LRM is the final decision maker.

Graph directory table (GDT): It is the table managed by LRM and includes very important information from the system, such as blocks and their graphs, the number of accesses to each block in the graph, the host of each graph, and the maximum delay in accessing each graph.

Graph constructor (GC): It constructs a full graph from the available blocks in each inquiry, and delivers it to LRM for placement decisions.

Availability and delay system care (ADSC): When LRM finds that the system is not in an appropriate level regarding the delay and availability, this component starts working by receiving messages

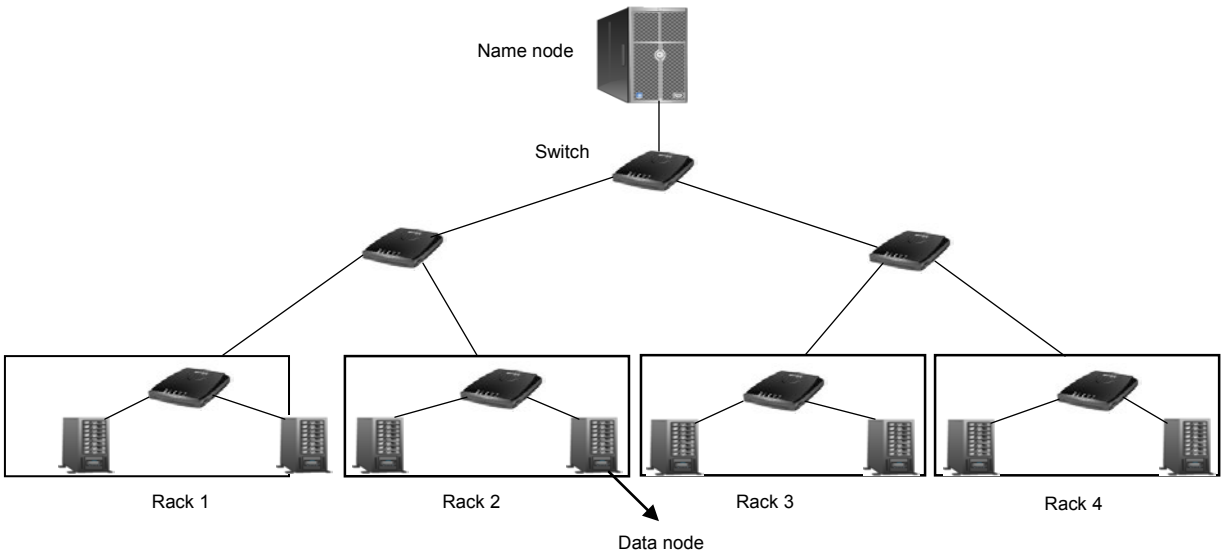

Fig. 1 Architecture of the Hadoop distributed file system

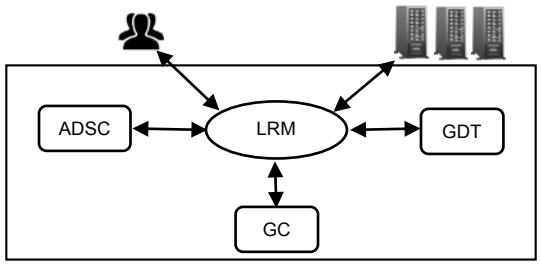

Fig. 2 Components of the coordinator

ADSC: availability and delay system care; LRM: locality replication manager; GDT: graph directory table; GC: graph constructor

from LRM. ADSC determines the suitable data nodes to place the graphs in the system again and then sends this information to the system. LRM changes the host of the graphs and simultaneously updates GDT by receiving information from ADSC.

We assume that all the data follow the policy of 'writing once, reading several times' (Ghemawat *et al.*, 2003). Therefore, we do not need a mechanism to adapt the data.

3.2 Symbols and functions

Table 1 lists all the symbols used in this paper; the method of using these symbols will be presented later.

3.2.1 Availability

The first goal of cloud storing clusters is to provide the highest level of availability for blocks and also for their graphs. Assume that the decision variable $\theta(i, j)$ is equal to 1 if B_i is on the data node of m_j; else, we regard it as zero.

In addition, we determine P_j as a possible failure of data node m_j ($1 \leqslant j \leqslant M$). The failure for data nodes occurs randomly. Each block may exist in several inquiries, each of which is considered as a complete graph and distributed in several nodes. A block is not available if one node (block) is not available in the graph (inquiry), and one graph is not available when all its blocks are not available. Therefore, the probability of availability block being available in the system is determined by

$$P(B_i) = 1 - \prod_{j=1}^{M} \theta(i, j) \cdot P_j , \tag{1}$$

Since the availability of all blocks in a graph is more important than the availability of one block, the availability of one graph inquiry is defined as

$$P(G_k) = \prod_{i=1}^{|G_k|} P(B_i). \tag{2}$$

3.2.2 Delay

Minimizing the delay of each storing system is a key issue. This delay depends on the bandwidth and the rate of transfer of the storing disks.

