QoS-Aware Data Replication for Data-Intensive Applications in Cloud Computing Systems

Jenn-Wei Lin, Chien-Hung Chen, and J. Morris Chang, *Senior Member*, *IEEE*

Abstract—Cloud computing provides scalable computing and storage resources. More and more data-intensive applications are developed in this computing environment. Different applications have different quality-of-service (QoS) requirements. To continuously support the QoS requirement of an application after data corruption, we propose two QoS-aware data replication (QADR) algorithms in cloud computing systems. The first algorithm adopts the intuitive idea of high-QoS first-replication (HQFR) to perform data replication. However, this greedy algorithm cannot minimize the data replication cost and the number of QoS-violated data replicas. To achieve these two minimum objectives, the second algorithm transforms the QADR problem into the well-known minimum-cost maximum-flow (MCMF) problem. By applying the existing MCMF algorithm to solve the QADR problem, the second algorithm can produce the optimal solution to the QADR problem in polynomial time, but it takes more computational time than the first algorithm. Moreover, it is known that a cloud computing system usually has a large number of nodes. We also propose node combination techniques to reduce the possibly large data replication time. Finally, simulation experiments are performed to demonstrate the effectiveness of the proposed algorithms in the data replication and recovery.

Index Terms—Cloud computing, data-intensive application, quality of service, data replication, network flow problem

1 Introduction

Cloud computing provides scalable computing and storage resources via the Internet [1], [2], [3]. It also enables users to access services without regard to where the services are provided and how they are delivered, similar to water, electricity, gas, and telephony utilities [4]. With the flexible and transparent features in the resource allocation and service provisioning, more and more data-intensive applications are developed in the cloud computing environment. The data-intensive applications devote most of their execution time in disk I/O for processing a large volume of data, for example, data mining of commercial transactions, satellite data processing, web search engine, and so on. Apache Hadoop [5] is an emerging cloud computing platform dedicated for data-intensive applications.

Due to a large number of nodes in the cloud computing system, the probability of hardware failures is nontrivial based on the statistical analysis of hardware failures in [6], [7], [8]. Some hardware failures will damage the disk data of nodes. As a result, the running data-intensive applications may not read data from disks successfully. To tolerate the data corruption, the data replication technique is extensively adopted in the cloud computing system to provide high data availability [9], [10], [11], [12]. However, the QoS requirement of an application is not taken into account in the data replication. When data corruption occurs, the QoS requirement of the application cannot be supported continuously. The reason is explained as follows. With a large number of nodes in the cloud computing system, it is difficult to ask all nodes with the same performance and capacity in their CPUs, memory, and disks [13]. For example, the Amazon EC2 is a realistic heterogeneous cloud platform, which provides various infrastructure resource types to meet different user needs in the computing and storage resources [14]. The cloud computing system has heterogeneous characteristics in nodes. Due to the node heterogeneity, the data of a high-QoS application may be replicated in a low-performance node (the node with slow communication and disk access latencies). Later, if data corruption occurs in the node running the high-QoS application, the data of the application will be retrieved from the low-performance node. Since the low-performance node has slow communication and disk access latencies, the QoS requirement of the high-QoS application may be violated. Note that the QoS requirement of an application is defined from the aspect of the request information. For example, in [15], the response time of a data object access is defined as the QoS requirement of an application in the content distribution system.

This paper investigates the *QoS-aware data replication* (QADR) problem for data-intensive applications in cloud computing systems. The QADR problem concerns how to efficiently consider the QoS requirements of applications in the data replication. The main goal of the QADR problem is to minimize the data replication cost and the number of QoS-violated data replicas. By minimizing the data replication cost, the data replication can be completed quickly. This can significantly reduce the probability that the data corruption occurs before completing data replication. Due to limited

- *J.-W. Lin is with the Department of Computer Science and Information Engineering, Fu Jen Catholic University, New Taipei City 24205, Taiwan. E-mail: jwlin@csie.fju.edu.tw.*
- *C.-H. Chen is with the Department of Electrical Engineering, National Taiwan University, Taipei 10617, Taiwan. E-mail: d01921025@ntu.edu.tw.*
- *J.M. Chang is with the Department of Electrical and Computer Engineering, Iowa State University, 391A Durham, 2215 Coover Hall, Ames, Iowa 50011-2252. E-mail: morris@iastate.edu.*

Manuscript received 15 Feb. 2013; revised 24 May 2013; accepted 23 June 2013; published online 16 July 2013.
Recommended for acceptance by P. Balaji.
For information on obtaining reprints of this article, please send e-mail to: tcc@computer.org, and reference IEEECS Log Number TCC-2013-02-0027.
Digital Object Identifier no. 10.1109/TCC.2013.1.

replication space of a storage node, the data replicas of some applications may be stored in lower-performance nodes. This will result in some data replicas that cannot meet the QoS requirements of their corresponding applications. These data replicas are called the QoS-violated data replicas. The number of QoS-violated data replicas is expected to be as small as possible.

To solve the QADR problem, we first propose a greedy algorithm, called the *high-QoS first-replication* (HQFR) algorithm. In this algorithm, if application i has a higher QoS requirement, it will take precedence over other applications to perform data replication. However, the HQFR algorithm cannot achieve the above minimum objective. Basically, the optimal solution of the QADR problem can be obtained by formulating the problem as an *integer linear programming* (ILP) formulation. However, the ILP formulation involves complicated computation. To find the optimal solution of the QADR problem in an efficient manner, we propose a new algorithm to solve the QADR problem. In this algorithm, the QADR problem is transformed to the *minimum-cost maximum-flow* (MCMF) problem. Then, an existing MCMF algorithm is utilized to optimally solve the QADR problem in polynomial time. Compared to the HQFR algorithm, the optimal algorithm takes more computational time. However, the two proposed replication algorithms run in polynomial time. Their time complexities are dependent on the number of nodes in the cloud computing system. To accommodate to a large-scale cloud computing system, the scalable replication issue is particularly considered in our QADR problem. We use node combination techniques to suppress the computational time of the QADR problem without linear growth as increasing the number of nodes.

To the best of our knowledge, this is the first paper to investigate the QADR problem in the cloud computing system. Overall, the main contributions of this paper are summarized as follows:

- Unlike previous data replication schemes of the cloud computing system, our data replication algorithms consider the QoS requirements of applications.
- For optimally solving the QADR problem, we present how to formulate the QADR problem as an ILP formulation. Considering the computational complexity in solving the ILP formulation, we transform the QADR problem to the MCMF problem to obtain the polynomial-time optimal solution.
- The proposed replication algorithms can accommodate to a large-scale cloud computing system. We utilize node combination techniques to suppress the computational time of the QADR problem.

The rest of the paper is organized as follows: Section 2 introduces the preliminaries of this paper. Section 3 presents our data replication algorithms. Section 4 evaluates the performance of the proposed algorithms. Finally, Section 5 concludes the paper.

2 Preliminaries

2.1 System Model

We refer to the architecture of the Hadoop distributed file system (HDFS) [10] to design our replication algorithms.

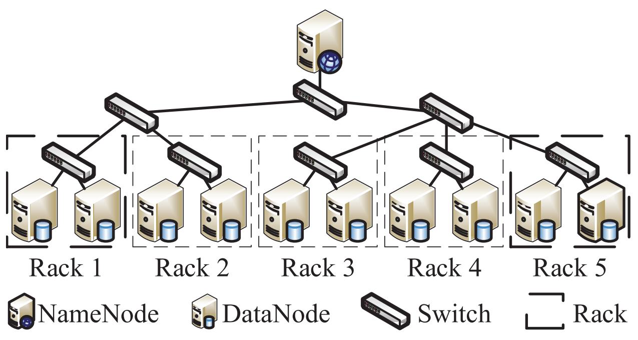


Fig. 1. The architecture of the Hadoop distributed file system.

The HDFS has many similarities with the proprietary distributed file system: Google file system [16]. It consists of a single *NameNode* and a set of *DataNodes*. The NameNode and DataNodes are deployed within a number of racks, as shown in Fig. 1, each of which has an associated rack number to represent in which rack it is located. The NameNode mainly manages the file system namespace and the locations of data blocks (the mapping of data blocks to DataNodes). A file is split into one or more data blocks. Then, these data blocks are dispersedly stored in DataNodes. In Hadoop, applications are executed in DataNodes. When the application would like to process a data block, it acts as an HDFS client to send a block read (write) request to the NameNode. The NameNode finds the corresponding DataNode to process this read (write) request. Each DataNode also periodically sends a heartbeat message to the NameNode to represent that it is functioning properly. In the HDFS architecture, there is also an Ethernet switch in each rack to provide the node communication within the rack (intrarack node communication). There are also some aggregated Ethernet switches for providing the node communication between racks (inter-rack node communication). For all the switches, a logical tree network topology is formed among them because switches usually adopt the common communication protocol: spanning tree protocol (STP) [17].

2.2 Related Work

To tolerate failures in cloud computing systems, the techniques of checkpoint and data replication have been extensively used.

The checkpoint technique is used to tolerate the Namenode failure. In [9], the NameNode periodically saves its file system namespace as a persistent record called a checkpoint. The checkpoint is stored in the disk of the NameNode. When the NameNode incurs a transient failure, the state of the file system namespace can be restored using the most recent checkpoint.

The data replication technique is used for the DataNode to protect its stored data blocks against failure. In HDFS [10], the default replica factor of a data block is 2. Whenever a data block is written to the DataNode i, the original copy of this data block is stored in the disk of the DataNode i. Two replicas of this data block are stored in two DataNodes whose rack numbers are different from that of the DataNode i. This replica placement manner particularly considers the possible power outage and switch failure in a rack. In addition to data replication, the work of [11] particularly discussed the problem of maintaining the consistency of data replicas in the cloud computing system. The lazy update method is used to separate the processes of data replica updates and data accesses, which can improve the throughput of data accesses and reduces response time. Unlike the typical three-replica replication strategy, the authors of [12] presented a cost-effective data reliability management mechanism. The mechanism is based on proactive replica checking to reduce the number of replicas stored while meeting the data reliability requirement. It can reduce the storage space consumption. From the above literature, we can know that the data replication issue has been investigated in cloud computing systems. However, the QoS requirement of the application is not be concerned in data replication.

