A Proactive, Cost-aware, Optimized Data Replication Strategy in Geo-distributed Cloud Datastores

Ta-Yuan Hsu
University of Illinois at Chicago
Chicago, Illinois
thsu4@uic.edu

Ajay D. Kshemkalyani
University of Illinois at Chicago
Chicago, Illinois
ajay@uic.edu

Abstract

Geo-replicated cloud datastores adopt the replication methodology by placing multiple data replicas at suitable storage zones. This can provide reliable services to customers with high availability, low access latency, low system cost, and decreased bandwidth consumption. However, this has the potential to increase the whole system overheads of maintaining more resource replicas, and to also degrade the system utilization due to unnecessary storage space cost. Thus, it is important to determine the suitable replication zones on-the-fly to increase the availability of data resources and maximize the system utilization. Specifically, it is essential to determine the appropriate number of replicas for different data resources at each zone in a particular time interval. We propose Cost Optimization Replica Placement (CORP) algorithms to enable state-of-art proactive provisioning replication of data resources based on an one-step look-ahead workload behavior pattern forecast over the distributed data storage infrastructure using statistical techniques. The experimental results show the cost effectiveness of the proposed replication strategies.

CCS Concepts

• **Networks** → *Cloud computing*; **Network simulations**; • **Information systems** → **Remote replication**; • **Computing methodologies** → **Distributed algorithms**.

Keywords

data replication; workload prediction; cloud storage; social networks

ACM Reference Format:
Ta-Yuan Hsu and Ajay D. Kshemkalyani. 2019. A Proactive, Cost-aware, Optimized Data Replication Strategy in Geo-distributed Cloud Datastores. In *Proceedings of the IEEE/ACM 12th International Conference on Utility and Cloud Computing (UCC '19), December 2–5, 2019, Auckland, New Zealand.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3344341.3368799

1 Introduction

The delivery of cloud computing resources as a utility provides users with flexible services in a transparent manner, with cheap costs and more customized operations. One may pay for such demand resources to cloud data providers (CDPs) as per their usage like a utility. As outsourcing continues to rise in popularity, the widespread adoption of cloud-based data services in distributed systems raises the challenges of providing high availability and efficient access to resources due to the larger scale. Data replication is commonly used in data intensive applications (e.g., social networks) to reduce access latency and enhance the data availability and reliability by provisioning appropriate replicas of data resources situated at different geographical locations. However, the overheads of loading and maintaining these replicas become higher and expensive. Thus, a fine balance of the advantages and overheads of replication is needed.

Static replication of data resources in dynamic environments having time-varying workloads is ineffective for optimizing system utilization. For example, the operating behaviors of social media applications vary based on different time intervals (e.g., morning vs. afternoon; Monday vs. Sunday). For example, for a specific time period, it is not required to create replicas for those data items with lower requested frequency in that time period. Adaptive data replication algorithms can address the above issue by enabling dynamic provisioning of data resources to applications based upon the patterns of workload traces. Practically, there are three fundamental issues in respect to the dynamic replication process. The first is to select which data items need to be replicated. The second is to select how many replicas should be loaded in the whole cloud system. The third is to decide the locations where the replicas should be placed. Two different dynamic provisioning approaches are widely used. *Reactive* approaches pre-define thresholds to specify the solutions to the above issues. *Proactive* approaches monitor, predict, and capture changes in workload patterns to manipulate data resource placement.

Contributions: In this paper, we consider the problem of optimizing the cost on the distributed data storage systems. We propose Cost Optimization Replica Placement (CORP) algorithm and design a proactive provisioning replication scheme across multiple CDPs. According to current data resource allocation and historical changes in workload patterns, our replication framework, composed of CDPs, is designed to employ the autoregressive integrated moving average (ARIMA) model to concretely predict how many data access requests are made in the near future. CDP can dynamically deploy required data replicas in suitable geo-locations for serving the predicted requests. Since caching has potential for performance benefits, we also extend CORP as CORP (+cache) for different data-intensive workloads. We further propose the optimal placement solution to evaluate CORP (+cache) in steady states. We conduct an evaluation of cost-effectiveness of our CORP algorithms via trace-driven CloudSim simulator toolkit and realistic workload traces from Twitter. Results show that CORP with cache mechanism can highly reduce the total system cost in comparison to the stand-alone caching strategy and static partial replication.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 gives the application and system models that support our workload prediction architecture. Section 4 gives our proposed approach along with the details of CORP and CORP(+cache) algorithms. Section 5 shows the simulation experiments along with the cost effectiveness evaluation of the algorithms in comparison with other traditional approaches for the same time slot system. It also evaluates CORP (+cache) with respect to the optimal placement strategy and illustrates the trade-off between them. Section 6 gives a discussion and concludes.

