On the Dynamic Maintenance of Data Replicas based on Access Patterns in A Multi-Cloud Environment

Mohammad Shorfuzzaman
Department of Computer Science
College of Computers and Information Technology
Taif University, Taif, KSA

***Abstract*—Cloud computing provides services and infrastructures to enable end-users to access, modify and share massive geographically distributed data. There are increasing interests in developing data-intensive (big data) applications in this computing environment that need to access huge datasets. Accessing such data in an efficient way is deterred with factors such as dynamic changes in resource availability, provision of diverse service quality by different cloud providers. Data replication has already been proven to be an effective technique to overcome these challenges. Replication offers reduced response time in data access, higher data availability and improved system load balancing. Once the replicas are created in a multi-cloud environment, it is of utmost importance to continuously support maintenance of these replicas dynamically. This is to ensure that replicas are located in optimal data center locations to minimize replication cost and to meet specific user and system requirements. First, this paper proposes a novel approach to distributed placement of static replicas in appropriate data center locations. Secondly, motivated by the fact that a multi-cloud environment is highly dynamic, the paper presents a dynamic replica maintenance technique that re-allocates replicas to new data center locations upon significant performance degradation. The evaluation results demonstrate the effectiveness of the presented dynamic maintenance technique with static placement decisions in a multi-cloud environment.**

Keywords*—Multi-cloud environment; data replication; distributed algorithm; response time; dynamic maintenance; QoS constraint

I. INTRODUCTION

Cloud computing is appearing as an emerging technology that provides scalable computing system and offers endless opportunities for the computing community. It provides large-scale computing and storage resources comprising data centers [1], [2]. In a cloud computing scenario, computing and storage resources can be delivered as a service irrespective of the location and time as other necessary utilities in life [3]. In general, there are three categories of cloud computing architecture for the delivery of services, namely, Software as a Service (SAAS), Platforms as a Service (PAAS) and Infrastructure as a service (IAAS) are common [1]. In a SaaS architecture, a cloud service provider hosts and manages software applications intended for the end-users instead of letting them using locally-run applications. The IaaS architecture offers hardware and software resources for data storage and processing as well as networks and other necessary infrastructure for deployment of operating systems and applications. In a PaaS architecture, users are furnished with necessary tools and programming languages and the service provide hosts an application delivery platform to continuously support development and delivery of end-user applications [4].

Due to the use of scalable data centers in cloud computing, end users are relieved of the burden associated with application provisioning and management. Popular cloud services providers such as Amazon S3 [5], Google [6], App iCloud[1] , Microsoft Azure[2], and DropBox[3] are serving thousands of millions users through a huge number of servers distributed over many datacenters around the world. Hence, cloud computing infrastructures are being used for effective processing of large data sets without the huge upfront investments required to purchase traditional data centers. Accordingly, there are increasing interests in developing data-intensive (big data) applications in this computing environment that need to access huge datasets. For instance, Facebook, Twitter, and big data analytics applications, such as the Human Genome Project [7], are using cloud computing infrastructures for processing and analyzing their petabyte-scale data sets, using a computing framework such as MapReduce and Hadoop[4].

Data availability and accessing data efficiently is an important demand for these data intensive applications [8]. Furthermore, cloud infrastructure has heterogeneous resources and the resources have diverse performances. Also, there may be demands of different service quality requirements from different users. Besides, overall system performance is also a critical factor. To effectively address these challenges, the need for data replication is apparent. In data replication, data are replicated at different replicas to provide data access to the users in a nearby location. Replication techniques increase data availability and reduce cost of data access and response time [9], [3], [10]. It also distributes the workload to the replica servers by routing user requests to different sites. Replication techniques decrease congestion-related performance degradation probability by distributing the load of network to network of multiple paths. To obtain maximum gain from replication, strategic placement of the file replicas is critical [9], [11], [12].

[1] https://www.icloud.com/

[2] https://azure.microsoft.com/

[3] https://www.dropbox.com/

[4] http://www.slideshare.net/kevinweil/hadoop-at-twitter- hadoop-summit-201

Data replication technique is widely used in the cloud computing system to provide high data availability [13], [14], [15], [16]. While research has been done on replica placement, little has focused on the dynamic maintenance of the replicas over the time. Moreover, until now, replication in multi-cloud environments is not considered in the existing research. Therefore, determining data center locations of replicas in a multi-cloud environment mimicking general network topology is an open problem.

This paper first presents a new distributed algorithm that determines static locations for placing replicas in multi-cloud environments representing a general network topology to improve data availability and satisfy user demands for time critical data. The algorithm also maximizes the degree of satisfied users while minimizing aggregated response time upon data access. The proposed distributed replica placement algorithm provides benefits with respect to reliability and scalability issues. A multi-cloud environment is highly dynamic where user requests and network latency fluctuate persistently. Data centers that hold replicas currently may not be the best locations to obtain replicas later. Hence, an algorithm is proposed for the dynamic maintenance of replicas based on the user data access patterns and the cumulative aggregated response time. Overall, the paper makes the following contributions:

- Unlike existing data replication strategies in cloud computing system, the proposed replication algorithm is targeted for multi-cloud environments.
- A novel algorithm is presented for distributed replica placement in multi-cloud environments mimicking general graph topology.
- A dynamic maintenance algorithm is presented to relocate replicas in optimal locations to sustain overall response time to a minimal.

The rest of this paper is organized as follows. Section II presents some selected existing approaches from the literature. Section III describes the clustering approach to static replica placement. The proposed dynamic replica maintenance technique is presented in Section IV. Performance evaluation is described in Section V. Section VI concludes the paper and suggests directions for future work.