Table 1 Used parameters

Variable	Explanation
B_i	Block i in the file, $1 \le i \le B$
M	Number of nodes (physical nodes)
m_j	Node j (physical node of j)
Storage_j	Storage of node j
α	Coefficient for storage space
load_j	Load of node j
p_j	Probability of failure for node j, $1 \le j \le m$
G	Dependence graph
G_k	The kth graph, $1 \le k \le G$
G_{new}	New graph
$\|G_k\|$	Number of nodes in graph (group) k
$\|G\|$	Number of available graphs in the storage cloud
S	Storage cloud
$S(G_k)$	Number of available nodes in graph k
$S(m_j)$	Number of available nodes in node j
$P(B_i)$	Probability of being available for block i
$P(G_k)$	Probability of being available for graph k
$P(S)$	Probability of the storage cloud being available
Band_j	Bandwidth for node j
$A(i, j)$	Delay in accessing block i in node j
r_i	Number of copies for block i
$\overline{L_{B_i}}$	Average delay for block i
$\overline{L_{G_k}}$	Average delay for graph k
$\overline{L_S}$	Average delay of the system
t_i	Time for block i
$\text{QoS}(\text{inq}_i)$	QoS of inquiry i, which is here the maximum tolerable delay for the inquiry
δ	Coefficient for the appropriacy of data cloud

Therefore, if the blocks are placed on the data nodes with a maximum bandwidth and a higher transfer rate, there will be less delay in data access. Since each block has several copies, the delay of B_i is shown by

$$\overline{L_{B_i}} = \frac{1}{r_i} \sum_{j=1}^{M} \theta(i, j) \cdot A(i, j), \tag{3}$$

where $A(i, j)$ is the delay caused by the bandwidth and data transfer in data node m_j.

Since the delay of a group of blocks (the inquiry graph) is more important than the delay of one block, we have

$$\overline{L_{G_k}} = \frac{1}{|G_k|} \sum_{i=1}^{|G_k|} \overline{L_{B_i}}. \tag{4}$$

3.2.3 Function of target

The function of the presented target in this section is to select a data node to host an inquiry graph, stated as

$$\max[(\text{Qos}(\text{req}_i) - \overline{L_{G_{\text{new}}}}) + P(G_{\text{new}}) + |S(G_{\text{new}}) \cap S(m_j)| + \text{storage}_j \cdot \alpha - \text{load}_j]. \tag{5}$$

Before presenting the target function, it is necessary to explain the actions taken by LRM before leaving the copies and using the function.

First, each inquiry of the user is sent to LRM. LRM delivers the inquiry to a GC unit, and then receives the result in the form of a complete graph. After this step, LRM enters the stage of replication management. To manage replication in the data cloud, two steps must be followed:

1. Choosing the copy: This means choosing the best copies for one inquiry. To choose a copy, the inquiry of the user is submitted to the LRM in the form of a graph. LRM looks for a node that already has that graph, or whether the new graph is a subset of a graph existing in that graph. After finding the desired node(s), the QoS of the new inquiry (the maximum tolerable delay) is examined with any of the found nodes. The first node that can satisfy the QoS of the inquiry graph is selected by LRM, and the inquiry graph leads to that node. However, if there is no node that has a copy of the inquiry graph, or such a node does exist, but its inquiry graph cannot satisfy its QoS, it will work as follows: First, LRM lists all the nodes that have part of the new graph and arranges them in ascending order based on QoS that they can satisfy; Then, a number of nodes that cover all the nodes in the inquiry graph are selected, and the average QoS provided by such nodes is measured. If the obtained average QoS can satisfy the QoS of the inquiry graph, these nodes are recorded in GDT as the host group of the new graph. If none of the copy selection methods work to choose a node or nodes to host the new graph, we try replica placement.

2. Replica placement: It refers to placing the replica in the best data node. If LRM cannot find a node for the QoS of the inquiry using the replica selection method, it selects a node from other nodes that satisfy function (5).

Function (5) is the function of the suggested goal in this study. The two noticeable statements in this formula are $\text{storage}_j \cdot \alpha - \text{load}_j$ and $\text{Qos}(\text{req}_i) - \overline{L_{G_{\text{new}}}}$.

With the first statement, the load distribution is balanced in the data cloud, and, with the second, a node that can satisfy the QoS of the inquiry graph is selected, which has been taken here as the maximum tolerable delay. By the statement $|S(G_{\text{new}}) \cap S(m_j)|$ in function (5), a node is selected for inquiry graph placement that has the most commonality with the target graph. Using this statement causes the existing number of copies from each block to reach a minimum level.