2 Related Work

The challenge of dynamic resource management and allocation in distributed systems and cloud environments has been dealt with via several *reactive* approaches. Evaluated from different kinds of users' access patterns, six replication strategies are proposed for hierarchically distributed data grids in an initial work on dynamic data replication [19]. The users' dynamic and distributed nature has been used in [18] to design suitable replica placement strategies in the grid environment. A dynamic data replication mechanism called Latest Access Largest Weight (LALW) is proposed in [7]. A dynamic distributed cloud data replication algorithm CDRM is proposed in [25]. CDRM is a cost-effective framework for replication designed on the HDFS platform in a cloud storage system. Dynamic data replication strategy (D2RS) in hierarchical cloud environments to improve system availability is proposed in [23]. Combined with a checkpoint strategy, the above algorithm is extended as a new algorithm called Dynamic Adaptive Fault-tolerance (DAFT) in [22]. It can dynamically provision and deliver computing resources in a transparent manner to achieve higher availability, performance, and reliability. Based on D2RS, the concept of knapsack has been used in [9] to optimize the cost of data replication between data centers without impacting the data availability. Because of pricing differences among different resources across cloud data stores, a lightweight heuristic solution to minimize monetary cost of the application in hot-spot or cold-spot objects is proposed in [15]. Based on the above system framework, an optimal offline algorithm and two online algorithms are further proposed in [16]. The offline algorithm is provided to minimize the cost of different access operations and potential migration, while satisfying eventual consistency and latency. The two online algorithms (deterministic and randomized) are designed to make a trade-off between residential and migration costs and dynamically select storage classes with provable performance guarantees. Based on the scale of applications, and dynamic workloads, the models proposed in [20] optimize request response time during normal operations to meet SLA (scalability, latency, and availability) requirements. A hierarchical data replication strategy (HDRS) proposed in [14] can dynamically store replicas, based on the access load and labeling technique, to offer high data availability and low bandwidth consumption, while satisfying QoS requirements and storage capacity constraints.

The weakness of reactive replication approaches is their inability to anticipate the unexpected changes in workload. Since the workload changes in cloud environments follow patterns that may vary from time to time, *proactive* methods can overcome the above deficiency by pre-deciding the correct amount of resources before the expected increase or decrease of demand. An approach to the problem of workload prediction based on identifying similar past occurrences of the current short-term workload history in public clouds is proposed in [6]. An event-aware strategy to more effectively predict workload bursts by exploiting prior knowledge associated with scheduled events is proposed in [21]. Artificial neural networks (ANN) and linear regression were used to develop prediction-based resource measurement and provisioning strategies in [11]. A cloud workload prediction module for SaaS providers based on the ARIMA model is proposed in [3]. This model can meet the QoS with a cost-effective amount of resources by estimating the future requirements.

3 System Cost Model

3.1 Adaptive Cloud Data Provider Architecture

Our intention is to design a system architecture that can support modern social web storage services. The whole framework is a hierarchical geo-distributed cloud data store system composed of multiple geographically distributed store servers (see Figure 1). They are deployed in different zones dispersed across the world. (e.g., TX state is one zone). For simplicity, each zone employs only one store server (ZS). Multiple ZSs, which are connected with lower network access cost, constitute a Geo-Data Cluster (*GDC*). Multiple *GDCs* are fully connected by WANs with higher network access cost. Each cloud data object is replicated in a subset of these ZSs. Since we focus on geo-replication, each zone hosts at most one replica (this may be straightforwardly extended as multiple replicas). Each ZS hosts one Cloud Data Provider (CDP) to manage cloud data replication for end-user access services. The top layer of CDP receives access requests, which are processed by the middleware layer. The system architecture model of CDP is schematized in Figure 2, where it presents the key components of our approach. This is patterned along [3, 5].

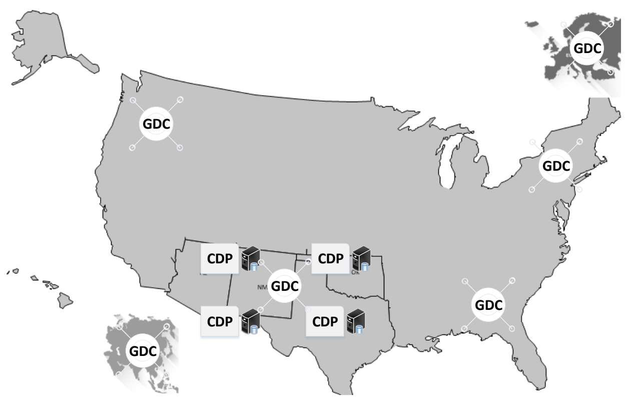

Figure 1: System Architecture.

The target applications are social media applications, such as Instagram or Twitter. When a user creates a data object (e.g., a post, a tweet, or a photo) on his connecting ZS, this ZS and its binding *GDC* are referred to as the *master* ZS (MZS) and the *master GDC* (MGDC) of that data object, respectively. The MZS in the MGDC always stores this object until it is deleted by its creating user. A data object may be replicated across different zones in the same *GDC* or different *GDCs*. If a zone replicates an object to store it locally, this zone is called a *replica* zone of this object (the corresponding ZS is a *replica* ZS). Otherwise, it is a *nonreplica* zone. We denote a *replica* ZS h by $\delta[h] = 1$. Thus, if ZS h is a *nonreplica* ZS, $\delta[h] = 0$. When there exists at least one replica zone for an object in a *GDC*, this *GDC* is called a *replica GDC* of this object. For an object, the *master* zone and the *master GDC* are always a *replica* zone and a *replica GDC*, respectively. We also denote a *replica GDC* d by $\alpha[d] = 1$; for a *nonreplica GDC* d, $\alpha[d] = 0$.