II. Related Work

There is a handful of data replication strategies found in the cloud computing environments. One of the earliest work proposed by Ghemawat et al. [6] where a static distributed data replication algorithm in clouds (Google File System – GFS) places data chunks based on a number of factors such as: 1) to choose replica locations in the chunk servers with below-average disk space utilization; 2) to set a maximum count on the replicas that are created recently on each chunk server; 3) to scatter replicas of a particular chunk across racks. The creation of data chunk replicas is triggered when the number of replicas drops below a specific level. The data in DataNode is protected against failure through replication technique. For instance, in Hadoop Distributed File System (HDFS) [14], to safeguard against failure a data block is replicated twice in different racks containing two DataNodes. The authors in [17] propose a dynamic data replication algorithm which is distributed in nature and targeted for cloud environment. The algorithm works a cloud system based on HDFS technology and decides to place replicas considering the current workload on the nodes and conforming to a lower bound on the number of total replicas that are created. Wang et al. [18] present a prediction based centralized cloud data replication technique that considers weighting factor for replication decision.

The authors in [19] made a number of contributions for cloud data replication as such: 1) came up with a model to link system availability and the number of replicas; 2) identify data items for replication based on popularity; 3) computed an appropriate number of replicas and locations based on an effective system byte rate. An adaptive technique [20] for cloud data replication is proposed which works based on file prediction. The technique takes availability and efficient access into account while predicting files in data centers. Besides replication, Gai and Daio [15] investigate replica consistency issue in cloud computing system. The authors adopt a lazy update scheme to split data access and update that can improve throughput and cut down response time. In addition, the authors in [21] work on a data management technique focusing on reliability issues. The technique works by checking the replicas proactively in an attempt to decrease the replica count which will reduce the storage usage in turn. A data storage mechanism [22] is proposed to enhance data availability and privacy which works by combining multiple clouds and a set of protocols such as Byzantine quorum system protocols, cryptographic secret sharing, and erasure coding. The evaluation results demonstrate that the proposed system brings forth a cost effective way to improve data availability and privacy for critical applications. In a relatively recent effort done by X. Wu [23], a cost effective data set replica placement strategy is proposed for cloud environments. The technique starts with designing a cost model for data management which includes storage cost and transfer cost. Then a replica placement technique is presented that decides the location of replicas using based on a location graph problem. The technique uses access frequency and average response time to decide which data set should be created. Experimental results demonstrate the benefit of the technique in term of lower management cost with fewer replicas.

Even though a quite a bit of work has been done to deal with replication issues in cloud computing environments, less attention was paid to address the issue of QoS requirement. The authors in [24] present two data replication algorithms that address the issue of QoS requirements in cloud systems. First, replication is done based on meeting high QoS requirement on a first come first service basis. Nevertheless, the performance of this algorithm is not satisfactory in reducing the replication cost and the number of unsatisfied requests. Hence, the second algorithm converts the replication problem into a minimum-cost maximum-flow (MCMF) problem which can generate an optimal solution to the problem in polynomial time. This problem has also been studied widely in data grid systems [25], [26], [27].

Even though the issue of dynamic maintenance of replicas in cloud environments is not well studied in the literature, some researchers addressed this issue. Liao et al. [28] propose a dynamic replica deletion strategy for distributed storage systems under cloud computing environments. The strategy aims at reducing storage that is occupied by the replicas and their maintenance cost. The authors come up with a mathematical model to illustrate the relationship between QoS requirement for the requested data and the number of replicas that need to be created. Performance results suggest that the proposed technique can save disk space and reduce maintenance cost while satisfying the QoS requirements. The authors in [29] present a file replication and consistency maintenance technique in Hadoop cluster. As claimed by the authors, Hadoop's strategy to maintain three replicas of each file leads to poor storage utilization and hence they propose integrated data replication and consistency maintenance (IRM) technique. They only create replicas for popular files and this method has already been proven to be an effective technique in the literature. The implementation includes the use of MapReduce programming model and HDFS in the clusters of commodity computers. The experimental results show that the technique can reduce data access time and increase data locality at the same time. A relatively earlier effort was done by Chun et al. [30] that deal with efficient replica maintenance in distributed storage systems. The proposed strategy works by aggregating disk spaces of many nodes over the Internet. The novelty of the strategy comes from the viewpoint of fault-tolerance in case of network and disk failure.

III. Clustering Approach to Static Replica Placement

This section will first describe the static replica placement (SRP) problem in a multi-cloud scenario that minimizes the number of replicas created and aims at meeting the user QoS requirements. The proposed solution starts with a system model that shows the targeted multi-cloud architecture where each cloud provider contains different number of data centers for storing replicas. Then, a static replica placement approach is presented based on clustering data mining technique.