4 Locality replication method

It can be concluded that LRM is the core of data replication management. The key duties of LRM are: (1) receiving the inquiry and sending it to appropriate node(s) to satisfy the desired quality by the user, and (2) taking care of the availability and the delay of the system and keeping them at a desired level. The details of these two duties are explained below.

4.1 Finding appropriate node(s) to send the inquiry

The details of this action are presented in Algorithm 1. First, the blocks of the files are randomly placed on the physical nodes in this algorithm. After receiving each inquiry for the blocks, these steps are executed as follows: The coordinator first receives the inquiry and then makes a graph for it (G_{new}). The coordinator looks for a node or a set of nodes in GDT, which includes the graph or part of it and can satisfy the QoS of the inquiry (lines 5 and 6). The result of this search may be a node or a set of nodes. If the result of the search is one node (lines 8 and 10), the coordinator leads the inquiry to that node. If the result is a set of nodes (lines 10–16), the coordinator first arranges them in ascending order based on the QoS they can satisfy. Then, it selects the nodes from the beginning of the list until the physical nodes cover all the new graph nodes. After covering all the graph nodes, if the average QoS of the selecte nodes can satisfy the QoS of the inquiry, then this set of physical nodes is recorded as the host of the new graph in the coordinator.

Algorithm 1 Finding the appropriate node(s) to send the inquiry

Input: file f with b blocks.
Output: QoS-based distributed replicas on physical nodes.
1 Distribute b blocks of file f on physical nodes randomly;
2 **for** each input inquiry inq_i **do**
3 Coordinator receives inq_i;
4 Form a graph G_{new} for inq_i;
5 Search t=a node (or a set of nodes) in $S(M)$ which G_{new} (part of G_{new}) hosts;
6 Meet $\text{QoS}(\text{inq}_i)$ in the graph directory table;
7 **if** $t \neq \varnothing$ **then**
8 **if** t is a node **then**
9 Coordinator redirects inq_i to t;
10 **else** t is several nodes **then**
11 Coordinator sorts t based on the supported QoS non-descendingly;
12 $S=\varnothing$;
13 **while** nodes in S do not host G_{new} **do**
14 $S=S\cup$(the first node of t);
15 **end while**
16 **if** average $\text{QoS}(s)$ meets $\text{QoS}(\text{inq}_i)$ **then**
17 Register s in the coordinator;
18 **end if**
19 **end if**
20 Coordinator changes the access field for each node (block) in G in the directory group table;
21 Coordinator removes each node in G and t whose access field reaches zero;
22 (Replica evaporation) except for the replica which is the last from the main version;
23 **else**
24 Coordinator lies in a new replica for each node in G_{new} in a physical node;
25 Calculate the minimum value by function (5);
26 Coordinator registers a G_{new} in the graph directory table;
27 **end if**
28 **end for**

After the replica is selected, the last step is to change the access fields of nodes in the graph and evaporate the replica (lines 20–22). It is possible that, due to different accesses to graphs, some nodes are accessed more, some are accessed less, and some are never accessed. The access field of every node increases in the coordinator as for each access to the node, and no access causes the field to decrease such that, when it reaches zero, it means that the node is not effective in the accesses of the graph, and should be omitted from the graph by the coordinator. Before omitting the replica, the coordinator examines whether the block is the last replica of the original block (whether the two versions remain from each

block, including the original version and the replica). If the answer is yes, the coordinator will prevent the block from being omitted. Fig. 3 shows a sample of sending an inquiry to LRM, creating a graph of inquiry, and evaporating the replica.

Now we get back to Algorithm 1. If the coordinator cannot find any graph from the existing graphs in the cloud (lines 5 and 6), it puts a new replica from each of the existing blocks in the graph on the node to minimize function (5). After the node is found, the new graph is recorded in the coordinator with the related node (lines 24–26).

4.2 Taking care of availability and delay of the system

To reach this goal, if the target QoS is violated in δ of the inquiries, LRM orders ADSC to restructure the system. By restructuring we mean placing the graph inquiries again on the physical nodes to keep the availability and delay of the system at a desired level. ADSC also reacts to Algorithm 2 by receiving this order. As can be found from this algorithm, ADSC uses the genetic complementary algorithm to do this.

Algorithm 2 Keeping availability and delay of system at a desired level

Input: number of groups of requested blocks G, number of physical nodes N, maximum generation number Gens, and current generation t.

Output: near optimal mapping $G \rightarrow N$.