The similar QADR problem has been discussed for content distribution systems [15]. In the work, the authors investigated how to place the replicas of each data object in appropriate servers of the system. With the data object replication, clients can access data objects from their proximity servers. The authors also proved that the concerned QADR problem is NP-complete. Two heuristic algorithms, called *l-Greedy-Insert* and *l-Greedy-Delete*, were proposed to gradually insert and remove the replicas of each data object without violating the specified QoS requirement, respectively. However, the proposed heuristic algorithms in [15] take long execution time. To reduce the execution time, the authors of [18] presented a new heuristic QADR algorithm that is based on the idea of cover set. The cover set of a server u is the set of servers that can serve the requests from u within $q(u)$, where $q(u)$ is the required service quality requirement. By finding the cover set of each server, the good QADR solution can be obtained.

The QADR problem has also been studied on data grid systems [19], [20], [21]. The authors of [19] formulated the QADR problem as a dynamical programming problem. Then, a distributed replica placement algorithm was proposed, which exploited the historical access records of popular data files to compute replication locations designed to satisfy QoS requirements. The proposed replication algorithm is based on a fully distributed dynamic programming technique to avoid the limitations of the centralized algorithm (e.g., reliability and performance bottleneck). The authors of [20] addressed the replica placement problem in tree-based mobile grid environments to meet the QoS requirements of mobile users and load balancing of replicas. The QoS requirements are specified by the number of hops from the mobile user to the server with replica. In addition to increasing data availability, the addressed replica problem can improve data access performance. To efficiently solve the replica problem, the authors proposed a two-step solution that applies a dynamic programming approach and a binary search algorithm. The authors of [21] presented a dynamic replica replacement strategy, called least value replacement (LVR), which can satisfy QoS requirements and storage capacity constraints in a data grid environment. The main feature of LVR is that it can ascertain the importance of replicas in a grid site. Whenever a grid site is full without available storage space, the LVR can automatically decide on which replica to be replaced based on information such as access frequency and files future value.

The above replication algorithms of [15], [18], [19], [20], [21] are unsuitable for solving our QADR problem. Our QADR problem is discussed under a cloud computing system. The number of replicas for a data block is fixed. Based on the fixed replica factor, our QADR problem attempts to minimize the data replication cost. Due to the limited replication space of each node, our QADR problem particularly considers the replication contention among data blocks. The replication contention may cause some data replicas that cannot meet the QoS requirements of their corresponding applications. In such a case, a number of QoS-violated data replicas are generated. Our QADR problem also concerns how to minimize the number of QoS-violated data replicas. In [15], [18], [19], [20], [21], the problem of QoS violation minimization was not discussed in data replication. For a server with limited replication space, if there are many data object replicas to be placed in this server, the replicas of some data objects cannot be stored successfully. In such a case, the unsuccessful data object replicas will be put in other servers without QoS satisfaction. This problem is not handled in the above previous work.

3 QoS-Aware Replication Algorithms

In this section, we will present two replication algorithms for solving the QADR problem in the cloud computing system. Before elaborating the proposed algorithms, we give some definitions for clarifying the QADR problem.

Definition 1. *Given a cloud computing system with a set of storage nodes S, these storage nodes can also run applications in addition to storing data. The storage node functionality is similar to the storage node in HDFS [10].*

Definition 2. *For a storage node $r \in S$, if its running application writes a data block b to the disk of r, a replication request will be issued from r to replicate a number copies of b to the disks of other nodes. In the cloud computing system, $|S|$ is usually large. It is very possible that there may have many concurrent replication requests issued from different nodes at a certain time instant. Due to space limitation, each node cannot store too many data replicas from other nodes.*

Definition 3. *For a data block b, if it is replicated from node r to node q, one data replica dr of b will be stored at q. A desired access time T is specified for dr. In addition, dr is also associated with a replication cost RC and an access time AC.*

Definition 4. *When the original copy of b cannot be read due to data corruption, r attempts to retrieve the data replica dr from q. If AC is greater than T, dr is one QoS-violated data replica.*

Definition 5 (The QoS-aware data replication problem). *The main objective of the QADR problem is to find an optimal replica placement strategy P, which can minimize both the total replication cost of all data blocks and the total number of QoS-violated data replicas.*

3.1 Intuitive Method

3.1.1 Basic Idea and Assumptions

In Section 1, we have stated that a high-QoS application has the stricter requirement in the response time of a data access than a normal application. The high-QoS application should take precedence over the low application to put its data replicas on the high-performance storage nodes. By sorting the QoS requirements of applications, if there is limited replication space in the high-performance storage node, it can first store the data replicas of high-QoS applications. Thereafter, when the high-QoS application reads a corrupt data replica, its QoS requirement can be continuously supported by retrieving the data replica from a high-performance node. Before describing the details of the HQFR algorithm, we make the following assumptions:

- For convenience, it is assumed that a node runs one application during a time interval. Only one file is opened in the execution of the application. In Hadoop, the file is divided into a number of data blocks, each of which contains 64 MBytes. The data block is a data access (replication) unit.
- As mentioned in Section 2.2, each data block in HDFS has two default data replicas against data corruption. In addition to the node failure, the switch failure is additionally concerned. The switch failure may cause that the data replicas in a rack cannot be accessed by the nodes in other racks. The switch failure is also regarded as the rack failure. For considering the possible rack failure, the two replicas of a data block and its original copy cannot be all put in a single rack. According to this data availability requirement, our proposed replication algorithms also make each data block with two replicas except the original copy. These two replicas can be placed in the same rack or different racks, but their associated rack numbers are different from that of the original copy.
- Like [15], the QoS requirement of an application is defined from the aspect of the request information, such as the response time of a data object access. In [15], it was explicitly stated that the data response time can be specified in the form of service-level agreements (SLAs). In this paper, the proposed replication algorithms are mainly designed for data-intensive applications. The main characteristic of a data-intensive application is that it performs most operations on the disk-resident data. In such a case, the QoS requirement of a data-intensive application can be defined as the access time to retrieve a data block. The expected data access time can also be specified in the SLA.

3.1.2 High-QoS First-Replication Algorithm

Table 1 lists the notations used in our replication algorithms. The HQFR algorithm is given in Fig. 2, whose operations are elaborated as follows.

When an application would like to write a data block, the node executing the application would issue a replication request for the data block. The information about the QoS requirement of the application (the desired access time of the data block) is also attached on the replication request to generate a QoS-aware replication request.

TABLE 1
Summary of Notations

Basic Notation	Description
S	A set of nodes in a cloud computing system.
$\|S\|$	The number of nodes in a cloud computing system.
S_r	A set of requested nodes which issued replication requests concurrently.
$\|S_r\|$	The total number of requested nodes in S_r.
r_i	A requested node in S_r.
${S_n}^{r_i}$	A set of nodes that store a data block replica sent from r_i.
${S_q}^{r_i}$	A set of qualified nodes corresponding to the requested node r_i.
${S_{uq}}^{r_i}$	A set of un-qualified nodes corresponding to the requested node r_i.
$R(r_i)$	A function determines the associated rack number of the node r_i.
${S_n}^{\overline{R(r_i)}}$	A set of nodes which rack numbers are different with that of r_i.
q_j	A node in ${S_n}^{\overline{R(r_i)}}$.
r_f	A given replication factor.
$a(q_j)$	The available replication space for qualified node q_j in terms of the amount of block space.
$T_{access}(r_i, q_j)$	The replica access time from q_j to r_i.
$T_{storage}(r_i, q_j)$	The storage time to store one data replica from r_i to q_j.
Additional Notation for ILP	**Description**
$x(r_i, q_j)$	The {0,1} variable indicates whether a data replica is placed at node q_j from node r_i.
$y(r_i, q_j)$	The {0,1} variable indicates whether a QoS-violated data replica is placed at node q_j from node r_i.
Additional Notation for MCMF	**Description**
$s(s')$	The source node on a flow graph (a reduced flow graph).
$t(t')$	The sink node on a flow graph (a reduced flow graph).
rr_m	A requested rack node on a reduced flow graph.
S_rr_m	A set of requested nodes that are combined as rr_m.
qr_n	A qualified rack node on a reduced flow graph.
S_qr_n	A set of qualified nodes that are combined as qr_n.

Multiple QoS-aware replication requests may be issued concurrently from a number of nodes. These concurrent replication requests will be processed in a sequence based on the ascending order of their associated access time. If the replication request i has a higher QoS requirement than the replication request j, the replication request i is associated with a smaller access time than the replication request j. In such a case, the HQFR algorithm will first process the replication request i to store its corresponding data replicas.

When processing a QoS-aware replication request from the requested node r_i, it is required to find the

```
Input: A set of requested nodes $S_r$.
Output: QoS-aware data replica placement.
1: Sort the requested nodes in $S_r$ based on their associated access time.
2: for each requested node $r_i$ in the sorted list of the requested nodes do
3:     $S_q{}^{r_i}$ ← Find the corresponding qualified nodes of $r_i$ using Eq. (1)
       and (2) to verify all storage nodes.
4:     Select the $r_f$ qualified nodes from $S_q{}^{r_i}$ which have smaller data
       replica access time than other qualified nodes in $S_q{}^{r_i}$.
5:     for each selected qualified node $q_j$ do
6:         Store one data replica from $r_i$ to $q_j$.
7:         Update the available replication space of $q_j$.
8:     end for
9:     if $|S_q{}^{r_i}| < r_f$ then
10:        $S_{uq}{}^{r_i}$ ← Find the corresponding un-qualified nodes of $r_i$ only
           using Eq. (1).
11:        Select the $r_f - |S_q{}^{r_i}|$ un-qualified nodes from $S_{uq}{}^{r_i}$ which have
           smaller data replica access time than other qualified nodes in $S_{uq}{}^{r_i}$.
12:        for each selected un-qualified node $uq_j$ do
13:            Store one data replica from $r_i$ to $uq_j$.
14:            Update the available replication space of $uq_j$.
15:        end for
16:    end if
17: end for
```

Fig. 2. The high-QoS first-replication algorithm.

correspondingly qualified nodes that satisfy the QoS requirement of the running application in r_i. It has been known that the access time of a data block is used to represent the QoS requirement of a data-intensive application. Assumed that the QoS requirement of the running application in r_i is $T_{qos}(r_i)$ time units. If the node q_j would like to be one qualified node of r_i, it needs to meet the following two conditions:

- The nodes q_j and r_i cannot be located within the same rack. This condition is for considering the possible rack failure (see Section 2.2)

$$R(r_i) \neq R(q_j), \tag{1}$$

  where R is the function to determine in which rack a node is located (see Table 1).
- The data replica access time from q_j to r_i ($T_{access}(r_i, q_j)$) needs to meet the $T_{qos}(r_i)$ constraint:

$$T_{access}(r_i, q_j) = T_{disk}(q_j) + T_{comm}(r_i, q_j) \leq T_{qos}(r_i), \tag{2}$$

  where $T_{disk}(q_j)$ is the disk access latency for retrieving a data block replica from the disk of q_j, and $T_{comm}(r_i, q_j)$ is the network communication latency for transmitting a data block replica from q_j to r_i.