A. System Model

Figure 1 shows a high level architecture of a multi-cloud setup comprising of a number of cloud providers interconnected multi-cloud proxies. User communication with the data centers in the cloud providers are facilitated through the multi-cloud proxies. Before describing the proposed clustering approach to static replica placement in this multi-cloud environment, a logical structure of the multi-cloud system model is presented as shown in Figure 2. The multi-cloud system which is considered here consists of $DC = \{DC_1, DC_2, \dots, DC_n\}$ different data centers and $U = \{U_1, U_2, \dots, U_m\}$ distributed end-users that share both resources and data in a multi-cloud system. This is modeled using an undirected graph $G = (DC, E)$. Here, DC is the set of data centers, and $E \subseteq DC \times DC$ is the set of links among the data centers. It is assumed that there is an upper bound bandwidth constraint $(bc(e_i))$ on each link $e_i \in$ Ecapacity. Also, each data center $DC_i \in DC$ is characterized by the following 4-tuple:

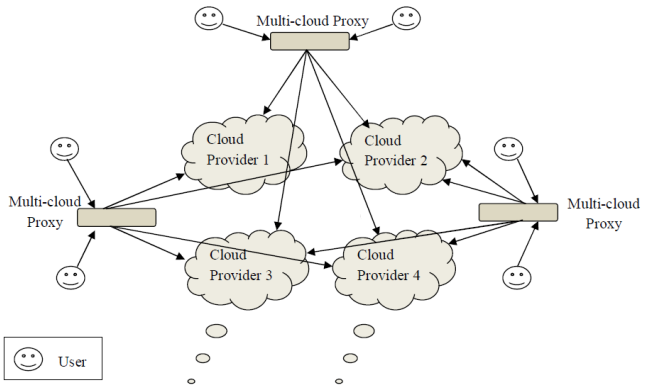


Fig. 1. A multi-cloud architecture comprising a number of cloud providers

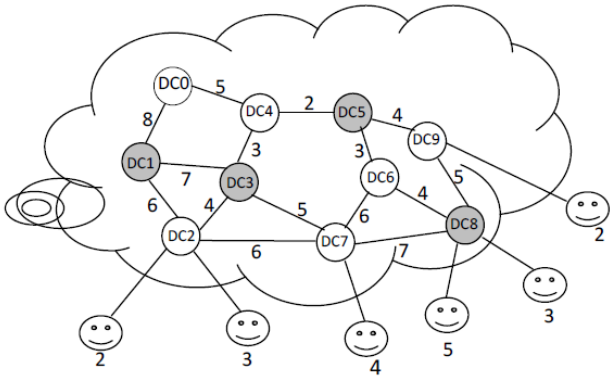


Fig. 2. A logical structure of a multi-cloud system model consisting of data centers and users

$$DC_i: \langle ID, S, C, L, d(u, v) \rangle \quad (1)$$

where ID is the data center *id*, S is the storage capacity of the data center, L is a workload capacity constraint, C is the storage cost to store a replica and $d(u, v)$ is a communication cost over link $(u, v) \in E$. It is assumed that for a same data file different data centers may have different storage costs. A data center containing a replica can be considered as a replica server.

The workload capacity constraint is defined as an upper bound on the user requests' number that is processed by the replica server during a specified time period. The replication strategy has to ensure that the user requests are satisfied while limiting the workload of each replica server to its capacity. Associated with different replica servers the workload constraint can be different. A replica server is overloaded if the total workload of the replica server becomes greater than the server's capacity constraint. In addition, as shown in Figure 2, end users are connected to the data centers. Note that each replica server (data center) provides services to multiple users and a user $U_i \in U$ is always associated with a specific server $DC_i \in DC$ in the multi-cloud structure. The number on the link denotes the cost of communication. It is considered that the logical graph is connected. Therefore, a replica server is able to communicate with any other server in the multi-cloud via some path.

A user $U_i \in U$ sends his/her requests to its associated server for retrieving data specifying QoS requirements. The

QoS is defined as an upper bound q (U_i) on the cost retrieval. Note that QoS requirements may be different for different users. The requirement specifies that all the requests by U_i will be handled by some servers by a communication delay q (U_i). The requirement is satisfied when a request is handled by the closest replica server which satisfies q (U_i). The requirement is violated otherwise.

Now, the QoS-aware static replica placement problem in hand is precisely defined. Let DC' represent a set of data centers that contain the replica of a requested file. Let $min(d_i)$ be the minimum communication time for the data center i to retrieve a requested file from all the probable retrieval paths from i to the set of replicas DC'. In case a data center holds the target replica, the value of $min(d_i)$ becomes 0. To this end, our goal is to calculate a replica set of minimum size, $DC' \subseteq DC$ such that the retrieval of the requested file at each data center in DC meets the QoS requirement, i.e., $min(d_i) < q$ (U_i) for each i in DC. Here, the communication cost between the users to the local data centers is not counted since the cost does not change the decision of replication.

B. Data Model

A central database (Central DB), located at the origin server (represented as $DC0$ in the graph) stores all the data required by the cloud applications. To reduce the response time, each data center maintains a local database, called datacenter database (Datacenter DB). The goal is to replicate the most frequently used data items from the central database. For maintaining consistency of data, the origin server sends updates to each replica server. It is assumed that updates must first be performed on the authoritative copy stored at the origin server, where they are then propagated to the replica servers. That is, updates only come from the origin server and the replica servers act as repositories for data retrieval. Popularity of a file is calculated using access rate and it is assumed that popularity persists into the near future.

In order to retrieving data a user $U_i \in U$ sends his/her requests to its associated data center specifying some QoS requirements. The request is processed locally if the data center contains the requested data. Otherwise, the request is forwarded to the nearest data center that contains the replica of the requested data. Thus, associated with every user $U_i \in U$ will be a non-negative weight $count(U_i)$ representing the traffic (data access counts which represent the data popularity) originating from that node over a certain period of time. These data access counts will contribute to the response time for servicing the data requests from the users.

When data is accessed, the information about requesting data center is stored. Additionally, the statistics showing the number of accesses and updates are obtained for each data item. The access rate or popularity is calculated in terms of the number of accesses over a period of time. The popularity of a data item varies over time in relation to the types of stored data. Generally, a newly created data has the highest popularity. Then, the access rate decreases over time. For example, a newly posted YouTube video is watched by most of the visitors. However, over the time its popularity and the number of visitors start to decrease.