1 Φ_t=Generate initial population randomly;
2 **while** (t<Gens) **do**
3 Evaluate the fitness of each individual in Φ_t based on function (6);
4 $\Phi_{temp}=Q$ highest individuals based on their fitness;
5 $\Phi_{t+1}=Q-L$ highest individuals based on their fitness from Q;
6 $\Phi_{t+1}=\Phi_{t+1}\cup$($L$ remaining individuals crossed from Q with a two-point cross-over);
7 $\Phi_{t+1}=\Phi_{t+1}\cup$($K$ individuals generated randomly);
8 Φ_{t+1}=Mutation(Φ_{t+1});
9 $t=t+1$;
10 **end while**

Genetic algorithms (GAs) are the exploratory methods that repeatedly search in a large space of data for an answer close to the optimum. Each possible solution is encoded in the form of a chromosome.

The group of chromosomes is called a 'population'. In GA, a primary population should be formed.

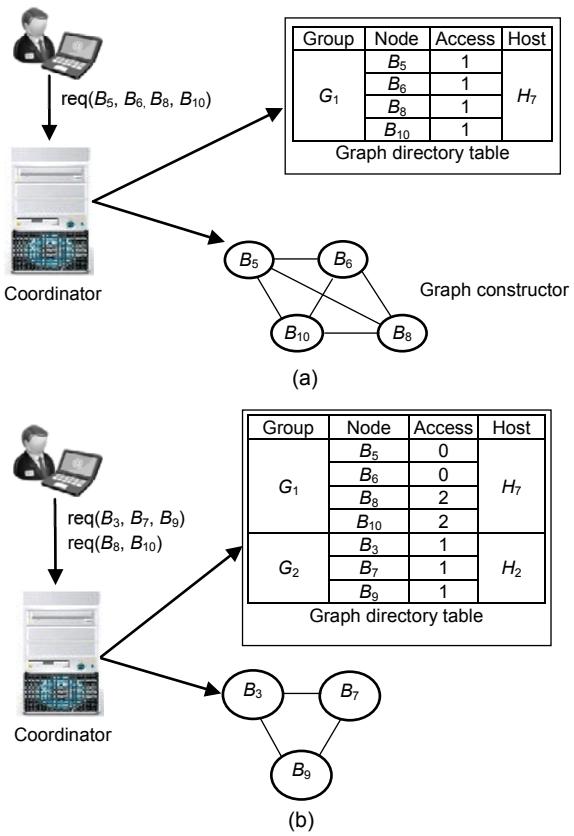

Fig. 3 Managing the graph directory table (GDT): (a) sending the inquiry to the locality replication manager (LRM), making the graph, and recording it in GDT; (b) updating the GDT by entering new inquiries into LRM

This population of the chromosomes, which are randomly made, is the starting point of the completion process. After the primary population is made, the selection step begins in which the chromosomes are selected or omitted for the next population based on their quality.

The next step is called the 'cross-over'. In this step, a few pairs of chromosomes are selected from the population, and some of their parameters are exchanged to create a pair of valid chromosomes. After the cross-over, it is time for mutation. In mutation, each chromosome changes into a valid chromosome from the population. After these steps, the new population is examined. A fit value is allocated to each chromosome by the objective function. Then, the new population is examined. Our purpose is to find a chromosome with the most optimum fitness value.

If the value is not met, the aforementioned steps are repeated to make a new population. This process goes on until the value is found. Now we study the steps and methods of the GA used in this study.

4.2.1 Encoding

To produce each chromosome, encoding was carried out (Fig. 4). One chromosome has been made for a limited number of graphs and physical nodes, and is presented as a set of integers.

4.2.2 Function of the target and selection

The appropriateness of the chromosomes is determined for selection in the population with the following function:

$$\min\{\overline{L_S} / P(S)\}, \tag{6}$$

where

$$P(S)=\prod_{i=1}^{|G|}P(G_i) \text{ and } \overline{L_S}=\frac{1}{|G|}\sum_{i=1}^{|G|}\overline{L_{G_i}}.$$

Function (6) shows the proportion of delay to the availability of the whole cloud system.

If Q is the total number of chromosomes in the primary population, then, the Q–K of the chromosomes, which have the highest appropriateness, are selected and transferred to the next population based on some conditions to be mentioned later in this paper. The K of the chromosomes for the next population is randomly made to prevent fast convergence from occurrence and to escape from being trapped in local minima.

4.2.3 Cross-over

From the Q–K chromosomes selected for the next generation, L chromosomes ($L<N–K$) are transfered to the new population by cross-over, where N is the number of physical nodes. As shown in Fig. 5, in the two-point cross-over, two indices are selected randomly, and the content between them is exchanged for the two chromosomes.

4.2.4 Mutation

The mutation step is done with a transfer rate of 0.5. The entry of each index selected for the mutation step which shows the physical node of the graph, is replaced with another physical node that is randomly obtained.