Based on (2), we know that the data replica access time consists of the disk access latency and network communication latency. In addition to the disk access time and transmission time, the two latencies are dependent on the disk workload and the network traffic load. These two load parameters are dynamical, which are difficult to be estimated precisely. The authors of [15] explicitly stated that the server load and network load are not considered in the estimation of the data replica access time because the hardware resource addition can sufficiently provide the capacity requirement of load. For involving the disk workload and network load in the estimation of data replica access time, a queuing model can be used to estimate the two load parameters. Several queuing models have been developed [22], which are also extensively used to evaluate the performance of a computer system. For example, in [23], the queuing model $M/G/m/m+r$ is used to evaluate the performance of the cloud computing system. Basically, the estimation of the data replica access time is not the main issue of this paper. We focus on developing QADR algorithms based on different data replica access times between storage nodes.

According to the above two conditions, all the qualified nodes with respect to the requested node r_i can be found. Then, r_f qualified nodes are selected from all the qualified nodes. These r_f qualified nodes have smaller data replica access time than other qualified nodes. Next, the data block of r_i will be, respectively, made one replica to be stored in each of r_f qualified nodes. These r_f qualified nodes will also update their, respectively, available replication space.

From the above-given operations, the replication cost of the HQFR algorithm can be represented as the total storage cost (time) taken by all the requested nodes to store their respective data block replicas. Like [15] and [18], the replication cost is represented as the sum of the storage costs of all data block replicas, as follows:

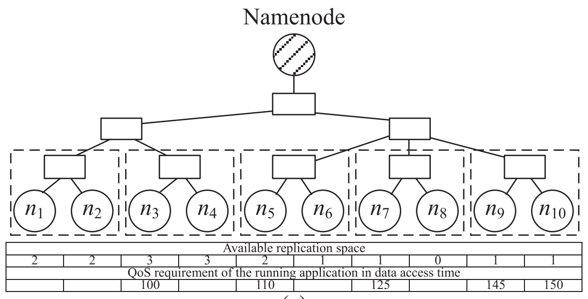
$$\sum_{\forall r_i \in S_r} \sum_{\forall q_j \in S_n{}^{r_i}} T_{storage}(r_i, q_j). \tag{3}$$

$T_{storage}(r_i, q_j)$ is similar to $T_{access}(r_i, q_j)$. In (2), we have clearly defined $T_{access}(r_i, q_j)$ to be the sum of the network communication latency and the disk access latency for retrieving a data block replica from node q_j to node r_i. Therefore, $T_{storage}(r_i, q_j)$ includes the time to transmit a data block replica from r_i to node q_j and the time to write the data block replica to the disk of q_j.

Intuitively, the requested nodes can concurrently store their data block replicas. However, two or more requested nodes may contend to place their data block replicas at the same qualified node with a limited amount of replication space. In such a case, a placement sequence is required to be made for determining what requested nodes can place their data block replicas in the qualified nodes, which is given in Fig. 2.

3.1.3 Time Complexity

Next, we analyze the time complexity of the HQFR algorithm. The HQFR algorithm can be divided into two parts: the arrangement of the replication request sequence and the execution of each replication request. In the first part, if the QoS requirement of a requested node is high, it will be associated with a small access time to replicate its corresponding data block. Its replication request is also performed first and then that of the request node with the lower QoS requirement. The first part follows the sorting order of the associated access time to arrange the replication sequence of request nodes (see line 1 of Fig. 2). It is known that the time complexity of the sorting problem is $O(n \log n)$. If there are $|S_r|$ requested nodes, the first part will take $O(|S_r| \log |S_r|)$. By following the given replication request sequence, the second part alternatively performs the QoS-aware replication request of each requested node. In this part, it first takes $O(|S|)$ to find the correspondingly qualified nodes of a requested node. Then, it selects r_f best qualified nodes to store the data block replica. For all $|S_r|$



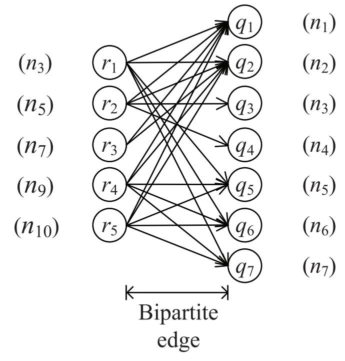
Available replication space									
2	2	3	3	2	1	1	0	1	1
QoS requirement of the running application in data access time									
		100		110		125		145	150

(a)

Original node	Original Destination node								
	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
n_1	5	20	55	60	35	40	85	90	135	140
n_2	15	10	55	60	35	40	85	90	135	140
n_3	15	20	45	60	35	40	85	90	135	140
n_4	15	20	55	50	35	40	85	90	135	140
n_5	15	20	55	60	50	65	110	115	135	140
n_6	15	20	55	60	60	55	110	115	135	140
n_7	105	110	165	170	130	135	90	115	135	140
n_8	105	110	165	170	130	135	110	95	135	140
n_9	105	110	165	170	130	135	110	115	95	140
n_{10}	105	110	165	170	130	135	110	115	135	100

(b)

Fig. 3. An example to demonstrate a QADR problem. (a) A small-scale cloud system example. (b) The replica access time between any two nodes in the cloud system example.



(a)

Bipartite edge	(r_1, q_1)	(r_1, q_2)	(r_1, q_5)	(r_1, q_6)	(r_1, q_7)
	15	20	35	40	85
	(r_2, q_1)	(r_2, q_2)	(r_2, q_3)	(r_2, q_4)	
	15	20	55	60	
	(r_3, q_1)	(r_3, q_2)			
	105	110			
	(r_4, q_1)	(r_4, q_2)	(r_4, q_5)	(r_4, q_6)	(r_4, q_7)
	105	110	130	135	110
	(r_5, q_1)	(r_5, q_2)	(r_5, q_5)	(r_5, q_6)	(r_5, q_7)
	105	110	130	135	110

(b)

Fig. 4. The weighted bipartite graph corresponding to a QADR problem. (a) The constructed bipartite graph. (b) The weight of each bipartite edge.

requested nodes, their replication requests can be done in $O(|S_r||S|)$. Overall, the time complexity of the entire HQFR algorithm is $O(|S_r|\log|S_r|) + O(|S_r||S|)$. If $|S_r|$ is close to $|S|$, this time complexity will be $O(|S|^2)$.

Although the HQFR replication algorithm can easily solve the QADR problem in polynomial time, it may not achieve the two minimum objectives of the QADR problem. An example is given in Fig. 3. For convenience, Fig. 3 considers the cloud computing system with 10 datanodes located within five racks. Each rack has two datanodes. At a time instant, five concurrent QoS-aware replication requests are concurrently issued from requested nodes n_3, n_5, n_7, n_9, and n_{10}. Based on the HQFR algorithm, the replication request sequence is arranged as n_3, n_5, n_7, n_9, and n_{10}.

Then, we model the relationship between the requested nodes and their correspondingly qualified nodes as a weighted bipartite graph, as shown in Fig. 4. In this figure the weighed value on a bipartite edge denotes the data replica storage cost of a requested-qualified node pair. The five requested nodes r_1, r_2, r_3, r_4, and r_5 correspond to the datanodes n_3, n_5, n_7, n_9, and n_{10} in Fig. 3, respectively. For the seven qualified nodes q_1, q_2, q_3, q_4, q_5, q_6, and q_7 they correspond to the datanodes n_1, n_2, n_3, n_4, n_5, n_6, and n_7 in Fig. 3, respectively. Based on (1) and (2), the qualified nodes of the requested node r_1 are the nodes q_1, q_2, q_5, q_6, and q_7. In Fig. 4, the requested node r_1 has links with these qualified nodes. According to the given replication request sequence of the HQFR algorithm (r_1, r_2, r_3, r_4, r_5), the requested nodes r_1 and r_2 are prior to the requested node r_3 to perform the QoS-aware replication requests. From Fig. 4, we can also see that q_1 and q_2 are the qualified nodes of all the requested nodes. Since the replication requests of the requested nodes r_1 and r_2 are first executed, the replication spaces of q_1 and q_2 are fully occupied by the requested nodes r_1 and r_2. This will cause that the requested nodes r_3 and r_5 cannot find a sufficient number of qualified nodes (r_f qualified nodes) to store the replicas of their data blocks. The QoS-aware replication requests of r_3 and r_5 cannot be performed successfully. Note that r_f in Fig. 4 is set to 2.

However, if the replication request sequence is (r_3, r_4, r_1, r_2, r_5) rather than (r_1, r_2, r_3, r_4, r_5), all the requested nodes can perform their QoS-aware replication requests successfully. In such a situation, the minimum replication cost is 860. From the weighted bipartite graph of Fig. 4, we can see that the requested nodes r_1 and r_2 can place their data block replicas in the qualified nodes q_5 and q_6, as well as the qualified nodes q_3 and q_4, respectively. In such replica placement, the requested nodes r_3 and r_5 can successfully perform their QoS-aware data replication by placing their data block replicas in the qualified nodes q_1 and q_2, as well as the qualified nodes q_5 and q_7, respectively.

From the above violation example, we know that the HQFR algorithm can easily cause the requested nodes to contend some qualified nodes. If these qualified nodes have not enough storage space for data block replicas, the minimum objectives of the QADR problem will not be achieved. In a cloud computing system, many applications may simultaneously execute. As a result, there are many requested nodes to issue their replication requests concurrently. The above replication contention case easily occurs. Instead of the HQFR algorithm, we will propose a new replication algorithm to ensure the two minimum objectives of the QADR problem.