C. Clustering Approach to Replica Placement

Let each data center in the multi-cloud be i which by now is acquainted with its directly adjacent data center. In addition, this data center i also knows about the communication delay to reach any neighboring data center k in the multi-cloud structure through the $i \rightarrow k$ path. Furthermore, a data center always maintains information about these direct adjacent data centers that store replicas of the requested data and the in-degree of each of them. The in-degree of a data center having replicas determine the percentage of other data centers that are connected to users and their requests are satisfied by this particular data center.

Each data center, i, connected to a user knows its QoS requirement (i.e., q (U_i)) to retrieve a data item. The data center incessantly checks its own status and the status of its neighboring data centers whether a replica of the requested data item exists in its own storage or at the storage of any of its neighboring data centers that can be communicated through a delay $\leq q$ (U_i). In course of time, if data center i or any of its neighboring data centers reachable within the stipulated delay becomes devoid of replica, any of the following two will occur. First, data center i will become a cluster core and it will copy the requested replica in its own storage if all its neighboring data centers are away with a delay $> q$ (U_i). Alternatively, data center I will select a core for the new cluster from any of the neighboring data centers that can be reached within q (U_i) delay and it has a highest in-degree. This core data center will now store replica to be accessed by other data centers that are connected to the users. This strategy tries to increase the count of data centers that can read this latest copy of replica.

IV. Dynamic Maintenance of Static Replica Placement

Clustering approach to replica placement ensures that replicas are created in near-optimal locations based on the user requirements for the current session. However, the data centers currently holding the replicas may not be the best locations for replicas to be there in future due to the changes in user requests and network conditions. Hence, dynamic maintenance of replicas which means relocation of replicas in optimal locations is necessary to sustain overall response time to a minimal. At the same time, the goal is to maximize the percentage of users whose QoS requirements are met.

A. Problem Formulation

The dynamic replica maintenance problem is now formulated in a multi-cloud environment. Formally, the problem of replica maintenance problem is expressed as an optimization problem as follows:

$$minimize_{\ response\_time} \ \& \ maximize_{\ user\ QoS}$$

The problem is to choose M replica locations among N potential data centers ($N > M$) by minimizing the overall response time and maximizing the rate of satisfied user QoS requirements considering a given traffic pattern (i.e., access frequencies' recurring pattern of users for different data items). More specifically, the goal is to identify a set of new data center locations with the minimal response time where

services for each user request can be provided by a data center within its quality requirement.

Now, the replica maintenance problem in hand is generalized. It is assumed that in cloud applications, updates to the data are infrequent and that the consistency can be more relaxed than in, for example, high-performance commercial databases. Given this, the proposed approach achieves greater scalability while making modest compromises in terms of update propagation and replica synchronization. In particular, the updates are delivered from the origin server (data center) to all other data centers having replicas via application-level multicast, in which each server (data center) receives the updates from its parent and is responsible for further distributing the updates to its children. Without loss of generality, it is assumed that this origin server constitutes the root of the tree that is going to be embedded. Given the embedded tree over the general multi-cloud, the replica maintenance problem can be modeled as a dynamic programming problem and its solution can be obtained in a distributed fashion. To solve the problem in hand using a dynamic programming approach, it is needed to solve different parts of the problem (also called sub-problems), then combine the solutions of the sub-problems to attain an overall solution. This approach tries to solve each sub-problem only once, thus reducing the number of computations. Hence, the new dynamic programming problem is a generalization of the replica maintenance problem we intend to solve, and thus the solution to this new problem also solves the original replica maintenance problem.

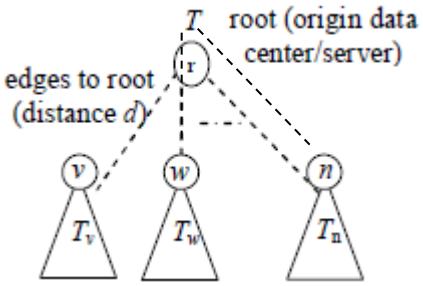


Fig. 3. An embedded tree with possible sub-trees representing the sub-problems

Let $T = (V, E)$ be the embedded tree over the general multi-cloud rooted at the origin server. T can be viewed as a combination of a number of sub-trees such as Tv, Tw, and so on where each sub-tree is connected to the origin server with a list of edges of length d. Now, the replica maintenance problem in T can be solved, by first solving an extended problem in a slightly different tree built from each sub-tree such as T_v and an additional edge list connected to it as shown in Figure 3. The edge-list is of length d and one side of it is connected to the root of T_v and the other side to the origin data center. At any data center from v to the root, there is a replica server data center containing the copy of the data file which can be accessed by the sub-tree T_v. Now, the total response time for sub-tree T_v can be calculated by considering the replica server at any distance towards the root. This response time considers the data access cost from the edge-list. Thus, this became a generalization of the original replica maintenance problem: when the edge-list length is zero and $T_v = T$, the replica maintenance problem is reduced to the original one.

Now, we formally formulate our goal for finding an updated set of replicas that has minimal response time so that QoS requirement of each user v is satisfied by $RU\{r\}$, i.e.,

$$\min_{s \in RU\{r\}} d(v, s) \leq q(v)$$

where, d(v, s) denotes the distance between v and s.

B. Dynamic Replica Maintenance Strategy

In dynamic replica maintenance strategy (DRMS), a shortest path tree is built at the onset. As noted previously, the updates are forwarded from the origin server data center to all the replica server data centers using application-level multicast. It is assumed that this origin server constitutes the root of the tree that is going to be embedded. To do this, the all-pairs shortest paths are calculated and a shortest path tree is built which has root, the origin server (see Figure 4).