4.2.5 Genetic algorithm

Algorithm 2 shows all the required steps of the genetic algorithm, which is used to keep the availability and delay of the system at a desired level.

5 Time complexity

Before comparing the algorithm used in this study with existing algorithms, the time complexity of LRM in the worst situation is shown. Since LRM consists of two separate algorithms, the time analysis is done separately for each algorithm.

5.1 Time complexity of Algorithm 1

Theorem 1 If a file exists with b blocks, M data nodes and inq inquiries in the time period of Δt, the time complexity of this algorithm is equal to $O(\text{inq}\cdot b^2)$.

Proof As for Δt of time unit, the following steps are followed for each inquiry. Assume that all the blocks are requested in each inquiry (the worst condition), the time needed to make the graph of the inquiry will be $O(b^2)$ (line 4). In line 5, linear search is used to find the node(s) for which time of $O(M)$ is spent. To arrange the searched nodes for m ($m\leq M$), in line 5, the time needed is equal to $O(m\log_2 m)$ (line 11). In line 24, to find the physical node to place the graph of the new inquiry, the time complexity is equal to $O(Mb)$, where b is the maximum of the existing nodes in G_k. Since M is the upper

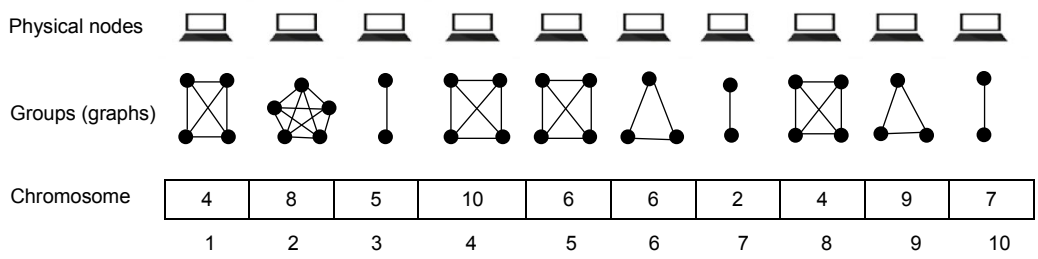

Fig. 4 Creation of a chromosome from physical nodes and graphs

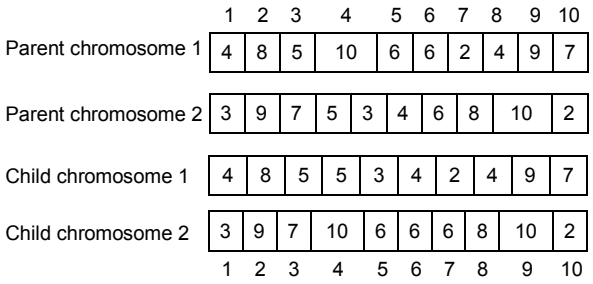

Fig. 5 Two-point cross-over

bound for m and assume $M<b$, the time complexity of this algorithm is equal to $O(\text{inq}\cdot b^2)$.

5.2 Time complexity of Algorithm 2

Theorem 2 Assume that there are $|G|$ graphs of inquiry existing in data cloud, N chromosomes in each population, and Gens generations, the time complexity of this algorithm is equal to $O(|G|)$.

Proof If the primary population has N chromosomes and the length of each chromosome is G, based on the number of graphs of inquiry, the time needed to make that population will be $O(N|G|)$.

To examine the suitability of each chromosome in line 5, a time equal to $O(N|G|)$ will be spent. To select the best chromosomes in line 6, we arrange them and this takes a time that is equal to $O(N\log N)$.

The time complexity of the cross-over in the worst condition will be equal to $O(L|G|)$ (line 8), if the distance between the two selected pairs equals the length of chromosome G. The time complexity for making a random chromosome K equals $O(K|G|)$ (line 9). The maximum time for mutation equals $O(N)$ (line 10), if all the chromosomes go under this process. Thus, the time complexity of Algorithm 2 is $O(\text{Gens}\cdot N\cdot|G|)$. However, since the values of N and Gens have been previously determined and will not change, we can conclude that the time complexity of this algorithm is equal to $O(|G|)$.

6 Assessing the efficiency

We use a series of simulations to assess the efficiency of LRM. The advantage of doing this is that we can easily change the parameters and examine the individual effect of each one on the efficiency of the system. We first study the simulation environment, and then compared the efficiencies of the other two algorithms with our suggested algorithm.

6.1 Simulation environment

We make a simulator called LRMsim using Java language to assess the advantages and the practicality of our idea. This simulator makes a modular simulation framework that helps us to model different structures of the data cloud and other resources. We can assess our idea and the efficiency of the method using these simulations. Simulation helps us determine different structures from network, different types of nodes, different resources for nodes, a method for replication, and different parameters.