3.2 Optimal Replica Placement

3.2.1 ILP Formulation

The optimal solution of the QADR problem can be obtained using *integer linear programming*. The ILP is a well-known technique used to solve the optimal problems with the following characteristics: a linear objective function, a number of linear constraints, and an integer solution set [24]. Given an instance P of the QADR problem, the corresponding ILP formulation can be expressed as (4) to (9). The used notations can be found in Table 1. In the given ILP formulation, the data replica placement can be obtained based on the binary variables x and y. If $x(r_i, q_j)$ is 1, the node q_j stores one data block replica of the requested node r_i. If $y(r_i, q_j)$ is also 1, the corresponding data block replica is one QoS-violated replica.

$$\text{Minimize}\left(\sum_{\forall r_i \in S_r} \sum_{\forall q_j \in \overline{S_n^{R(r_i)}}} x(r_i, q_j) \times T_{storage}(r_i, q_j)\right) + \left(\sum_{\forall r_i \in S_r} \sum_{\forall q_j \in \overline{S_n^{R(r_i)}}} y(r_i, q_j) \times k\right), \tag{4}$$

$$\text{subject to } \forall r_i \in S_r, \sum_{\forall q_j \in \overline{S_n^{R(r_i)}}} x(r_i, q_j) = r_f, \tag{5}$$

$$\forall q_j \in S, \sum_{\forall r_i \in S_r} x(r_i, q_j) \le a(q_j), \tag{6}$$

$$\forall r_i \in S_r \wedge \forall q_j \in \overline{S_q^{r_i}}, y(r_i, q_j) = x(r_i, q_j), \tag{7}$$

$$\forall r_i \in S_r \wedge \forall q_j \in S_q^{r_i}, y(r_i, q_j) = 0, \tag{8}$$

$$\forall r_i \in S_r \wedge \forall q_j \in \overline{S_n^{R(r_i)}}, x(r_i, q_j), y(r_i, q_j) \in \{0, 1\}. \tag{9}$$

Based on (4), there are two minimum terms in the objective function of the ILP formulation. The first minimum term is the total replication cost of all data replicas, and the second minimum term is the number of QoS-violated data replicas. The second minimum term is prior to the first minimum term. The coefficient k is used to ensure that the number of QoS-violated data replicas will be first minimized. The reason will be explained later. In the ILP formulation, if the requested node r_i puts one data block replica in the node q_j, this event is recorded by setting 1 in $x(r_i, q_j)$. However, if the replica is a QoS-violated data block replica, $y(r_i, q_j)$ will also be set as 1. By adding up all the values of y, the total number of QoS-violated data replicas can be obtained. This number is expected to be as small as possible by associating with a constant coefficient $k = \max_{\forall r_i \in S_r \wedge \forall q_j \in S}\{T_{storage}(r_i, q_j)\} + 1$. With the setting of k, each $y(r_i, q_j)$ has a larger coefficient than each $x(r_i, q_j)$. It is also known that the values of $x(r_i, q_j)$ and $y(r_i, q_j)$ are either 0 or 1. Under the minimization requirement of (4), the given ILP formulation avoids setting 1 to each $y(r_i, q_j)$. Note that the value of each $y(r_i, q_j)$ is also dependent on (7) and (8) in addition to (4). From the above description, we can know that the ILP formulation will first minimize the number of QoS-violated data replicas. Then, it minimizes the total replication cost of all data replicas.

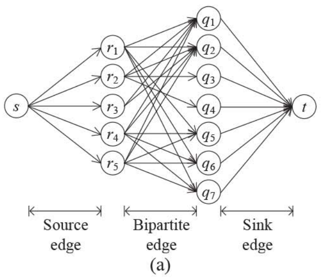
Based on the given replication factor r_f, each requested node stores r_f data block replicas in r_f storage nodes, respectively. These storage nodes cannot have the same rack number as the requested node to meet the rack constraint mentioned in Section 3.1. (5) expresses the possible storage nodes of a requested node. (6) denotes that each node cannot store too many data block replicas to exceed the capacity of its replication space. Due to the replication contention and limited replication space, the requested node r_i may put one data block replica in a nonqualified node q_j. In such a case, the generated data replica in node q_j is a QoS-violated data replica. In addition to $x(r_i, q_j)$, this QoS-violated replication event is particularly recorded in $y(r_i, q_j)$ (see (7)). (8) is used to represent that it is impossible for the requested node r_i to generate a QoS-violated data replica in each of its qualified nodes because these qualified nodes can meet the QoS requirement. Therefore, each corresponding $y(r_i, q_j)$ is set to 0. Based on (7) and (8), the total number of QoS-violated data replicas can be easily counted by finding the value of each $y(r_i, q_j)$ with 1. The final constraint in (9) is given for enhancing data availability. A node can only store at most one data replica from a requested node.

In the above ILP formation, there are $|S_r| \times |S|$ binary variables in x and y. In the cloud computing system, the number of nodes $|S|$ is usually large. Solving ILP is well known to be time consuming [25]. If $|S_r|$ or $|S|$ is large, the above ILP formulation will take much computational time to obtain the optimal solution of the QADR problem.

3.2.2 Optimal Solution with Efficient Computational Time

The optimal solution of the QADR problem can be obtained in a more efficient manner. Instead of mapping to an ILP formulation, we transform the QADR problem to the minimum-cost maximum-flow problem. The problem transformation is also beneficial in minimizing the replication cost of QoS-violated data replicas. The MCMF problem is a variation of the well-known minimum-cost flow problem, which is defined as follows: Given a network flow graph G with two special nodes, source node s and sink node t, what is the minimum total transmission cost to send an amount of flow f from s to t as much as possible? Each edge (i, j) on G has an associated cost $c(i, j)$ and an attached capacity $u(i, j)$. The associated cost indicates the required cost for sending one flow unit via the edge (i, j). The attached capacity denotes the maximum amount of flow that can be transmitted via the edge. If a subflow f_s is transmitted via edge (i, j), the amount of this subflow $f_s(i, j)$ must be less than or equal to $u(i, j) (f_s(i, j) \le u(i, j))$. The transmission cost of the subflow f_s on edge (i, j) is $f_s(i, j) \times c(i, j)$.

There have been several polynomial-time MCMF algorithms [26], [27], [28]. After transforming the QADR problem to the MCMF problem, one of the existing MCMF algorithms can be applied to obtain the optimal solution of the QADR problem in polynomial time.



Source edge	(s, r_1)	(s, r_2)	(s, r_3)	(s, r_4)	(s, r_5)
	(2, 1)	(2, 1)	(2, 1)	(2, 1)	(2, 1)
Bipartite edge	(r_1, q_1)	(r_1, q_2)	(r_1, q_5)	(r_1, q_6)	(r_1, q_7)
	(1, 15)	(1, 20)	(1, 35)	(1, 40)	(1, 85)
	(r_2, q_1)	(r_2, q_2)	(r_2, q_3)	(r_2, q_4)	
	(1, 15)	(1, 20)	(1, 55)	(1, 60)	
	(r_3, q_1)	(r_3, q_2)			
	(1, 105)	(1, 110)			
	(r_4, q_1)	(r_4, q_2)	(r_4, q_5)	(r_4, q_6)	(r_4, q_7)
	(1, 105)	(1, 110)	(1, 130)	(1, 135)	(1, 110)
	(r_5, q_1)	(r_5, q_2)	(r_5, q_5)	(r_5, q_6)	(r_5, q_7)
	(1, 105)	(1, 110)	(1, 130)	(1, 135)	(1, 110)
Sink edge	(q_1, t)	(q_2, t)	(q_3, t)	(q_4, t)	(q_5, t)
	(2, 1)	(2, 1)	(3, 1)	(3, 1)	(2, 1)
	(q_6, t)	(q_7, t)			
	(1, 1)	(1, 1)			

(b)

Fig. 5. The flow graph corresponding to a QADR problem. (a) The constructed flow graph. (b) The pair (capacity, cost) on each flow edge.

Theorem 1. *For the QADR problem, its two minimum objectives (the minimum total replication cost and the minimum number of QoS-violated data replicas) can be achieved in polynomial time by transforming it to the MCMF problem.*

Proof. In the following proof, we first demonstrate that the QADR problem can be reduced (transformed) to the MCMF problem in polynomial time. Then, we show that the solution of the MCMF problem can be used to represent the optimal solution of the QADR problem. The notations used in this proof can be referred to in Table 1. Let P and Q be the instances of the QADR and MCMF problems, respectively. In the QADR problem, the replication relationship between requested nodes and qualified nodes can be modeled as a weighted bipartite graph (see Fig. 4). For instance P, it has a correspondingly weighted bipartite graph (WBG_P).

Based on WBG_P, we further construct a network flow graph (NFG_Q) to represent instance Q. A source node s and a sink node t are added on two sides of WBG_P to connect with each of its requested nodes and each of its qualified nodes, respectively. The edges on NFG_Q can be divided into three types: source edges, bipartite edges, and sink edges. The source edges connect source node s with each requested node. The bipartite edges originally exist on WBG_P that connect requested nodes and qualified nodes. For the sink edges, they connect each qualified node with the sink node. Finally, a flow f is given, which will be transmitted from s to t of NFG_Q. By setting appropriate (capacity, cost) values on the edges of NFG_Q and the amount of flow on f, the instance Q can be a graphic representation as NFG_Q. The value settings will be discussed later.

By adding a source node and a sink node, the graphic representation of the instances $P(WBG_P)$ can be extended to that of the instance $Q(NFG_Q)$ in polynomial time. An example is given in Fig. 5 that extends the WBG_P of Fig. 4. The graph extension implies that instance P can be reduced (transformed) to instance Q in polynomial time. Next, we need to show that the optimal solution of instance P can be derived by solving the MCMF solution of instance Q.

If the replication factor of a data block is r_f, each requested node needs to place r_f data replicas at the disks of r_f storage nodes. In instance P, one data replica is corresponding to one flow unit in instance Q. To let the source node transmit r_f flow units to each requested node, the (capacity, cost) on each source edge is set to $(r_f, 1)$. The amount of flow f entering the source node is set to $\sum_{i=1}^{|sr|} r_f = |S_r| \times r_f$. After receiving r_f flow units, a requested node selects r_f qualified nodes with smaller transmission costs than others. Then, the requested node distributes the received flow to these qualified nodes. A qualified node can only receive one flow unit from a requested node. The (capacity, cost) on a bipartite edge (r_i, q_j) is set to $(1, T_{storage}(r_i, q_j))$, where $T_{storage}(r_i, q_j)$ is the original weight value on the bipartite edge (r_i, q_j) of WBG_P. This weight value can represent the transmission cost of one flow unit on the edge (r_i, q_j) of NFG_Q. To consider the amount of available replication space in a qualified node, the (capacity, cost) on the sink edge (q_j, t) is set to $(a(q_j), 1)$. With this capacity setting, even if a qualified node connects with many requested nodes, the total amount of flow entering this qualified node is not larger than $a(q_j)$.