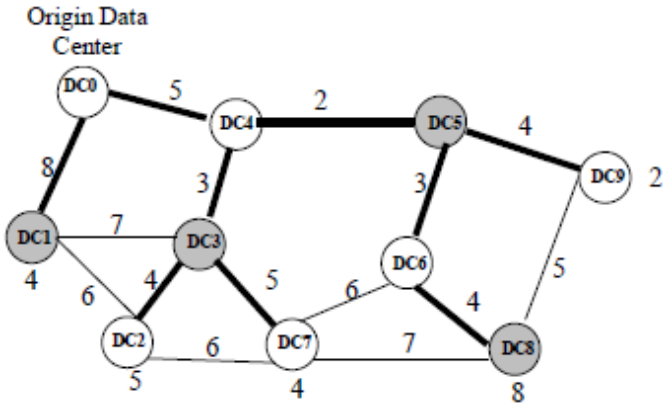


Fig. 4. An embedded all pairs shortest path tree in the multi-cloud system

Given the embedded tree over the multi-cloud model each data center node can determine the cost for generating a local replica and the cost of data transfer from a replica server data center anywhere towards the root. Then a parent data center node assesses the cost of creating a local replica versus the cost of data transfer using the results provided by its children. This process continues until it reaches the root data center of the embedded tree. Then, based on the assessed data, actual replica placement starts at the root data center and finishes at the leaves.

Let $RMC(v, rd)$ represents the replica maintenance cost contributed by the sub-tree with the root data center v when the replica is placed at a distance of rd towards the origin data center. The value of rd can vary from 0 (when the replica is in v itself) to the distance to the origin server. When rd is equal to 0 the replica maintenance cost considers read cost of all the descendants of data center node v, the storage cost, and the update cost for the replica at node v. On the contrary, when rd is greater than 0 (i.e., the replica is placed to any of the ancestors of node v except the root data center), RMC (v, rd) denotes the cost of replica maintenance for the sub-tree rooted at data center v that contains all its descendants' read cost.

In Figure 4, it is considered that the number of updates originated by the origin data center is 3 for a specific time period and the storage cost for the data file in the replica server is 10. As before, it is also assumed that accesses and updates need one unit of bandwidth for per file transfer per network hop. Now, if at "node 3" a replica is placed, the replica maintenance cost for the sub-tree with the root data center 3 is *RMC* (3, 0) = 40(read cost) + 24(update cost) +10 (storage cost) = 74. However, if at "node 4" a replica is placed, then the corresponding cost will be *RMC* (3, 1)= 67(read cost) + 0(update cost) +0(storage cost) = 67.

Now, in order to determine the locations of new replica based on changing popularity and replica update frequencies, the dynamic maintenance algorithm is called at regular intervals. If old replicas are still in a newly found replica set they are retained. From the old set, other replicas are removed and new replicas are generated as required. Let *OR* represents the set of replicas that are created by the distributed static replica placement strategy and *NR* represents the newly calculated updated set of replicas during maintenance process. Thus, the dynamic maintenance algorithm will create replicas contained in the set $NR - (OR \cap NR)$. The selected interval is calculated by the rate of request so that a short interval results for high arrival rates. This produces higher overhead but adapts more quickly in changing access patterns.

Calculation of replica maintenance costs by the terminal, non-terminal, and root data center nodes and the determination of new replica locations based on these calculated costs are done in a bottom-up fashion starting from the terminal data center nodes. Each data center, *v*, calculates the optimal replica maintenance cost for its sub-tree considering the replica location at any distance from *v* to the root data center. Then data center *v* determines the location of the replica whether it should be fetched from any data center located up on the path towards the root, or it should be created in its own storage based on the calculated replica maintenance cost. In case *v* is a non-terminal data center node, it is also possible that the replica should be created in children data center nodes of *v* if placing replicas in them results in lower replica maintenance cost. Data center *v* records these costs and data center locations for potential replica creation. It is important to note that a data center node commences the calculation of costs once all of its children nodes have finished calculating the same.

Now, the actual determination of new replica locations occurs in a top-down fashion starting from the root data center. More specifically, a data center will determine if it needs to create a replica on its own storage or not. The root data center starts the process by determining the minimum of the two replica maintenance costs calculated before and its associated replica location either in itself or in the children data center nodes down the hierarchy. Root data center node then forwards this location information to all of its children. Upon receiving this location information, each of the children data center nodes verifies it with its location value calculated before. If they match, a replica is created in its own storage and this information is further forwarded down the hierarchy. Otherwise, it forwards an updated location information that was received from the root data center to its children and this process continues till the bottom of the embedded tree is reached.

V. Performance Evaluation

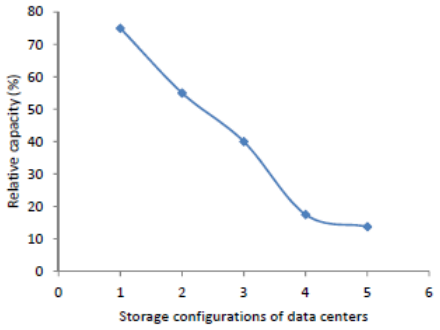
This section describes the performance analysis of the dynamic maintenance algorithm and compares it with the clustering based static placement as factors such as data center storage capacities, data access patterns, and user QoS constraints are varied.