User access requests are passed to the local ZS. If that ZS does not store a replica, it invokes a remote access request to retrieve the data object from other ZSs with replicas in the same *GDC* or different *GDCs*. Apparently, as the number of replicas of a data object increases, the data availability and utilization become higher; but the operational cost of maintaining and creating more replicas also increases. Furthermore, if the number and placement of replicas are inappropriate, the problem of poor QoS arises. Our approach is based on access request workload prediction. The major components of CDP, shown in Figure 2, are explained:

- *Workload–based Predictor* (**WP**) : Performs an estimation of future access demand to the local ZS for the target data object. The prediction approach utilizes the auto-regressive integrated moving average model (ARIMA). For the MZS, the future demand estimate is directly passed to the local **CORP**, specified in the next step. For other ZSs, the demand estimate information will be transferred to the **CORP** of the MZS.
- *CORP Algorithm* (**CORP**) : We propose the **CORP** algorithm that performs cost optimization to determine the replica placement locations of the target data object based on the predicted future access demand from **WP**. **CORP** returns an optimal replica allocation pattern for the next time interval to minimize the total system cost (details in Section 4.1). For a target data object, **CORP** runs only in the corresponding MZS.
- *Cloud Data Provisioner* : 1) It receives access requests from *Access Control* and forwards them to the local ZS. 2) For the MZS of a target data object, it accepts an updated replica allocation pattern from **CORP** to figure out the migration process and to deploy suitable replicas on a periodic basis.

The function of **WP** designed in this paper realizes access workload prediction using the universal ARIMA time series forecasting approach [8] for the next time interval. ARIMA has been applied to the underlying workload fitting models in [1, 24], where the web-based workloads present strong autocorrelation.

At the end of each time interval, the data of historical workload traces during that time interval will be retrieved and transferred as the number of access requests by the Workload-based Predictor. The numbers of observed access requests for a sequence of time

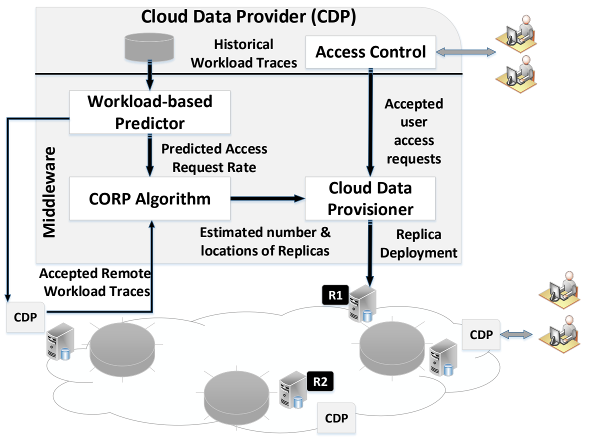
Cloud Data Provider (CDP)
Historical Workload Traces
Access Control
Workload-based Predictor
Accepted user access requests
Middleware
Predicted Access Request Rate
CORP Algorithm
Cloud Data Provisioner
Estimated number & locations of Replicas
Replica Deployment
Accepted Remote Workload Traces
CDP
R1
CDP
R2
CDP

Figure 2: Architecture for adaptive cloud data provider.

intervals compose a time series to fit the ARIMA model estimator. It is implemented as a cyclic shift operation, where the actual number of access requests at the last prediction cycle is added to the time series while the oldest value is discarded. Once the access requested time series is completed, the ARIMA model-fitting process is performed with the Box-Jenkins approach [8]. Based on this approach, the time series can be made to be stationary by differencing d times but remains nonstationary after differencing $d - 1$ times (called "integrated process").

Lags of the stationarized series in the forecasting model are called autoregressive terms. An autoregressive model (AR) is a linear regression of the current value of the series against one or more prior values of the series. The value of p lags in AR is called the order of the AR model. Lags of the forecasting errors are called moving-average terms. A moving average (MA) model is a linear regression of the current value of the series against the white noise or random shocks of one or more prior values of the series. The value of q lags in MA is called the order of the MA model. The Box-Jenkins ARIMA(p,d,q) model is a combination of the integrated AR and MA models For selecting the suitable predictors in ARIMA, Akaike's Information Criterion (AIC) is used to determine the orders of p and q. AIC is defined as

$$AIC = -2Log(L) + 2(p + q + k + 1) \tag{1}$$

where L is the likelihood of the data, k may be one or zero (k = 1 if $c \neq 0$ and $k = 0$ if $c = 0$). The value of c is the average of the changes between consecutive observations. The corrected AIC with n parameters for ARIMA can be written as

$$AICc = AIC + \frac{2(p + q + k + 1)(p + q + k + 2)}{T - p - q - k - 2} \tag{2}$$

By minimizing AICc, p and q can be obtained. Using the above method to determine the orders of p, d, and q for the ARIMA model, the historical workload data is fit to the model to be used in WP, of which the output outcome is the predicted access request rate for the next coming time interval. The length of the time interval can be flexibly adjusted for different applications. According to the evaluation in [17], an interval of dozens of minutes could suit the selected cloud provider well.

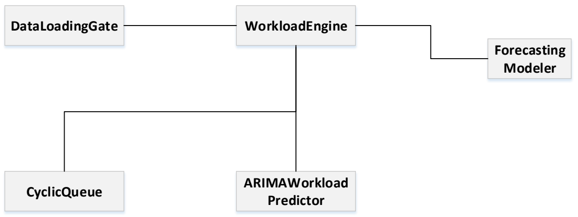

Figure 3: Diagram for ARIMA workload-based predictor.

3.2 ARIMA Configuration

Fig. 3 presents the diagram of the WP function. The *WokloadEngine* is the key component in the WP. The *DataLoadingGate* is responsible for adding workload data into the *CyclicQueue* and to trigger the change of the most recent workload traces. The *CyclicQueue*, which is composed of a queue object, is used to store the received workload data, which is modeled as a time series with a finite length. The length of the *CyclicQueue* is equal to the number of past time intervals influencing the prediction result. After *ARIMAWorkloadPredictor* creates a suitable ARIMA model, the workload prediction is accomplished by *ForecastingModeler*. *ARIMAWorkloadPredictor* and *ForecastingModeler* are realized by R packages [10], which is a statistical analysis program kit. They are not only capable of training a fitting model for a time series, but output a predicted value (the predicted number of access requests in the next time interval) fed back into the *WorkloadEngine*. The predicted value will be used in CORP algorithm we propose to allocate replicas with the cost optimization. This procedure is discussed in the later subsections.