According to NFG_Q, if one or more flow units cannot be successfully transmitted from s to t, this represents that some requested nodes cannot put their data block replicas in appropriately qualified nodes. In such a case, these data replicas will be stored in unqualified (QoS-violated) nodes, called QoS-violated data replicas. It has been known that the MCMF solution of NFG_Q can transmit maximum amount of flow from s to t. Therefore, the amount of unsuccessful flow from s to t can be minimized. From the perspective of the instance P, the number of QoS-violated data replicas can be minimized. In addition to maximizing the amount of transmitted flow, the MCMF solution can also give the minimum total transmission cost. By following the MCMF solution of NFG_Q to place data replicas of instance P, the minimum total replication cost of instance P can be obtained.

From the above description, we can obviously know that the two minimum objectives of the QADR problem can be achieved in polynomial time by transforming the QADR problem to the MCMF problem. Theorem 1 is proven. □

Based on Theorem 1, we can design the optimal QADR algorithm with polynomial time by observing the amount of flow leaving each requested node. As shown in lines 10-22 of Fig. 6, if the requested node r_i cannot completely send out its incoming flow, it will have QoS-violated data replicas. In such a case, the unqualified nodes with respect

```
Input: A set of requested nodes $S_r$.
Output: Optimal placement for the QoS-satisfied and QoS-violated data
    replicas.
 1: $Sq \leftarrow \emptyset$.
 2: for each requested node $r_i$ in $S_r$ do
 3:     $Sq^{r_i} \leftarrow$ Find the correspondingly qualified nodes of $r_i$.
 4:     $Sq \leftarrow Sq \cup Sq^{r_i}$.
 5: end for
 6: Use $S_r$ and $Sq$ to model a network flow graph.
 7: Set appropriate (capacity, cost) values on the edges of the network flow
    graph.
 8: Apply an existing polynomial-time MCMF algorithm to obtain the MCMF
    solution of the network flow graph.
 9: Obtain the optimal placement for the QoS-satisfied data replicas from the
    MCMF solution.
10: $S_{ur} \leftarrow \emptyset$ and $S_{uq} \leftarrow \emptyset$.
11: for each requested node $r_i$ in $S_r$ do
12:     $f_{leaving} \leftarrow$ the amount of flow leaving from $r_i$.
13:     if $f_{leaving} < r_f$ then
14:         $S_{ur} \leftarrow S_{ur} \cup r_i$.
15:         $\overline{S_q^{r_i}} \leftarrow S - S_q^{r_i}$. /* The un-qualified nodes for $r_i$*/
16:         $S_{uq} \leftarrow S_{uq} \cup \overline{S_q^{r_i}}$.
17:     end if
18: end for
19: Use $S_{ur}$ and $S_{uq}$ to model a new network flow graph.
20: Set appropriate (capacity, cost) values on the edges of the new network
    flow graph.
21: Apply the MCMF algorithm on the new network flow graph to obtain
    the MCMF solution.
22: Follow the MCMF solution to make the optimal placement of the QoS-
    violated data replicas.
```

Fig. 6. The optimal replica placement algorithm.

to the requested node r_i are required to be found for storing the QoS-violated data replicas of r_i. Then, the requested node r_i and its correspondingly unqualified nodes are collected in S_{ur} and S_{uq}, respectively. To also minimize the total QoS-violated replication cost of all requested nodes, the MCMF problem transformation is performed again by modeling a new network flow graph based on the sets S_{ur} and S_{uq}. The optimal placement of all QoS-violated data replicas can be made by following the obtained MCMF solution (see lines 19-22 of Fig. 6).

3.3 Scalable Replication Issue

The computational complexity of the QADR problem is strongly dependent on how many requested and qualified nodes are involved in solving the problem. The numbers of requested and qualified nodes ($|S_r|$ and $|S_q|$) are related to the number of nodes ($|S|$) in a cloud computing system. In the cloud computing system, $|S|$ is usually large. Moreover, if most of nodes concurrently execute the applications with low-QoS requirements, $|S_r|$ and $|S_q|$ will be very close to $|S|$. Due to possibly large values for $|S_r|$ and $|S_q|$, the scalable replication issue needs to be concerned in solving the QADR problem. If not, large computational time will be incurred.

For avoiding large computational time, we utilize the rack-based combination and equivalent state combination to combine appropriately requested and qualified nodes in the original network flow graph. By performing the above two node combination techniques, the network flow graph can be reduced without taking large computational time in solving the corresponding QADR problem.

The basic idea of the rack-based combination is as follows: As stated in Section 2.1, nodes are deployed within a number of racks in a cloud computing system. The group-based management architecture is adopted by some cloud management literature [29], [30], [31]. The nodes are classified into a number of groups (racks). In a group, if all nodes have similar CPU and disk performance, there are few performance categories after classifying the performance capabilities of nodes within a group. This also means that the group capability can be easily characterized. In such a case, the management system can quickly and systemically find the group with enough capability to handle a service request. Next, one of the nodes in the group is designated to handle such service request. With the above management manner, the nodes with similar CPU and disk performance are usually deployed in the same rack. These nodes are also connected with each other using similar bandwidth via the switch of the rack. Based on this node deployment manner, if two or more requested nodes are located in the same rack, they are suitable to be combined as a requested rack node. Similarly, the qualified nodes in the same rack are also combined as a qualified rack node. In addition to combining requested and qualified nodes based on the rack unit, the following equations are used to model the relationships between the requested rack nodes and the qualified rack nodes on the reduced flow graph. For a requested rack node rr_m, the total number of replication requests is

$$|S\_rr_m|, \tag{10}$$

where S_rr_m is the set of requested nodes in rr_m (see Table 1). Each requested node issues one replication request at each time. The cardinality of S_rr_m can be used to represent the total number of replication requests in rr_m.

The QoS requirement of rr_m is

$$\min_{\forall r_i \in S\_rr_m} T_{qos}(r_i), \tag{11}$$

where $T_{qos}(r_i)$ has been known to be the QoS requirement of the request node r_i. For all the requested nodes in rr_m, they may have different QoS requirements. It has been mentioned that the QoS requirement is represented as the desired access time of a data block (see Section 3.1). In (11), the smallest access time is used to represent the QoS requirement of rr_m. This is for conveniently finding the correspondingly qualified nodes of rr_m. If the qualified node q_j can meet the QoS requirement of rr_m, it must satisfy the QoS requirements of all requested nodes in rr_m.

For a qualified rack node qr_n, the size of its replication space is the sum of the replication space of each qualified node in qr_n, as follows:

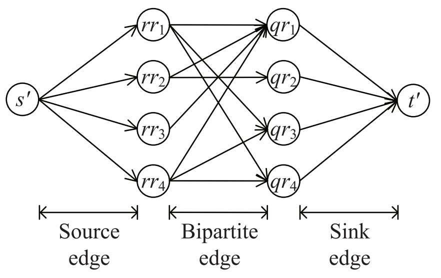
$$\sum_{\forall q_j \in S\_qr_n} a(q_j), \tag{12}$$

where S_qr_n and $a(q_j)$ are known to be the set of qualified nodes in qr_n and the available replication space of a qualified node q_j, respectively (see Table 1).

The storage cost of qr_n with respect to rr_m is

$$\max_{\forall r_i \in S\_rr_m \wedge \forall q_j \in S\_qr_n} T_{storage}(r_i, q_j), \tag{13}$$

where $T_{storage}(r_i, q_j)$ has been defined to be the storage cost to store one data replica from r_i to q_j. There are different requested nodes and qualified nodes in rr_m and qr_n,



Node mapping between the *NFG* and *RNFG*

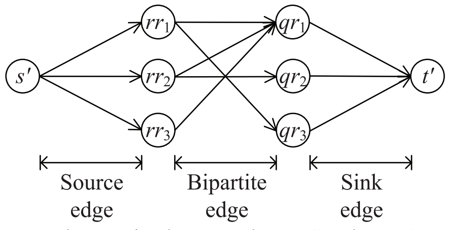
$s \rightarrow s'$	$s \rightarrow s'$
$r_1 \rightarrow rr_1$	$q_1, q_2 \rightarrow qr_1$
$r_2 \rightarrow rr_2$	$q_3, q_4 \rightarrow qr_2$
$r_3 \rightarrow rr_3$	$q_5, q_6 \rightarrow qr_3$
$r_4, r_5 \rightarrow rr_4$	$q_7 \rightarrow qr_4$

(a)

Source edge	(s', rr_1)	(s', rr_2)	(s', rr_3)	(s', rr_4)
	(2, 1)	(2, 1)	(2, 1)	(4, 1)
Bipartite edge	(rr_1, qr_1)	(rr_1, qr_3)	(rr_1, qr_4)	
	(2, 20)	(2, 40)	(1, 85)	
	(rr_2, qr_1)	(rr_2, qr_2)		
	(2, 20)	(2, 60)		
	(rr_3, qr_1)			
	(2, 110)			
	(rr_4, qr_1)	(rr_4, qr_3)	(rr_4, qr_4)	
	(4, 110)	(4, 135)	(2, 110)	
Sink edge	(qr_1, t')	(qr_2, t')	(qr_3, t')	(qr_4, t')
	(4, 1)	(6, 1)	(2, 1)	(1, 1)

(b)

Fig. 7. The reduced flow graph after the rack-based combination. (a) The reduced flow graph. (b) The pair (capacity, cost) on each flow edge.



Node mapping between the *NFG* and *RNFG*

$s \rightarrow s'$	$s \rightarrow s'$
$r_1, r_4, r_5 \rightarrow rr_1$	$q_1, q_2 \rightarrow qr_1$
$r_2 \rightarrow rr_2$	$q_3, q_4 \rightarrow qr_2$
$r_3 \rightarrow rr_3$	$q_5, q_6, q_7 \rightarrow qr_3$

(a)

Source edge	(s', rr_1)	(s', rr_2)	(s', rr_3)
	(6, 1)	(2, 1)	(2, 1)
Bipartite edge	(rr_1, qr_1)	(rr_1, qr_3)	
	(2, 110)	(2, 135)	
	(rr_2, qr_1)	(rr_2, qr_2)	
	(2, 20)	(2, 60)	
	(rr_3, qr_1)		
	(2, 110)		
Sink edge	(qr_1, t')	(qr_2, t')	(qr_3, t')
	(4, 1)	(6, 1)	(3, 1)

(b)

Fig. 8. The reduced flow graph after the equivalent state combination. (a) The reduced flow graph. (b) The pair (capacity, cost) on each flow edge.

respectively. Among all the requested and qualified node pairs between rr_m and qr_n, the maximum storage cost is used to represent the storage cost of qr_n with respect to rr_m. In (13), the maximum storage cost is for easily determining whether there is a link between rr_m and qr_n on the reduced flow graph. If the storage cost of qr_n can meet the QoS requirement of rr_m, there is a link between rr_m and qr_n. This also represents that if a node in rr_m reads a corrupt data block, it can retrieve the data block replica from a node in qr_n with the QoS satisfaction.