A. Simulation Method

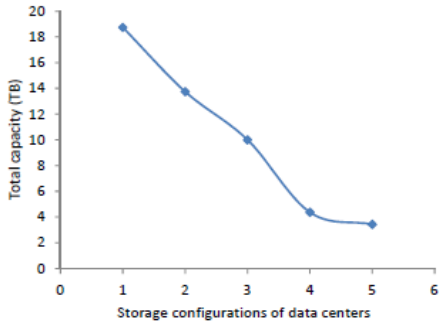
For the assessment of the algorithms, a Java based simulator program is used. The detailed configurations for the simulation are given below.

a) Testbed Environment

In the experiment, a multi-cloud composed of 156 data centers is used. Waxman model [31] is used to generate the multi-cloud topology. The available link bandwidth is computed using a uniform distribution with the range [0.622, 2.5] (Gbps). Data center storage capacities are also determined using the same distribution. The number of data files used in the system is 2500 with each data file size equal to 10 GB. This makes the total size of all data files used in the simulation approximately 25 TB. To measure the effectiveness of the algorithm a wide range of data center storage resource configurations are used in terms of the relative storage capacity, *r*, of the replica data center servers. Here, *r* is defined as a ratio of the total storage size of replica data center servers to the total size of all data files in the system. If *r* is 100%, it can be assumed that every data file could have a replica in the system. For the experiments *r* has been varied from 75% to 13% as shown in Figure 5.



(a)



(b)

Fig. 5. Relative storage capacity and total capacity for different data center storage settings in (a) and (b)

Each replica data center server can serve a number of data access requests from the users. The replica servers will run short of storage during the simulation. To place new replicas, a replacement strategy is necessary to ensure that new data files do not replace popular files. To this end, a modified Least Recently Used (LRU) replacement strategy was used based on the popularity of data to ensure that no replicas created in the in-progress replication period are removed.

B. Data Access Patterns

A number of data files are accessed by each job during the simulation. The simulation was conducted with 50 different jobs that were submitted with fixed probabilities. Some jobs were more popular than others. The data access requests from the users follow Poisson arrivals. Each user issues one access request on average per 2500 milliseconds and a data access pattern determines the sequence of the access requests. Two access patterns namely Gaussian random walk and the heavily tailed Zipf distribution were used. The Zipf distribution is given by: $P_i = K$, where P_i is the frequency of the ith ranked item, K is the popularity of the most frequently accessed data item and s determines the shape of the distribution. It is assumed that data access patterns can show temporal locality to some extent which means that recently accessed data are expected to be accessed again. Such an access pattern containing varying amount of temporal locality can be generated using Zipf distribution. Thus, in a system that is designed to react to file popularity, the Zipf distribution offers a natural testing ground. The index used to measure the amount of locality in the pattern is denoted by s. The observed parameter values are in the range of $0.65 < s < 1.24$. A higher value of s indicates an increased degree of locality. In this paper, we use $s = 0.85$ and 1.0 and refer to as Zipf-0.85 and Zipf-1.0 distribution respectively. Furthermore, Gaussian distribution is the most widely used family of distributions in statistics and many statistical tests are based on the assumption of normality. As such, it is a good base measure which can be used for easy informal comparison to known applications [27, 28].

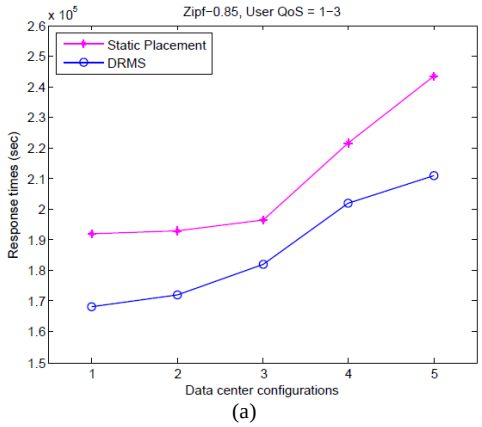
a) Performance Metrics

Each user data center site keeps record of the time required to receive a data item once it is requested. This elapsed time constitutes the foundation of assessing and comparing various replication strategies. Our dynamic replica maintenance strategy was assessed using the performance metrics which primarily include total response time in respect with job execution. Response time refers to the time that elapses from the moment when a data is requested until it is received and the specified job completes its execution. Total response time aggregates the response times of all the executed jobs for a simulation period. The goal is to achieve minimum total response time for our dynamic replica maintenance algorithm. The performance metrics also considers user satisfaction rate. User satisfaction rate is the proportion of users whose QoS constraints are met. The absolute values are actually of little interest but the relative performances demonstrate the superiority of dynamic maintenance algorithm over the static counterpart.

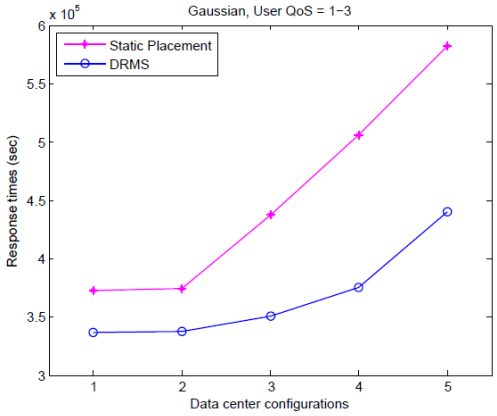
C. Results and Discussion

This section presents the experimental results of the static replica placement strategy and the dynamic replica maintenance strategy (DRMS) and compares them thoroughly based on the performance metrics. For a specific user, its QoS requirement is taken as a distance from the user to the closest replica data center server (such as number of hops) using a uniform distribution (i.e. the distance requests are uniformly distributed over the range). For example, a user QoS requirement of [1-3] implies that the closest data center with requested replica from the user should be any value between 1 and 3 and in such case the stipulated user QoS constraint deems to be satisfied.