3.3 Prediction Complexity

Several approaches can be applied for ARIMA model fitting. The fitting process used in this paper is based on the Hyndman-Khandakar algorithm [10]. It contains three major steps: (i) determine the order of differencing needed for stationarity; (ii) apply differentiation to the time series d times; (iii) decide the best fit model. The first step implements Kwiatkowski-Phillips-Schmidt-shin tests [12] used by R package. The time complexity of step (i) is O(n^2) (where n is the number of training time series instances from the workload data). The second step requires O(n) to perform d differentiations to the original time series values. The complete fitting process in the third step can be implemented with O(kn^2) for k iterations. Because k is a finite value (independent of n), the total time complexity in (iii) can be counted as O(n^2). Once the ARIMA model is ready, the prediction process for the next one time interval can be accomplished with O(max(p,q)), where p and q are the orders of AR and MA components, respectively.

4 Cost Optimization Replica Placement (CORP)

4.1 CORP Algorithm

We consider a time-slotted system, where the time is divided into slots with equal length time interval Δt and slot k $\in [1 \ldots T]$ is referred to as the discrete time period [kΔt,(k + 1)Δt]. Let $GDC_i = \{h_1,h_2,\ldots,h_{n_i}\}$ be a data storage cluster composed of n_i zones. Let $B = \{obj_1,obj_2,\ldots,obj_l\}$ be the replicated data object set of a zone h. The costs incurred in time slot k include the following five components in each zone h. (i) storage cost S defines the cost of maintaining a data object replica per unit size per unit time. (ii) network cost $O_{gdc}(h)$ defines the data object transfer cost between different *GDCs* per unit size. (iii) network cost $O_{in}(h)$ defines the data object transfer cost within the same *GDC* per unit size. (iv) *Get* transaction cost t_g denotes the cost of issuing an object *Get* request. (v) *Put* transaction cost t_p denotes the cost of issuing an object *Put* request.

After creating an object z in time slot k, each zone maintains a detailed access record for this object. The access record is represented as 3-tuple $\langle GN_k[z][h], PN_k[z][h], V_k[z] \rangle$. These are the number of *Gets* and *Puts*, and the size for the requests for object z in zone h in time slot k in the system. Let $AN_k[z][h]$ be the sum of $GN_k[z][h]$ and $PN_k[z][h]$. $AN_k[z]$ means the total access number of requesting object z in time slot k. $AN_k(z,d)$ denotes the total access request number in *GDC d* for an object in time slot k.

$$AN_k(z,d) = \sum_{zone\ h\ in\ d} AN_k[z][h] \tag{3}$$

Let $R_k[z]$ be the replica factor for object z in time slot k (the replica factor represents how many replicas of object z will be made). It hinges on the object access request traces per time slot for each zone. In order to determine the optimal placement of deploying object replicas so as to minimize the overall system costs, we introduce the following cost functions for a replicated data object/ or a comment (z) created in zone h of *GDC d* with size of V in time slot k. For ease of notation and without loss of generality, we ignore the indices k and z in the following.

• *Replication Cost.* Since we consider modern social media services, there are two types of data – objects (e.g., a photo) and comment messages (e.g., a replying comment) – that should be replicated. Once a user creates an object or replies a comment, the system may need to replicate this object or comment to other zones in the same *GDC d* or in other *GDCs*. Thus, the replication cost contains two different network transfer costs. The first is the cost of replicating an object between different zones, which reside in different *GDCs*. We define $r[d]$ as the number of replica zones for an object z in *GDC d*.

$$r[d] = \sum_{ZS\ l\ in\ d} \delta[l] \tag{4}$$

, where $\delta[l] = 1$ if a zone l is a *replica* zone; otherwise, $\delta[l] = 0$. Each ZS stores a Boolean vector $\delta[]$ for each object to track the registry of the object replicas distributed over the replica ZSs. By this bookkeeping implementation, the framework (each ZS) knows exactly which ZSs store which objects. The network transfer cost per unit data size for replicating object z from ZS h to these replica zones in *GDC* $d' \neq d$ is $\alpha[d']$ (O_{gdc} + ($r[d']$ − 1)× O_{in}). Note that here if $r[d'] > 0$, $\alpha[d'] = 1$; otherwise, $\alpha[d'] = 0$. We define the **F***irst* **N***etwork* **T***ransfer* **C***ost* per unit data size as

$$FNTC = \sum_{d' \neq d} \alpha[d']\{O_{gdc} + (r[d'] - 1) \times O_{in}\} \tag{5}$$

, where $\alpha[d'] = 1$ if *GDC* d' includes at least one *replica* zone; otherwise, $\alpha[d'] = 0$.