Fig. 7 illustrates the reduced network flow graph ($RNFG$) after executing the rack-based combination on the network flow graph of Fig. 5. In addition to combining requested and qualified nodes, it is also required to combine the replication requests, replication space, and storage costs based on the above equations. From Fig. 7, we can see that the requested nodes r_4 and r_5 in Fig. 5 are combined as the requested rack node rr_4. The qualified nodes q_1 and q_2 are combined as the qualified rack node qr_1. The total number of replication requests in rr_4 is 2 $(1 + 1)$. The total size of the replication space in qr_1 is 4 $(2 + 2)$. The storage cost of qr_1 with respect to rr_4 is 110 (max {105, 110, 105, 110}).

For the basic idea of the equivalent state combination, it is inspired as follows: In the reduced network flow graph, two or more requested rack nodes may have the same service state, i.e., they own the same qualified rack node set. These requested rack nodes can be further combined as a single requested rack node. Similarly, we can also combine two or more qualified rack nodes in the reduced network flow graph if they have the same request state, i.e., they are corresponding to the same requested rack node set. By applying the equivalent state combination in Fig. 7, the numbers of requested and qualified rack nodes can be further reduced, as shown in Fig. 8. Compared to the original network flow graph, there are fewer nodes in the reduced network flow graph. The existing polynomial-time MCMF algorithm can take less computational time on the reduced network flow graph. However, in the reduced flow graph, a node practically represents a group of nodes by performing the rack-based combination and equivalent state combination on the original network flow graph. It cannot guarantee that the optimal solution of the QADR problem can be obtained from the MCMF solution of the reduced network flow graph.

To represent the solution of the QADR problem, the MCMF solution of the reduced flow graph is required to be further processed. In the node combination techniques, the QoS requirement of a requested rack node and the storage cost of a qualified rack node have been defined in (11) and (13), respectively. In the two equations, the minimum and maximum functions are used to choose the minimum QoS requirement and maximum storage cost. With these two equations, if there is a link between the requested rack node rr_m and the qualified rack node qr_n on the reduced flow graph, it means that each qualified node in qr_n can store the data replica of a requested node in rr_m under the capacity constraint of the qualified node. This property will simplify the refinement process of the data replica placement.

For example, in Fig. 8, the MCMF solution of the reduced network flow graph determines that the amount of the flow from the requested rack node rr_1 to the qualified rack node qr_3 is 6. The requested rack node rr_1 consists of the nodes r_1, r_4, and r_5 on the original network flow graph. For the qualified rack node qr_3, it is formed by combining the qualified nodes q_5, q_6, and q_7. Therefore, in Fig. 8, the requested nodes r_1 in rr_1 can send one data block replica to the qualified nodes q_5 and q_6 in qr_3. For the requested nodes r_4 in rr_1, its data block is replicated in the qualified nodes q_1 and q_2 in qr_1. Finally, in rr_1, the two data block replicas of the requested nodes r_5 are stored in the qualified nodes q_5 and q_7 in qr_3. The above replica placement considers the capacity constraint of each qualified node.

4 Performance Evaluation

We used MatLab [32] to evaluate the performance of the proposed replication algorithms in a large-scale cloud computing system. Our simulation experiments were conducted by assuming that there are 3,500 nodes in a cloud computing system. The HDFS cluster at Yahoo! includes about 3,500 nodes [10].

4.1 Simulation Environment

In simulation experiments, we adopt a tree structure to be the network topology of the referred cloud computing system. As explicitly stated in Section 2.1, the switches of the cloud computing system usually use the STP protocol. The STP protocol can form a logical tree network topology among switches to provide the intrarack and inter-rack node communication. To simulate the tree network topology, the rack (group) formation and node distribution are done as follows: We assume that there are 100 racks in the cloud computing system, and each rack is equipped with one switch. The 100 racks are randomly distributed over a $1{,}000 \times 1{,}000$ unit square plane. A rack occupies a 10×10 subsquare plane. For any two racks, there is no intersection area in their corresponding subsquare planes. Among the 100 racks, one is specified as the central rack to organize all other racks as a tree topology with the height about 10. After forming the 100 racks with the tree structure connectivity, 3,500 nodes are randomly deployed within the 100 racks. For two nodes in the same rack, their locations are within the occupied subsquare plane of the rack.

Based on the generated network topology, the simulation experiments were performed over the following parameter settings. In each node, the available replication space is represented as the maximum number of data block replicas allowed to be stored. It is set by randomly selecting a number from the data block interval of [0, 50]. Similarly, a QoS interval is also used to set the QoS requirement of an application in the node r_i. The lower bound of the QoS interval is the time to access a data block from the local disk of r_i. The upper bound is the largest access time for r_i to retrieve a data block replica from another node. Next, a random number is selected from the setting QoS interval to be the QoS requirement of the application.

TABLE 2
Disk Access Time and Transmission Rates for a Data Block (64 Mbytes)

Disk access time (seconds)					
1.58	0.79	0.53	0.39	0.32	0.26

Transmission rate (megabits per second)					
100	1024	10240	16384	24576	32768

The Performance values refer to [33]–[36].

The performance values refer to [33], [34], [35], [36].

In a simulation run, we randomly specify 2,500 nodes to execute data-intensive applications, not all 3,500 nodes. Due to running the data-intensive applications, the 2,500 nodes frequently need to write data to their disks. As a result, there are many requested nodes that issue replication requests concurrently. The numbers of request nodes are assumed to be 500, 1,000, 1,500, 2,000, and 2,500. For the settings of the disk access latency and communication latency, we consider the dynamical workloads in the disk queue and communication link queue in addition to the given performance values of Table 2. The details are described as follows.

For the settings of the disk access latency and communication latency, we refer to commercial disk and switch equipment [33], [34], [35], [36]. The used disk access time and switch transmission rates in simulation experiments are given in Table 2. In addition, we also consider the dynamical workloads in the disk queue and switch link queue. Whenever one or more requested nodes concurrently issue their replication requests, there have been a random number of ongoing block read-write operations between multiple node pairs. For an ongoing block read-write operation between node i and node j, it is executed as follows: First, node i reads a data bock from its disk, and then transmits the data block to node j. After receiving the data block, node j writes the data block into its disk. The number of ongoing block read-write operations is randomly set within [0, 50]. Due to the ongoing block read-write operations, there have been a certain number of workloads in the disk queue and switch link queue before issuing replication requests.

After the above simulation parameter settings, 50 simulation runs are performed. The simulation results give the mean of 50 simulation runs.

4.2 Simulation Results

To solve the QADR problem, we have proposed the HQFR algorithm and the optimal algorithm by transforming the QADR problem into the MCMF problem. Here, the optimal algorithm is also called the MCMF replication (MCMFR) algorithm. For considering the computational time of the MCMFR algorithm, the node combination techniques are also applied in the algorithm. The new MCMFR algorithm is named as the C_MCMFR algorithm. In this section, we will demonstrate the performance results of the HQFR, MCMFR, and C_MCMFR algorithm. In addition to these three algorithms, the random and Hadoop replication algorithms were also evaluated in simulation experiments. The random replication algorithm randomly places the replicas of a data block at any nodes.

Fig. 9 shows the total replication costs for different numbers of requested nodes from 500 to 2,500. In Fig. 9a, the cloud computing system is configured with nine (3×3)

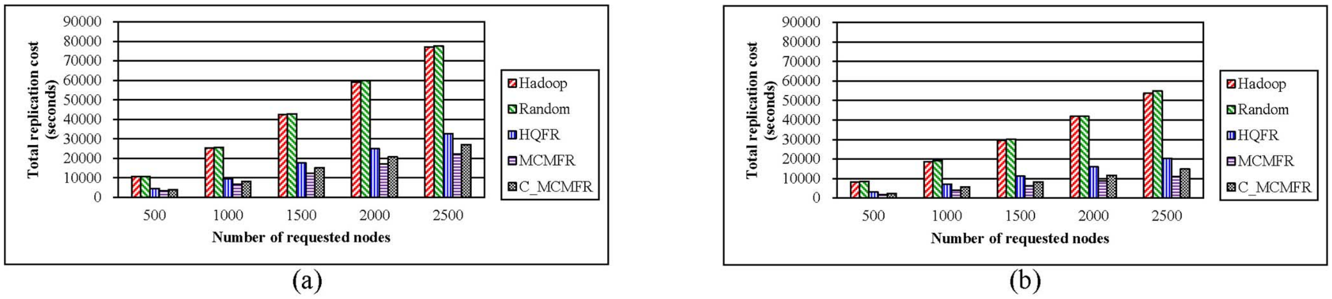


Fig. 9. Total replication cost under various device performance. (a) Nine types, (b) 36 types.