We start by considering the algorithms' performances in terms of total response time, the major concern from the viewpoint of the data consumer. Figure 6 shows the approximate values of response times (y-axis) as a function of varying data center storage capacities (x-axis) for static replica placement and DRMS. In the experiment, a moderate workload capacity is considered for the replica server data centers. It is taken from a uniform distribution of [100-200] in terms of GB. The user QoS constraint on replica server data center distance of [1-3] is specified from a uniform distribution to allow relatively relaxed range. Our dynamic maintenance strategy DRMS that considers the relocation of replicas generally performs better than the static replica placement model in terms of response times for both Zipf and Gaussian data access patterns. The reason is that DRMS creates a modest number of well-placed replicas compared to the static counterpart which substantially reduces data access latency. Consequently, this decreases the overall response time. In addition, low running time of DRMS contributes to the reduction of its overall response time. With the decrease in storage capacity of data center replica servers the response time increases due to the creation of lower number of replicas.

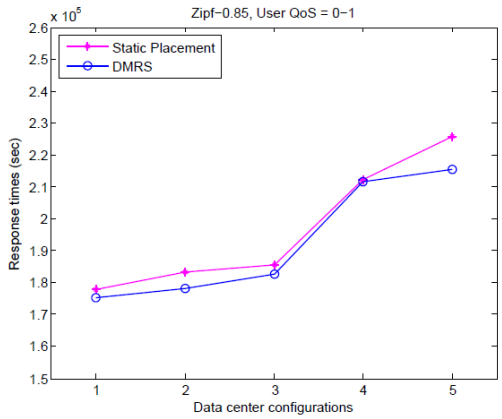


(a)

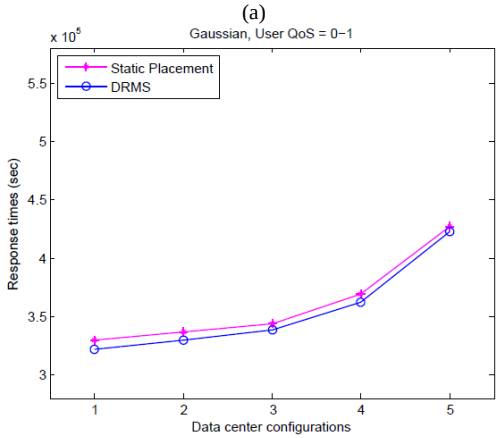


(b)

Fig. 6. Response times of static replica placement and DRMS considering relaxed QoS requirement [1-3] for Zipf-0.85 and Gaussian access patterns in (a) and (b)



(a)

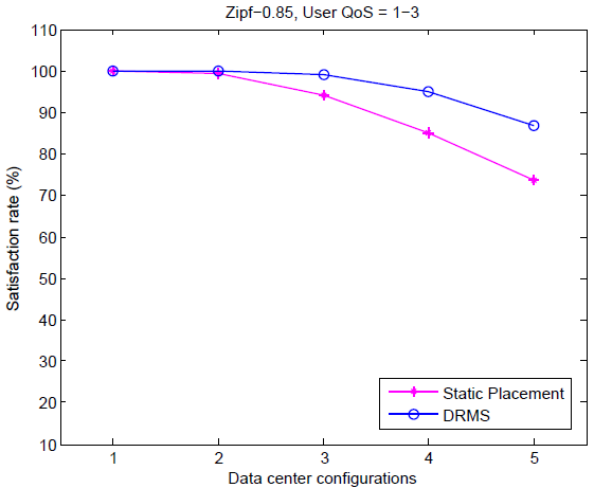


(b)

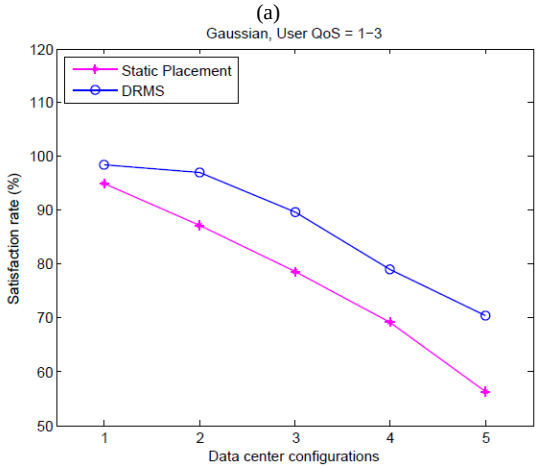
Fig. 7. Response times of static replica placement and DRMS considering more constrained QoS requirement [0-1] for Zipf-0.85 and Gaussian access patterns in (a) and (b)

For relatively more constrained QoS requirement ([0-1]) the performance improvement of dynamic maintenance model drops significantly for both Zipf and Gaussian access patterns as shown in Figure 7. In general, the performance benefit of DRMS over static placement becomes more obvious when user QoS requirements of wider ranges are considered.

Figure 8 demonstrates approximate values of user satisfaction rates for both the strategies using all storage configurations of data centers. DRMS performs better in most cases. Notably, the performance benefit of DRMS over static replica strategy is prominent when the storage capacity of data center servers becomes limited (for example in case of 17.5% and 13.75% relative capacities). However, user QoS satisfaction rates drop for both strategies in this case irrespective of the data access patterns and user QoS constraints used as shown in Figure 8.