Table 2 shows the configuration used in the data nodes, the algorithms used, and the inquiries entering into the system. The value for the number of generations in LRMsim was assumed to be 500. The reason for this was the fact that little optimization is obtained for values above 500.

The assumption is that the data cloud follows the policy of 'writing once, reading for several times', based on which the costs of adaptability and updating are ignored.

6.2 Results of assessment

We start with Fig. 6, which shows the number of replicas in each method. We compared our algorithm regarding the number of replicas with multi-objective evolutionary (MOE), multi-objective randomized greedy (MORG) (Hassan *et al.*, 2009), and Hadoop. In Hadoop, the factor of replica is fixed at 3, while it changes in other algorithms based on the conditions of the environment. Compared to the fixed replica factor, the results show that providing a dynamic number of replicas optimizes not only the use of resources, but also the efficiency obtained from the data cloud system. LRM reduces the cost of keeping a number of replicas using the dynamic factor of replica and a minimum number of the replicas, compared to the methods that do not use the fixed factor (Fig. 6).

In Fig. 7, the feature of locality of the requested blocks has been taken into account.

With a dynamic number of inquiries, fewer physical nodes will be accessed in LRM. The fewer the nodes accessed, the faster and with less delay the inquiries are satisfied.

One of the most important issues in replica management is reducing the costs of storing the replicas. The energy used to store these replicas is one

Table 2 Configuration of the locality replication manager

Parameter	Description
M	25 nodes (some of which are randomly placed in the rack)
Storage$_j$ (GB)	[100, 200, …, 1000]
p_j	[0.001, 0.002, …, 0.030]
$\|G_k\|$	[1, 2, …, 10]
$\|G\|$	Depending on the inquiries of the users
Band$_j$ (Mb)	1000
$A(i, j)$	[1, 2, …, 20]
r_i	Depending on the graph of that block
QoS (inq$_i$) (s)	[0.001, 0.002, …, 2.000]
α	0.2
Gen	500
Chromosome	15
Q	10
L	3
K	2
δ	25 violations of inqueries out of 100 cases

example of the costs. As shown in Fig. 8, fewer nodes are active in LRM to provide services to the requests, and the smaller the number of these nodes, the more the energy saved.

In Fig. 9, the distribution of the load is shown with a fixed number of inquiries as for different numbers of blocks. As shown in this figure, LRM has distributed the load in a more uniform way, compared to the other two algorithms.

Figs. 6–9 are the outputs of performing Algorithm 1, and comparing LRM with algorithms MOE and MORG. Also, we run Algorithm 2 to show that LRM is better than the other two algorithms, regarding availability and delay in the data cloud.

Fig. 10 shows that LRM performs better than MOE and MORG in availability, regarding the changes in system load. In addition, Fig. 11 shows that LRM, compared to MOE and MORG, reduces the delay of the system even under high loads.

Note that considering the locality of a request's blocks in LRM plays a key role in this reduction.

All the figures shown in this section lead us to conclude that LRM can successfully work as manager in the replication management for the data cloud.

7 Conclusions

This paper deals with the issue of replication management of blocks in one file for the data cloud

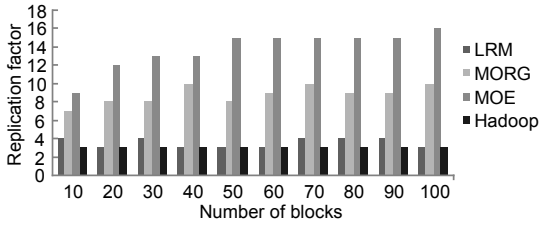

Fig. 6 Replication factor with each method
LRM: locality replication manager; MORG: multi-objective randomized greedy; MOE: multi-objective evolutionary

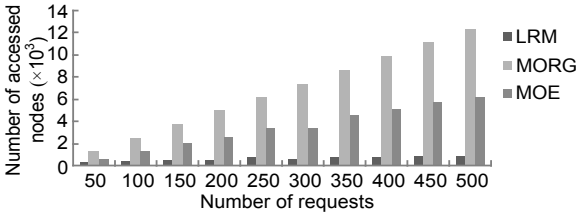

Fig. 7 Number of accessed physical nodes with each method
LRM: locality replication manager; MORG: multi-objective randomized greedy; MOE: multi-objective evolutionary