The second cost is the cost of replicating an object between zones within the same *GDC d*. Thus, the **S***econd* **N***etwork* **T***ransfer* **C***ost* per unit data size equals to

$$SNTC = (r[d] - 1) \times O_{in} \quad (6)$$

Since the data size of object z replicated is V, the total replication cost for creating this object (or replying to this comment) equals to

$$V \times (FNTC + SNTC) \quad (7)$$

• *Storage Cost*. The storage cost of an object z in zone h in time slot k with data size V equals to

$$SC(h) = S \times V \times \Delta t \times \delta[h] \quad (8)$$

• *Get Cost*. In our system, there are three different sub-cases for an object *Get* request, each of which corresponds to a *Get* cost function. Consider zone h issues $GN[h]$ *Get* requests for object z in time slot k. First, zone h stores a replica of object z. The *Get* cost equals to

$$GN[h] \times t_g \quad (9)$$

Second, zone h does not store a replica of object z, but some other ZSs store this object z in the same *GDC d*. The *Get* cost equals to

$$GN[h] \times (2t_g + V \times O_{in}) \quad (10)$$

Third, zone h does not store a replica of object z and no other zones store this object z in the same *GDC d*. The *Get* cost equals to

$$GN[h] \times (2t_g + V \times O_{gdc}) \quad (11)$$

• *Put Cost*. Similar to *Get* cost functions, there are also three *Put* cost functions, each of which corresponds to one of the above sub-cases. Consider zone h issues $PN[h]$ *Put* requests for object z in time slot k, each of which needs to retrieve object z and replicate comment message M with size of m. First, ZS h stores a replica of object z. The *Put* cost equals to

$$PN[h] \times (t_p + m \times (FNTC + SNTC)) \quad (12)$$

Second, ZS h does not store a replica of object z, but some other ZSs store this object z in the same *GDC d*. The *Put* cost equals to

$$PN[h] \times (t_p + t_g + V \times O_{in} + m \times (FNTC + SNTC)) \quad (13)$$

Third, ZS h does not store a replica of object z and no other zones store this object z in the same *GDC d*. The *Put* cost equals to

$$PN[h] \times (t_p + t_g + V \times O_{gdc} + m \times FNTC) \quad (14)$$

Besides, when ZS h in *GDC d* is a *replica* ZS of object z, it leads to additional *Put* cost due to comment update messages from other remote ZS.

$$\begin{aligned} RUM = \\ (t_p + m \times (\beta[d'] \times O_{gdc} + \neg\beta[d'] \times O_{in})) \times \sum_{d' \neq d} \sum_{zs\ i\ in\ d'} PN[i] \\ +(t_p + m \times O_{in}) \sum_{zs\ i \neq h\ in\ d} PN[i] \end{aligned} \quad (15)$$

, where $\beta[d'] = 0$, if $r[d'] > 1$. Otherwise, if $r[d'] = 1$, $\beta[d'] = 1$. Note that RUM is counted based on the point of view of incoming transmission. Therefore, RUM cannot be directly expressed by FNTC and SNTC.

• *Other Costs*. The cost of *Delete* to remove objects is free. *Post* transactions are considered as *Put* transactions. The *Post* cost is equal to the *Put* cost. *Copy* requests are implemented for replicating objects or object migration. A *Copy* request is composed of a *Get* transaction and a *Put* transaction. The cost of *Copy* equals to *Put* cost + *Get* cost.

Based on the above cost definitions, we can introduce three different cost functions as to an object z for a zone h in time slot k. First, zone h stores a replica of object z. The total zone cost for accessing the replica stored locally, defined as $SLC(h)$, is equal to

$$\begin{aligned} SLC(h) = S \times V \times \Delta t + GN[h] \times t_g \\ +PN[h] \times (t_p + m \times (FNTC + SNTC)) + RUM \end{aligned} \quad (16)$$

Second, zone h does not store a replica of object z, but some other ZSs store this object z in the same *GDC d*. The total zone cost for accessing a remote replica stored in a different ZS, defined as $RLC(h)$, is equal to

$$\begin{aligned} RLC(h) = GN[h] \times (2t_g + V \times O_{in}) + PN[h] \times \\ (t_p + t_g + V \times O_{in} + m \times (FNTC + SNTC)) \end{aligned} \quad (17)$$

Third, zone h does not store a replica of object z and no other zones store this object z in the same *GDC d*. The total zone cost for accessing a remote replica stored in a different *GDC*, defined as $RC(h)$, is equal to

$$\begin{aligned} RC(h) = GN[h] \times (2t_g + V \times O_{gdc}) + PN[h] \times \\ (t_p + t_g + V \times O_{gdc} + m \times FNTC) \end{aligned} \quad (18)$$

Therefore, the fundamental zone cost function $ZC_{z,k}$ of object z for ZS h in time slot k can be summed up with Storage Cost (SC), Transaction Cost (TC), and Network Transmission Cost (NTC). (For simplicity, we do not specify the subscripts z and k for SC, TC, NTC, and ZC.)

$$ZC(h) = SC(h) + TC(h) + NTC(h) \quad (19)$$

The transaction cost of ZS h can be represented as

$$\begin{aligned} TC(h) = GN[h] \times t_g + PN[h] \times t_p + \delta[h] \sum_{d} \sum_{zs\ i} PN[i] \times t_p + \\ \neg\delta[h] \times (GN[h] + PN[h]) \times t_g \end{aligned} \quad (20)$$

The network transmission cost of ZS h can be represented as

$$\begin{aligned} NTC(h) = GN[h] \times (\neg\delta[h] \times V \times (\alpha[d] \times O_{in} + \neg\alpha[d] \times O_{gdc})) \\ +PN[h] \times (\neg\delta[h] \times V \times (\alpha[d] \times O_{in} + \neg\alpha[d] \times O_{gdc}) \\ +m \times (FNTC + \alpha[d] \times SNTC)) \\ +(m \times (O_{in} + O_{gdc})) \times \sum_{d' \neq d} \sum_{zs\ i} PN[i] + (m \times O_{in}) \sum_{zs\ i \neq h\ in\ d} PN[i] \end{aligned} \quad (21)$$

For object z in time slot k, if zone h stores a replica of the object (i.e., $\delta[h] = 1$), $SC(h)$ equals to $SLC(h)$ in (16). If h is not a *replica* zone but its binding *GDC d* is a *replica GDC* (i.e., $\delta[h] = 0$ and $\alpha[d] = 1$), $SC(h)$ equals to $RLC(h)$ in (17). If h's binding *GDC d* is not a *replica GDC* (i.e., $\delta[h] = 0$ and $\alpha[d] = 0$), $SC(h)$ equals to $RC(h)$ in (18).