(a)



(b)

Fig. 8. User satisfaction rates with relaxed QoS requirement [1-3] for Zipf-0.85 and Gaussian patterns in (a) and (b)

VI. Conclusions and Future Work

This paper investigates the dynamic replica maintenance problem in a multi-cloud scenario. To this end, a novel approach to distributed placement of static replicas in appropriate data center locations is proposed. Motivated by the fact that a multi-cloud environment is highly dynamic, the paper presents a dynamic replica maintenance technique that re-allocates replicas to new data center locations upon significant performance degradation. Performance analysis of the proposed techniques is done in terms of total response time and user satisfaction rates. The simulation results showed that the proposed dynamic maintenance technique, DRMS, can considerably reduce response times compared to the static

counterpart. In addition, user satisfaction rates are shown to be relatively higher due to dynamic replica maintenance. These benefits are attained using a wide range of storage configurations of data center servers and data access patterns with a degree of temporal locality and randomness. 501554-3-Distributed Systems.

In the future, we plan to implement the proposed dynamic replica maintenance algorithm in a real multi-cloud platform. Moreover, the algorithm will also be extended to deal with the peak bandwidth usage due to network link constraints and traffic patterns.

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] J. 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.

[3] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman, "Data replication strategies in grid environments," in Proc. of the Fifth Intl. Conf. on Algorithms and Architectures for Parallel Processing, pp. 378–383, 2002.

[4] I.S. 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] Amazon, Amazon simple storage service (Amazon S3). Available: http://aws.amazon.com/s3

[6] S. Ghemawat, H. Gobioff, and S. T. Leung, "The Google file system," SIGOPS Oper. Syst. Rev., vol. 37, no. 5, pp. 29-43, 2003.

[7] Human Genome Project, http://www.ornl.gov/hgmis/home.shtml.

[8] A. Chervenak, "Giggle: A framework for constructing scalable replica location services," IEEE Supercomputing, pp. 1–17, 2002.

[9] J. H. Abawajy and M. Deris, "Data replication approach with data consistency guarantee for data grid," IEEE Transactions on Computers, vol. 63, no. 12, pp. 2975 – 2987, 2014.

[10] K. Ranganathan, A. Iamnitchi, and I. Foster, "Improving data availability through dynamic model driven replication in large peer-to-peer communities," in Proc. of the 2nd IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGRID'02), pp. 376–381, 2002.

[11] J. Abawajy, "Placement of file replicas in data grid environments," in Proc. of the Intl. Conf. on Computational Science, vol. 3038, pp. 66–73, 2004.

[12] K. Kalpakis, K. Dasgupta, and O. Wolfson, "Optimal placement of replicas in trees with read, write, and storage costs," IEEE Transactions on Parallel and Distributed Systems, vol. 12, no. 1, pp. 628–637, 2001.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] Q. Wei, B. Veeravalli, G. Bozhao, Z. Lingfang, and F. Dan, "CDRM: A cost-effective dynamic replication management scheme for cloud storage cluster," in 2010 IEEE International on Cluster Computing. pp. 188 – 196, 2010.

[18] S. Wang, K. Q. Yan, and S. C. Wang, "Achieving efficient agreement within a dual-failure cloud-computing environment," Expert Syst. Appl., vol. 38, no. 1, pp. 906-915, 2011.

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

[20] M. Hussein and M. Mousa, "A light-weight data replication for cloud data centers environment," International Journal of Innovative Research in Computer and Communication Engineering, vol. 2, no.1, pp. 2392-2400, Jan 2014.

[21] 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.

[22] A. Bessani, M. Correia, B. Quaresma, F. Andr'e, and P. Sousa, "DepSky: Dependable and secure storage in a cloud-of-clouds," in Proc. of the 6th Conference on Computer Systems, pp. 31-46, 2011.

[23] X. Wu, "Data sets replicas placements strategy from cost-effective view in the cloud," Scientific Programming, vol 2016, pp. 1-16, 2016.

[24] J. Lin, C. Chen, and J. M. Chang, "QoS-aware data replication for data-intensive applications in cloud computing systems," IEEE Transactions On Cloud Computing, vol. 1, no. 1, pp. 101-115, 2013.

[25] X. Fu, R. Wang, Y. Wang, and S. Deng, "A replica placement algorithm in mobile grid environments," Proc. Int'l Conf. on Embedded Software and Systems (ICESS '09), pp. 601-606, May 2009.

[26] 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. on Computing Technology and Information Management (ICCM), pp. 578-584, Apr. 2012.

[27] C. Cheng, J. Wu, and P. Liu, "Qos-aware, access-efficient, and storageefficient replica placement in grid environments," Journal of Supercomputing, vol. 49, no. 1, pp. 42–63, 2009.

[28] B. Liao, J. Yu, H. Sun, and M. Nian, "A QoS-aware dynamic data replica deletion strategy for distributed storage systems under cloud computing environments," in Int'l Conf.on Cloud and Green Computing (CGC), pp. 219-225, 2012.

[29] A. R. Varma and A. K. Shrivastava, "File replication and consistency maintenance in the Hadoop cluster using IRM technique," Int'l Journal of Advanced Research in Computer Engineering & Technology (IJARCET), vol. 3, no. 7, pp. 2424-2428, 2014.

[30] B. Chun, F. Dabek, A. Haeberlen, E. Sit, H. Weatherspoon, M. Kaashoek, J. Kubiatowicz, and R. Morris, "Efficient replica maintenance for distributed storage systems," in Proc. of the 3rd conf. on Networked Systems Design & Implementation, vol. 3, 2006.

[31] H. Wang, P. Liu, and J. Wu, "A QoS-aware heuristic algorithm for replica placement," Journal of Grid Computing, pp. 96–103, 2006.