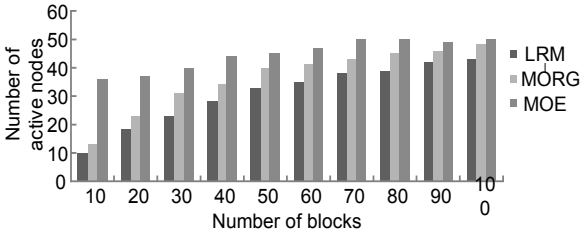

Fig. 8 Number of active physical nodes to manage the replicas with each method
LRM: locality replication manager; MORG: multi-objective randomized greedy; MOE: multi-objective evolutionary

to select a suitable replica factor and find a good place for the blocks. We could optimize factors such as the resources and energy, availability, and delay in the system by presenting an efficient manager for block replication. Therefore, we suggested an LRM using the concept of blocks' locality in one request and the complementary GA to solve the problem. To assess the success of LRM, we compared it to two other algorithms named MOE and MORG in a simulated environment written in Java. The results indicate that LRM performs better than the other two algorithms in a number of factors including the use of resources and energy, availability and delay of system.

In the future, we intend to assess the LRM on a real cluster of the data cloud. In addition, we would like to examine other algorithms of replication and

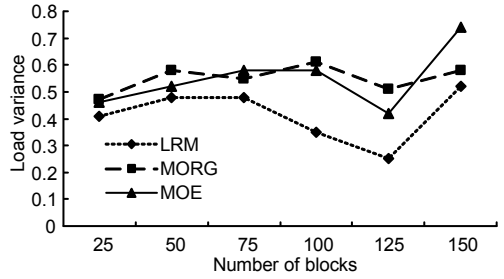

Fig. 9 Distribution of load in physical nodes with each method

LRM: locality replication manager; MORG: multi-objective randomized greedy; MOE: multi-objective evolutionary

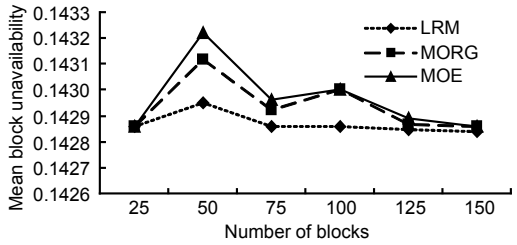

Fig. 10 System's average rate of not being available with each method

LRM: locality replication manager; MORG: multi-objective randomized greedy; MOE: multi-objective evolutionary

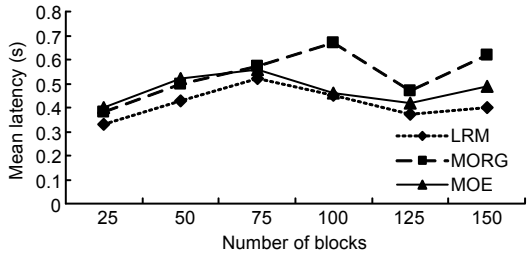

Fig. 11 Average delay in satisfying the requests with each method

LRM: locality replication manager; MORG: multi-objective randomized greedy; MOE: multi-objective evolutionary

different models of cost. Moreover, we intend to study the methods of online data movement to face the dynamic changes in the pattern of accessing blocks.

References

Aazami, A., Ghandeharizadeh, S., Helmi, T., 2004. Near optimal number of replicas for continuous media in ad-hoc networks of wireless devices. Proc. 1st Workshop on Multimedia Information Systems.

Amazon, 2008. Amazon Simple Storage Service (Amazon S3). Available from http://aws.amazon.com/s3.

Armbrust, M., Fox, A., Griffith, R., *et al*., 2009. Above the Clouds: a Berkeley View of Cloud Computing. Technical Report, No. UCB/EECS-2009-28, Department of EECS, California University, Berkeley.

Bonvin, N., Papaioannou, T.G., Aberer, K., 2009. Dynamic cost-efficient replication in data clouds. Proc. 1st Workshop on Automated Control for Datacenters and Clouds, p.49-56. http://dx.doi.org/10.1145/1555271.1555283

Buyya, R., Yeo, C.S., Venugopal, S., *et al*., 2009. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the fifth utility. *Fut. Gener. Comput. Syst.*, **25**(6):599-616. http://dx.doi.org/10.1016/j.future.2008.12.001

Chang, R.S., Chang, H.P., 2008. A dynamic data replication strategy using access weights in data grids. *J. Supercomput.*, **45**(3):277-295. http://dx.doi.org/10.1007/s11227-008-0172-6

Choi, S.C., Youn, H.Y., 2012. Dynamic hybrid replication effectively combining tree and grid topology. *J. Supercomput.*, **59**(3):1289-1311. http://dx.doi.org/10.1007/s11227-010-0536-6

Creeger, M., 2009. Cloud computing: an overview. *ACM Queue*, **7**(5):2-4.