When the system assumes full replication, where all the data objects are replicated in all the zones, the total system cost for one object in a time slot equals to

$$\sum_{d} \sum_{zone\ h\ in\ d} SLC(h) \quad (22)$$

Although such full replication can reduce user access latency and maximize data availability, it is infeasible because of the immense size of the data stores and the large number of zones. Consider the system under partial replication. Assume that there are r zones with replicas of an object z in time slot k. The total system cost TSC for object z in time slot k is equal to

$$TSC = \sum_{d} \sum_{zone\ h\ in\ d} ZC(h) \tag{23}$$

, where $\sum_d \sum_{zone\ h\ in\ d} \delta[h] = r$.

4.2 Cost Optimization Problem

Given the above system cost model, our goal is to determine the optimal distribution of replica placement for objects so as to minimize the overall TSC for each time slot. This problem is defined as follows.

$$\forall h, \delta[h]^{opt} \leftarrow \underset{\delta[h], \forall h}{\operatorname{argmin}} TSC \tag{24}$$

s.t. (repeated for $\forall\ k \in [1 \dots$ T])
(a) $\delta[h = master] \leftarrow 1$
(b) $\sum\limits_{d} \sum\limits_{zone\ h\ in\ d} 1 = N$ (i.e., there are totally N zones in the system.)

In this cost optimization problem, GN, PN, V, and m for each object in a time slot are known and the argmin is over the set of $\delta[h]$ (h=1,$\cdots$,N), for which TSC attains the minimum value. Since O_{gdc} is much higher than O_{in}, it is required to determine whether each GDC is a *replica GDC* in the system. If a GDC is not selected as a *replica GDC*, none of the zones of this GDC will store replicas of the object. On the other hand, if a GDC is assigned as a *replica GDC*, then, the system needs to determine whether each ZS h in this *replica GDC* is required to store a *replica* of the object. Intuitively, this hinges on the comparison between $SLC(h)$ and $RLC(h)$, which is defined as

$$DIF(h) = SLC(h) - RLC(h) \tag{25}$$

Apparently, the value of $DIF(h)$ can specify whether it is optimal for a zone to store a *replica* of the object. However, when a zone is selected as a *replica* zone, the cost of replying comments to the associated object would increase. More precisely, $DIF(h)$ should be defined as

$$\begin{aligned} DIF(h) = S \times V \times \Delta t - AN[h] \times (V \times O_{in} + t_g) \\ +(m \times O_{in} + t_p) \times \sum_{d} \sum_{zone\ i \neq h} PN[i] \end{aligned} \tag{26}$$

Equation (26) reflects how a replica of the object in a zone affects the system cost. Without a replica, one zone will increase the network transfer cost $AN[h] \times V \times O_{in}$. With a replica, one zone will additionally bring the storage cost and the comment update cost. Formally, based on $DIF(h)$ in equation (26), one can determine whether zone h is required to store a replica for an object. In a *replica GDC*, if $DIF(h)$ is negative, zone h will be a *replica* zone. Otherwise, it is unnecessary to store a replica of the object in zone h. Similarly, we define a global $DIF_g(d)$ in the following equation to determine whether a GDC d is a *replica GDC* by comparing the cost with one replica to that without any replica. In addition to the cost of transferring update comments within a GDC, $DIF_g(d)$ considers that of updating comments between $GDCs$.

$$\begin{aligned} DIF_g(d) = S \times V \times \Delta t - AN(d) \times (V \times O_{gdc} + t_g)+ \\ (m \times O_{gdc} + t_p) \times \sum_{d' \neq d} \sum_{zone\ i\ in\ d'} PN[i]+ \\ (m \times O_{in} + t_p) \times \sum_{zone\ i \neq h\ in\ d} PN[i] \end{aligned} \tag{27}$$

In equation (27), the zone h having the maximum PN in GDC d is selected as a *replica* zone. Similar to $DIF(h)$, if $DIF_g(d)$ is negative, GDC d will be a *replica GDC*. Otherwise, it is not required to store a replica of the object in GDC d in order to lower the system cost.