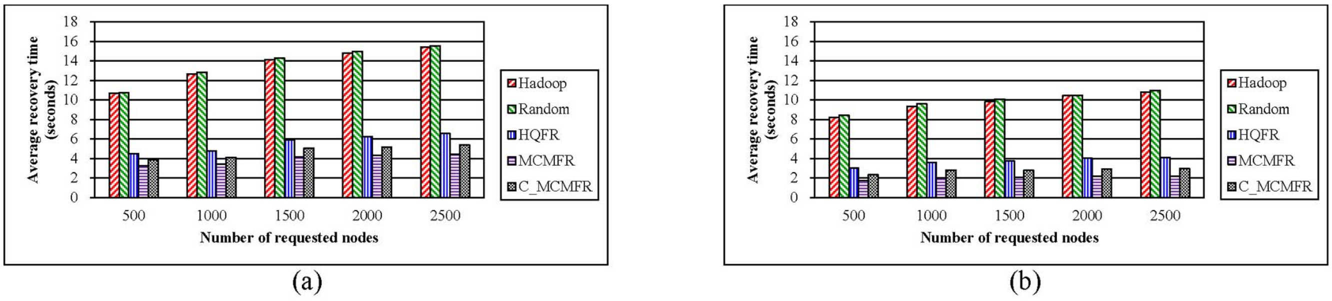


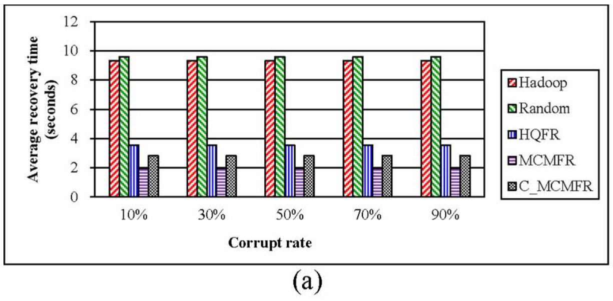
Fig. 10. Average recovery time under various device performance. (a) Nine types, (b) 36 types.

different types of device heterogeneity using the first three disk access time and transmission rates of Table 2. In Fig. 9b, all performance values of Table 2 are used to generate device heterogeneity with 36 (6×6) different types. As seen from Figs. 9a and 9b, the total replication costs of all algorithms increase with the number of requested nodes. The total replication cost has been defined in (3). The replication factor r_f is set to 2. Basically, the Hadoop replication algorithm adopts the random manner to place the replicas of a data block, but it additionally considers the possible rack failure. Therefore, the total replication cost of the Hadoop replication algorithm is similar to that of the random replication algorithm. Both algorithms do not take the QoS requirements of applications into data replication. These two algorithms have larger replication costs than the proposed replication algorithms. From Figs. 9a and 9b, we can also see that if the device performance is more diverse, our replication algorithms can reduce more replication costs than the Hadoop and random algorithms. As shown in Fig. 9a, the total replication cost of the Hadoop replication algorithm is about 2.47 times that of the MCMFR algorithm. However, in Fig. 9b, the total replication cost ratio between these two replication algorithms is about 3.79:1. For the proposed replication algorithms, the MCMFR algorithm can reduce the total replication cost of the HQFR algorithm by about 29 and 44 percent in Figs. 9a and 9b, respectively. Although the C_MCMFR algorithm reduces the computational time, it cannot minimize the replication cost. In the C_MCMFR algorithm, the QoS-violated data replicas are stored by randomly selecting a storage node from an unqualified rack node. However, the MCMFR algorithm also minimizes the total replication cost of QoS-violated data replicas in addition to the QoS-satisfied data replicas (see lines 19-22 of Fig. 6). Therefore, the MCMFR algorithm has a smaller total replication cost than the C_MCMFR algorithm. Compared to the MCMFR algorithm, it increases 21 and 36 percent of replication cost in Figs. 9a and 9b, respectively.

Fig. 10 shows the comparison of the average recovery time for a corrupt data block. If the requested node r_i cannot read a data block from its disk due to data corruption, how much time is taken by r_i to retrieve one replica of the data bock from another node? In Figs. 10a and 10b, the MCMFR algorithm has the smallest average recovery time, which can improve the average recovery time of the Hadoop algorithm by about 71 and 79 percent, respectively. From Figs. 10a and 10b, we also see that the average recovery time increases with the number of requested nodes in the proposed replication algorithms. The replication contention probability shows an upward growth trend with increasing number of requested nodes. Due to the limited replication space in a qualified node, a qualified node may not serve the replication requests from all its correspondingly requested nodes. As a result, some requested nodes cannot select their best qualified nodes to store their data block replicas. Later, if such a requested node reads a corrupt data block, it may take more time to retrieve the data block replica.

Unlike Fig. 10, the average recovery time shown in Fig. 11 is at different data corrupt rates. However, the average recovery time is also measured based on the unit of a corrupt data block. In Figs. 11a and 11b, the numbers of requested nodes are set as 1,000 and 2,000, respectively. The device heterogeneity is fixed with six types. When a failure occurs in the disk of a node, data blocks in the disk are not all corrupt. Different corrupt rates from 10 to 90 percent, respectively, are assumed. For the corrupt rate 10 percent, it denotes that if a failure occurs, 10 percent of data blocks are corrupt. From Fig. 11, we can see that the MCMFR algorithm still has the smallest average recovery time. Its recovery time is about one-fifth of the recovery time of the Hadoop algorithm.

Fig. 12 shows the recovery time in the worst case. The worst recovery time denotes the largest access time to retrieve only one failure-free data replica, which is measured by accumulating the access time of all data block replicas. As shown in Figs. 12a and 12b, the MCMF algorithm can decrease 71 and 21 percent of the worst



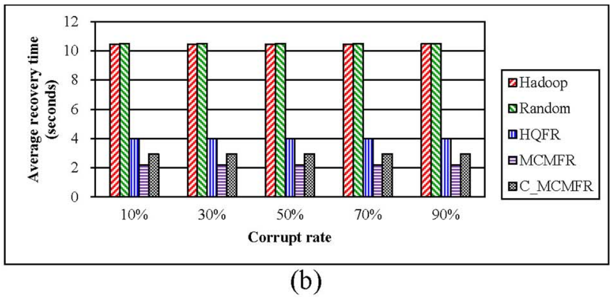
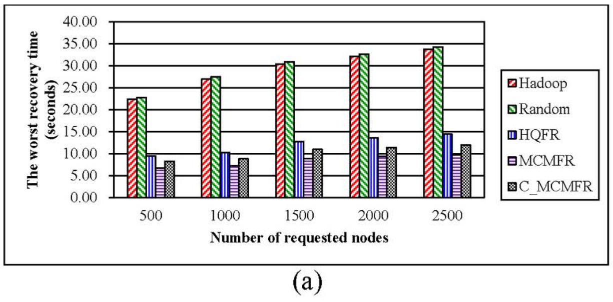


Fig. 11. Average recovery time under different corrupt rates: (a) 1,000 requested nodes, (b) 2,000 requested nodes.



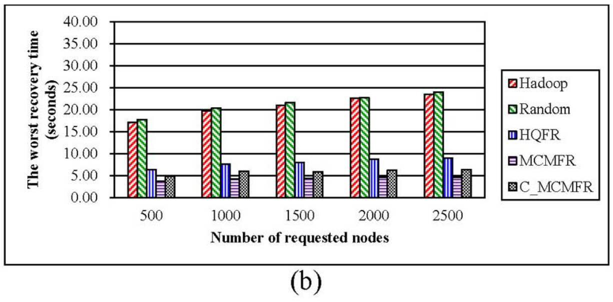
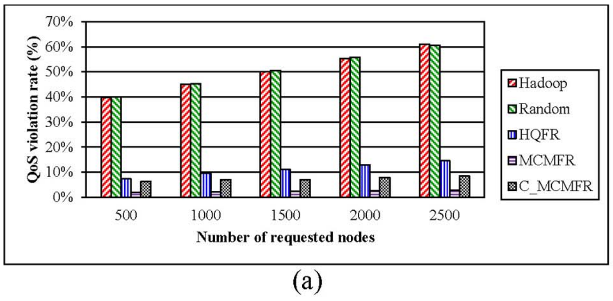


Fig. 12. The worst recovery time of the four replication algorithms. (a) Nine types, (b) 36 types.



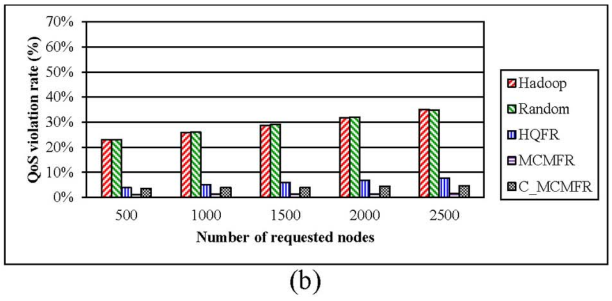


Fig. 13. The numbers of QoS-violated data blocks under various device performance. (a) Nine types, (b) 36 types.

recovery time of the Hadoop algorithm, respectively. Compared to the HQFR algorithm, the reduction ratio of the worst recovery time is about 30 and 45 percent in Figs. 12a and 12b, respectively.

Fig. 13 shows the comparison of the QoS violation ratios in the above concerned algorithms. The QoS violation ratio is defined as follows:

$$\frac{\text{The total number of QoS} - \text{violated data block replicas}}{\text{The total number of data block replicas}}. \tag{14}$$

In the Hadoop and random algorithms, the QoS requirement of an application is not considered in the data replication. In Figs. 13a and 13b, the QoS violation ratios of these two algorithms are approximately 50 and 28 percent, respectively. The QoS requirement is considered in the proposed replication algorithms. As mentioned in Section 3.2, the QoS-violated data replicas are generated due to the limited replication space of a node. In addition to minimizing the replication cost, the MCMFR algorithm can also minimize the number of QoS-violated data replicas. Compared to the HQFR and C_MCMFR algorithms, the MCMFR algorithm can reduce at least 78 and 67 percent of QoS-violated data replicas. Note that the main advantage of the C_MCMFR algorithm is in reducing the computation time of the QADR problem.

Concerning the scalable replication issue, we have utilized the rack-based and equivalent-state combination techniques to reduce the execution time in solving the QADR problem. To enhance this characteristic, we perform the execution time comparison among different replication algorithms, as shown in Fig. 14. The Hadoop and random algorithms take less execution time. Basically, these two algorithms randomly place data block replicas. By considering the QoS requirement in the data replication, the proposed replication algorithms are found take more execution time. Based on the given operation codes in Figs. 2 and 6, the HQFR and MCMFR algorithms do not perform complicated operations. Compared to the Hadoop algorithm, the execution time of the HQFR algorithm increases at least 1.24 times. The execution time of the MCMFR algorithm is about 6.4 times that of the HQFR algorithm. However, if the rack-based combination and the equivalent-state combination techniques are applied in the MCMFR algorithm, the execution time of solving the QADR problem can be reduced significantly. This can be obviously seen from the execution time of the C_MCMFR algorithm in Fig. 14. It is about 1.26 times that of the HQFR algorithm. From Fig. 14, we can also observe that the execution time of the C_MCMFR algorithm does not show a linear increase with varying number of requested nodes. The reason is explained as follows: Using the rack-based and equivalent-state combination techniques, the requested and

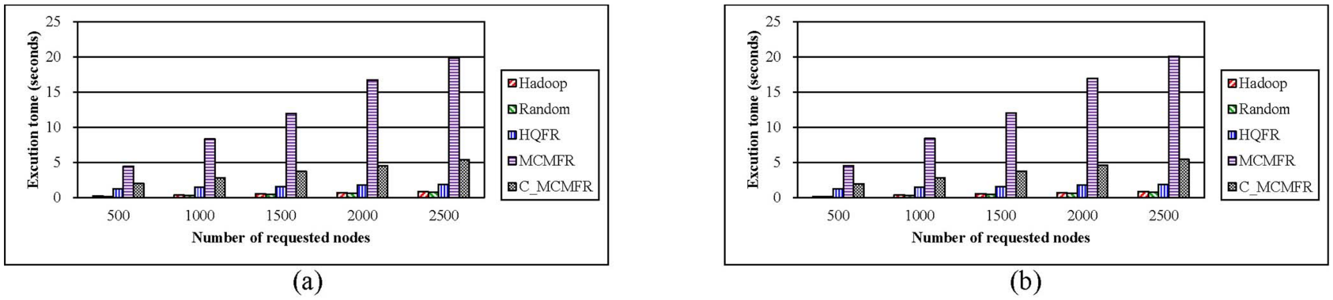


Fig. 14. Execution time of the four replication algorithms. (a) Nine types, (b) 36 types.

qualified nodes can be, respectively, combined as a smaller number of group nodes. In simulation experiments, the number of formed group nodes is at most 100 because there are 100 racks in the referred cloud computing system. Therefore, the execution time of the C_MCMFR algorithm cannot be large even if there are a large number of requested (qualified) nodes.