Dabrowski, C., 2009. Reliability in grid computing systems. *Concurr. Comput. Pract. Exp.*, **21**(8):927-959. http://dx.doi.org/10.1002/cpe.v21:8

Dikaiakos, M.D., Katsaros, D., Mehra, P., *et al*., 2009. Cloud computing: distributed Internet computing for IT and scientific research. *IEEE Internet Comput.*, **13**(5):10-13. http://dx.doi.org/10.1109/MIC.2009.103

Doğan, A., 2009. A study on performance of dynamic file replication algorithms for real-time file access in data grids. *Fut. Gener. Comput. Syst.*, **25**(8):829-839. http://dx.doi.org/10.1016/j.future.2009.02.002

Ghemawat, S., Gobioff, H., Leung, S., 2003. The Google file system. Proc. 19th ACM Symp. on Operating Systems Principles, p.29-43. http://dx.doi.org/10.1145/1165389.945450

Hassan, O.A.H., Ramaswamy, L., Miller, J., *et al*., 2009. Replication in overlay networks: a multi-objective optimization approach. Int. Conf. on Collaborative Computing: Networking, Applications and Worksharing, p.512-528. http://dx.doi.org/10.1007/978-3-642-03354-4_39

Intanagonwiwat, C., Govindan, R., Estrin, D., 2000. Directed diffusion: a scalable and robust communication paradigm for sensor networks. Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, p.56-67. http://dx.doi.org/10.1145/345910.345920

Lamehamedi, H., Shentu, Z., Szymanski, B., *et al.*, 2003. Simulation of dynamic data replication strategies in data grids. Proc. Int. Parallel and Distributed Processing Symp. http://dx.doi.org/10.1109/IPDPS.2003.1213206

Lei, M., Vrbsky, S.V., Hong, X.Y., 2008. An on-line replication strategy to increase availability in data grids. *Fut. Gener. Comput. Syst.*, **24**(2):85-98. http://dx.doi.org/10.1016/j.future.2007.04.009

Li, W.H., Yang, Y., Yuan, D., 2011. A novel cost-effective dynamic data replication strategy for reliability in cloud

data centres. IEEE 9th Int. Conf. on Dependable, Autonomic and Secure Computing, p.496-502. http://dx.doi.org/10.1109/DASC.2011.95

Li, W.H., Yang, Y., Chen, J.J., *et al*., 2012. A cost-effective mechanism for cloud data reliability management based on proactive replica checking. 12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing, p.564-571. http://dx.doi.org/10.1109/CCGrid.2012.33

Nukarapu, D.T., Tang, B., Wang, L.Q., *et al*., 2011. Data replication in data intensive scientific applications with performance guarantee. *IEEE Trans. Parall. Distr. Syst.*, **22**(8):1299-1306. http://dx.doi.org/10.1109/TPDS.2010.207

Qiu, L.L., Padmanabhan, V.N., Voelker, G.M., 2001. On the placement of Web server replicas. Proc. IEEE 20th Annual Joint Conf. of the IEEE Computer and Communications Societies. http://dx.doi.org/10.1109/INFCOM.2001.916655

Rahman, R.M., Barker, K., Alhajj, R., 2006. Replica placement design with static optimality and dynamic maintainability. Proc. 6th IEEE Int. Symp. on Cluster Computing and the Grid, p.434-437. http://dx.doi.org/10.1109/CCGRID.2006.85

Ranganathan, K., Foster, I.T., 2001. Identifying dynamic replication strategies for a high-performance data grid. Int. Workshop on Grid Computing, p.75-86. http://dx.doi.org/10.1007/3-540-45644-9_8

Shvachko, K., Hairong, K., Radia, S., *et al*., 2010. The Hadoop distributed file system. IEEE 26th Symp. on Mass Storage Systems and Technologies, p.1-10. http://dx.doi.org/10.1109/MSST.2010.5496972

Tang, B., Das, S.R., Gupta, H., 2008. Benefit-based data caching in ad hoc networks. *IEEE Trans. Mob. Comput.*, **7**(3):289-304. http://dx.doi.org/10.1109/TMC.2007.70770

Tang, X., Xu, J., 2005. QoS-aware replica placement for content distribution. *IEEE Trans. Parall. Distr. Syst.*, **16**(10):921-932. http://dx.doi.org/10.1109/TPDS.2005.126

Tu, M., Tadayon, T., Xia, Z., *et al*., 2007. A secure and scalable update protocol for P2P data grids. 10th IEEE High Assurance Systems Engineering Symp., p.423-424. http://dx.doi.org/10.1109/HASE.2007.40

Wei, Q., Veeravalli, B., Gong, B., *et al*., 2010. CDRM: a cost-effective dynamic replication management scheme for cloud storage cluster. IEEE Int. Conf. on Cluster Computing, p.188-196. http://dx.doi.org/10.1109/CLUSTER.2010.24