Algorithm 1: CORP Algorithm based on the functions $DIF(h)$ and $DIF_g(d)$

```
Input : ∀h, $\delta_{k-1}[h]$, $GN_k[h]$, $PN_k[h]$, master ZS $H_h$, and
  master GDC $H_d$
Output: ∀h, $\delta_k[h]$
1  Initialize: $\delta_k \leftarrow \delta_{k-1}$;
2  for each zone $i \in H_d \wedge i \neq H_h$ do
3      SetRep(i);
4  for each GDC $d \neq H_d$ do
5      Select h with the maximum $AN_k[h]$ in d;
6      Calculate $DIF_g(d)$;
7      if $DIF_g(d) < 0 \wedge$ IsReplicaGDC(k−1,d) then
8          $\delta_k[h] = 1$;
9          for each host $i \neq h \in d$ do
10             SetRep(i);
11     else if $DIF_g(d) < 0 \wedge \neg$IsReplicaGDC(k−1,d) then
12         Migrate a new replica to site host h from another
             replica GDC;
13         $\delta_k[h] = 1$;
14         for each host $i \neq h \in d$ do
15             SetRep(i);
16     else if $DIF_g(d) > 0 \wedge$ IsReplicaGDC(k−1,d) then
17         for each host $i \in d$ do
18             $\delta_k[i]$=0;
19         if $DIF_g(d) - O_{gdc} \times V < 0$ then
20             Select h with the maximum $AN_k[h]$ in d;
21             $\delta_k[h]$=1;
SetRep(i)
22 Calculate DIF(i);
23 if $DIF(i) > 0 \wedge \delta_{k-1}[i] = 1$ then $\delta_k[i] = 0$;
24 if $DIF(i) < 0 \wedge \delta_{k-1}[i] = 0$
   $\wedge$ $DIF(i)+V\times O_{in} < 0$ then $\delta_k[i] = 1$ ;
AN(k,d):
25 return $\sum_{host\ i\ in\ d} (GN_k[i] + PN_k[i])$
IsReplicaGDC(k, d)
26 if ∃ host $i \in d$: $\delta_k[i]$ is true then
       return true
27 else
       return false
```

On the above grounds, we propose Cost Optimization Replica Placement Algorithm (**CORP**), as Algorithm 1, which calculates the optimized cost of the object replicas in each time slot $k \in [1 \dots T]$. The replication strategy of **CORP** (run at the end of time slot $k-1$) is summarized as follows. First, determine the replica placement in the *master GDC*. The *master* ZS must be a *replica* ZS. Then, one can determine whether each of the other zones in the *master GDC* is required to store a replica of object z. Second, the system

Table 1: Definition of symbols and parameters used in the model. The symbols on the four bottom rows are indicated for each object in each time slot (normally, they should be denoted as $\delta_k[z][h]$, $\alpha_k[z][d]$, $r_k[z][d]$, and $\beta_k[z][d]$).

Term	Meaning
D	A set of *GDCs*
S	The storage cost per unit size per unit time
$V(z)$	Size of data item z
Δt	Time slot interval
O_{gdc}	Out-network price between *GDC*s
O_{in}	Out-network price within a *GDC*
$GN_k[z][h]$	Number of *Gets* for object z from zone h in time slot k
$PN_k[z][h]$	Number of *Puts* for object z from zone h in time slot k
$V_k[z]$	Size of object z in time slot k
$AN_k[z][h]$	The sum of $GN_k[z][h]$ and $PN_k[z][h]$
$R_k[z]$	The number of replicas of object z in time slot k
t_g	*Get* transaction cost
t_p	*Put* transaction cost
m	Comment message size
$\delta[h]$	Binary replication factor for zone h
$\alpha[d]$	Binary replication factor for *GDC d*
$r[d]$	Number of replicas for an object in *GDC d*
$\beta[d]$	Binary factor for *GDC d*. If $r > 1$, β=0; if $r = 1$, β=1.

needs to determine whether each of the other *GDCs* is a *replica GDC*. Then, the system will determine whether each zone in every *replica GDC* is a *replica* zone based on equation (26). We make some notes about this instantiation of **CORP**. Lines (2)-(3) specify the replica placement for the object in the *master GDC*. Lines (4)-(21) determine the replica placement for the object in the other *GDCs*. When *GDC d* is a *replica GDC* in time slot $k-1$ and it is determined that d is still required to be a *replica GDC* in time slot k, lines (8)-(10) will specify the replica placement in *GDC d*. Lines (12)-(15) specify the case almost similar to the above one. However, *GDC d* is not a *replica GDC* in time slot $k-1$. It needs to migrate a new replica from another *GDC*. The output of **CORP** is the replica placement distribution $\delta_k[h]$ for each ZS h. When there exists at least one ZS h such that $\delta_k[h] \neq \delta_{k-1}[h]$, *Cloud Data Provisioner* in the MZS would implement the migration process. Otherwise, no replica needs to be migrated or deleted. Table 1 summarizes parameters and inputs to the model.

4.3 CORP + cache.

The regular CORP runs the ARIMA migration process by an equal time interval. Caching is commonly used to decrease network traffic and reduce network link utilization. In addition, for a target data object z, there is only home ZS that stores it in the first time interval, during which there are much more access requests from other ZSs over the network. This leads to ineffective network utilization. We extend CORP to CORP + cache, which could save object z temporarily in ZS h after retrieving it from a *replica* ZS of object z. This replica ZS is called the source ZS of the client cache ZS h. As with the replica bookkeeping mechanism in CORP, a source ZS hosts a Boolean vector to record its own client cache ZSs in + cache. Thus, source ZSs for an object can know exactly where cache ZSs are. Accordingly, ZS h does not need to retrieve it each time an access request is issued, even if ZS h is not a *replica* ZS. ZS h is called a *caching* ZS for object z. When a *replica* zone issues an update to object z, the update request is applied to not only all *replica* ZSs, but also the ZSs with the cache data of object z. However, the caching data of object z in ZS h can only be accessed by users connecting to ZS h. It cannot be retrieved by other ZSs. CORP + cache runs the same algorithm as CORP. However, the migration process of CORP + cache differs from that of CORP. Assume that ZS h is a *caching* ZS for object z in *GDC d* in time slot $k-1$. If ZS h is determined as a *replica* ZS, ZS h can straightforwardly become a *replica* zone. Then, if there exists other ZSs in *GDC d* that do not store a replica of object z in time slot $k-1$ and these ZSs are determined as *replica* ZSs in time slot k, ZS h will replicate object z to these ZSs. On the other hand, assume that ZS h caching object z in time slot $k-1$ does not need to be a *replica* ZS in time slot k. If there is no other *replica* ZS in time slot $k-1$ in the same GDC and some ZSs in the same GDC, which do not store object z, become *replica* ZS, then, ZS h needs to replicate object z to them. After that, the caching data of object z in ZS h is deleted.