5 Conclusions and Future Work

We have investigated the QoS-aware data replication problem in cloud computing systems. To solve the QADR problem, the device heterogeneity is also considered in addition to the QoS requirements of applications. Two replication algorithms have been proposed. In the first algorithm, we adopt the intuitive idea of high-QoS first-replication to perform the QoS-aware data replication. However, this greedy algorithm cannot achieve the optimal solution to the QADR problem. The data replication cost and the number of QoS-aware data replicas cannot be minimized. In the second algorithm, we optimally solve the QADR problem in polynomial time by transforming the QADR problem to the MCMF problem. We also give the proof about the problem transformation (see Theorem 1). To make the proposed replication algorithms accommodate to a large-scale cloud computing system, we also present node combination techniques to handle the scalable replication issue of the QADR problem. The simulation results showed that the proposed replication algorithms can efficiently perform the QoS-aware data replication in cloud computing systems.

In the future, we plan to implement the proposed QADR algorithms in a real cloud computing platform. Moreover, the replication algorithms will also be extended to concern energy consumption. It is known that there are many storage nodes in a cloud computing system. The energy consumption is also an important metric for green cloud computing.

Acknowledgments

This research was supported by the National Science Council, Taiwan, R.O.C, under Grant NSC 102-2221-E-030-010-MY3.

References

[1] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R.H. Katz, A. Konwinski, G. Lee, D.A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Technical Report UCB/EECS-2009-28, Dept. of EECS, California Univ., Berkeley, Feb. 2009.

[2] M. Creeger, "Cloud Computing: An Overview," *Queue,* vol. 7, no. 5, pp. 2:3-2:4, June 2009.

[3] M.D. Dikaiakos, D. Katsaros, P. Mehra, G. Pallis, and A. Vakali, "Cloud Computing: Distributed Internet Computing for IT and Scientific Research," *IEEE Internet Computing,* vol. 13, no. 5, pp. 10-13, Sept. 2009.

[4] R. Buyya, C.S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the fifth Utility," *Future Generation Computer Systems,* vol. 25, no. 6, pp. 599-616, June 2009.

[5] Apache Hadoop Project, http://hadoop.apache.org, 2013.

[6] K.V. Vishwanath and N. Nagappan, "Characterizing Cloud Computing Hardware Reliability," *Proc. ACM Symp. Cloud Computing,* pp. 193-204, June 2010.

[7] E. Pinheiro, W.-D. Weber, and L.A. Barroso, "Failure Trends in a Large Disk Drive Population," *Proc. Fifth USENIX Conf. File and Storage Technologies,* pp. 17-28, Feb. 2007.

[8] B. Schroeder and G.A. Gibson, "Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You?" *Proc. Fifth USENIX Conf. File and Storage Technologies,* pp. 1-16, Feb. 2007.

[9] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, "Hadoop High Availability through Metadata Replication," *Proc. First Int'l Workshop Cloud Data Manage,* pp. 37-44, 2009.

[10] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop Distributed File System," *Proc. IEEE 26th Symp. Mass Storage Systems and Technologies (MSST),* pp. 1-10, June 2010.

[11] A. Gao and L. Diao, "Lazy Update Propagation for Data Replication in Cloud Computing," *Proc. Fifth Int'l Conf. Pervasive Computing and Applications (ICPCA),* pp. 250-254, Dec. 2010.

[12] W. Li, Y. Yang, J. Chen, and D. Yuan, "A Cost-Effective Mechanism for Cloud Data Reliability Management Based on Proactive Replica Checking," *Proc. IEEE/ACM 12th Int'l Symp. Cluster, Cloud and Grid Computing (CCGrid),* pp. 564-571, May 2012.

[13] C.N. Reddy, "A CIM (Common Information Model) Based Management Model for Clouds," *Proc. IEEE Int'l Conf. Cloud Computing in Emerging Markets (CCEM),* pp. 1-5, Oct. 2012.

[14] Amazon EC2. http://aws.amazon.com/ec2/, 2013.

[15] X. Tang and J. Xu, "QoS-Aware Replica Placement for Content Distribution," *IEEE Trans. Parallel and Distributed Systems,* vol. 16, no. 10, pp. 921-932, Oct. 2005.

[16] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *Proc. 19th ACM Symp. Operating Systems Principles,* vol. 37, no. 5, pp. 29-43, Dec. 2003.

[17] *IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges,* IEEE 802.1D Std., 2004.

[18] M. Shorfuzzaman, P. Graham, and R. Eskicioglu, "QoS-Aware Distributed Replica Placement in Hierarchical Data Grids," *Proc. IEEE Int'l Conf. Advanced Information Networking and Applications,* pp. 291-299, Mar. 2011.

[19] H. Wang, P. Liu, and J.-J. Wu, "A QoS-Aware Heuristic Algorithm for Replica Placement," *Proc. IEEE/ACM Seventh Int'l Conf. Grid Computing,* pp. 96-103, Sept. 2006.

[20] X. Fu, R. Wang, Y. Wang, and S. Deng, "A Replica Placement Algorithm in Mobile Grid Environments," *Proc. Int'l Conf. Embedded Software and Systems (ICESS '09),* pp. 601-606, May 2009.

[21] A.M. Soosai, A. Abdullah, M. Othman, R. Latip, M.N. Sulaiman, and H. Ibrahim, "Dynamic Replica Replacement Strategy in Data Grid," *Proc. Eighth Int'l Conf. Computing Technology and Information Management (ICCM),* pp. 578-584, Apr. 2012.

[22] D. Gross and C.M. Harris, *Fundamentals of Queueing Theory,* third ed. John Wiley & Sons, 1998.
[23] H. Khazaei and J.M.V.B. Mii, "Performance Analysis of Cloud Computing Centers Using M/G/m/m+r Queuing Systems," *IEEE Trans. Parallel and Distributed Systems,* vol. 23, no. 5, pp. 936-943, May 2012.
[24] S. Bradley, A. Hax, and T. Magnanti, *Applied Mathematical Programming.* Addison-Wesley, 1977.
[25] S. Dasgupta, C.H. Papadimitriou, and U.V. Vazirani, *Algorithms.* McGraw-Hill, 2008.
[26] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin, *Network Flows: Theory, Algorithms, and Applications,* first ed. Prentice Hall, Feb. 1993.
[27] P.T. Sokkalingam, R.K. Ahuja, and J.B. Orlin, "New Polynomial-Time Cycle-Canceling Algorithms for Minimum-Cost Flows," *Networks,* vol. 36, no. 1, pp. 53-63, June 2000.
[28] C.-X. Xu, "A Simple Solution to Maximum Flow at Minimum Cost," *Proc. Second Int'l Conf. Information Eng. and Computer Science (ICIECS '10),* pp. 1-4, Dec. 2010.
[29] W. Lin and D. Qi, "Research on Resource Self-Organizing Model for Cloud Computing," *Proc. Int'l Conf. Internet Technology and Applications,* pp. 1-5, Aug. 2010.
[30] A. Voulodimos, S. Gogouvitis, N. Mavrogeorgi, R. Talyansky, D. Kyriazis, S. Koutsoutos, V. Alexandrou, E. Kolodner, P. Brand, and T. Varvarigou, "A Unified Management Model for Data Intensive Storage Clouds," *Proc. First Int'l Symp. Network Cloud Computing and Applications,* pp. 69-72, Nov. 2011.
[31] L. Xu and J. Yang, "A Management Platform for Eucalyptus-Based IaaS," *Proc. IEEE Int'l Conf. Cloud Computing and Intelligence Systems,* pp. 193-197, Sept. 2011.
[32] MathWorks - MATLAB and Simulink for Technical Computing, http://www.mathworks.com, 2013.
[33] Speed Considerations, http://www.seagate.com/www/en-us/support/before_you_buy/speed_considerations, 2013.
[34] Hard Disk Performance, Quality and Reliability, http://www.pcguide.com/ref/hdd/perf/index.htm, 2013.
[35] Latency on a Switched Ethernet Network, http://www.ruggedcom.com/pdfs/application_notes/latency_on_a_switched_ethernet_network.pdf, 2013.
[36] T. Shanley, *InfiniBand Network Architecture,* first ed. Addison-Wesley, 2002.

Jenn-Wei Lin received the MS degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 1993, and the PhD degree in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1999. He is a full professor at the Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan. He was a researcher at Chunghwa Telecom Co., Ltd., Taoyuan, Taiwan from 1993 to 2001. His current research interests include cloud computing, mobile computing and networks, distributed systems, and fault-tolerant computing.

Chien-Hung Chen received the BS degree in computer science and information engineering from the Chung Hua University, Taiwan, in 2008, and the MS degree in computer science and information engineering from Fu Jen Catholic University, Taiwan, in 2012. He is currently working toward the PhD degree at the Department of Electrical Engineering, National Taiwan University. His research interests include cloud computing, mobile networks, and fault-tolerant computing.

J. Morris Chang received the PhD degree in computer engineering from North Carolina State University. He is an associate professor at Iowa State University. His industry experience includes positions at Texas Instruments, Microelectronic Center of North Carolina, and AT&T Bell Laboratories. He received the University Excellence in Teaching Award at Illinois Institute of Technology in 1999. His research interests include cyber security, wireless networks, and embedded computer system. Currently, he is a handling editor of *Journal of Microprocessors and Microsystems* and the Middleware & Wireless Networks subject area editor of *IEEE IT Professional*. He is a senior member of the IEEE.