5 Performance Evaluation

5.1 Experimental Setting

We evaluate the proposed **CORP** algorithms for replica placement of the data objects across *GDCs* with real traces of requests to the zone web servers from Twitter workload [13] and the CloudSim discrete event simulator [4]. These realistic traces contain a mixture of temporal and spatial information for each http request. The number of http requests received for each of the target data objects (e.g., photo images) is aggregated in 1800-secs intervals. By implementing our approaches on the Amazon cloud provider, it allows us to evaluate the cost-effectiveness of request transaction, data store, and network transmission, and to explore the impact of workload characteristics. We also propose a clairvoyant Optimal Placement Solution, based on the time slot system and object access patterns known in advance to evaluate CORP and CORP+cache.

Data Object Workload: Our work focuses on the data store framework on image-based sharing in social media networks, where applications have geographically dispersed users who put and get data, and fit straightforwardly into a key-value model. We use actual Twitter traces as a representation of the real world. *Put* to a timeline occurs when users post a tweet, retweet, or reply messages. We crawl the real Twitter traces as the evaluation input data. Since the Twitter traces do not contain information of reading the tweets (i.e., the records of *Gets*), we set five different ratios of *Put*/*Get* (P_{rate}: *Put* rate), where the patterns of *Gets* on the workloads follow Longtail distribution model [2]. The simulation workload contains several Tweet objects. The volume of each target tweet in the workload is 2 MB. The simulation is performed for a period of 20 days. The results for each object show that they have similar tendency.

The experiment has been performed via simulation using the CloudSim toolkit [4] to evaluate the proposed system. CloudSim is a JAVA-based toolkit that contains a discrete event simulator and classes that allow users to model distributed cloud environments, from providers and their system resources (e.g., physical machines and networking) to customers and access requests. CloudSim can be easily developed by extending the classes, with customized changes to the CloudSim core. We figure out our own classes for simulation of the proposed framework and model 9 *GDCs* in CloudSim simulator. Each *GDC* is composed of 4 zones. Each zone has only one ZS associated with 50GB storage space and corresponds to one (or a few) states in US or one country in Asia and in Europe. The price of the storage classes and network services are set for each *GDC* and each ZS based on Amazon Web Service (AWS) as of 2018.

5.2 Results and Discussion

The performance metrics we use are the performance in terms of cost and the cost improvement rates under varying P_{rate} of the proposed CORP and CORP+cache. In order to evaluate our proposed approach, we compare it to two different replication strategies. The first one is the standalone cache mode (+cache), where the home ZS is the only one *replica* ZS. The second one is the replication model with different numbers of *replica* GDC, where they are randomly pre-selected and each GDC includes at most one *replica* ZS. Cost is represented by the total system cost, which is composed of TC (Eqn. 20), NTC (Eqn. 21), and SC (Eqn. 19).

First, we use the term 'transaction' to denote both a *Put* transaction or a *Get* query transaction. Active and aggressive replication has the potential to provide a reduction in the number of distributed *Get* transactions at the cost of *Put* transactions. Lowering the number of transactions to find a data placement increases throughput significantly in cloud environments, while an increased number of transactions would lead to an over-utilization of the underlying systems. Thus, the total transaction cost (TC) is totally subject to the number of transactions. In our evaluations, we set different *Put* rates to generate different evaluation workloads. The lower the *Put* rate, the more the number of *Get* transactions should be included (i.e., the total number of transactions increases). Figure 4 presents the TCs of various replication strategies in different *Put* rates. Since CORP brings additional migration process, TC for CORP+cache is slightly higher than the standalone cache model or 2 pre-selected replicas+cache but much lower that others' TCs.

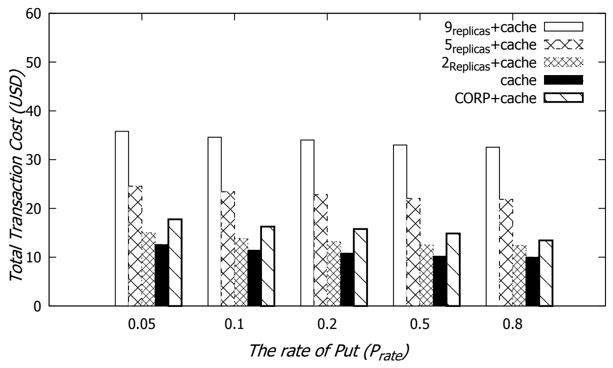

Figure 4: The Transaction Cost

Second, Figure 5 presents the NTC of CORP+cache in comparison with various replication strategies in different put rates. According to Eq. (21), NTC highly depends on the amount of data transmitted over the network. Thus, the smaller the NTC, the lower the network bandwidth consumption. Although NTC of CORP+cache is slightly higher that of the full GDC replication (each GDC contains one replica), it is much lower than others' NTCs.

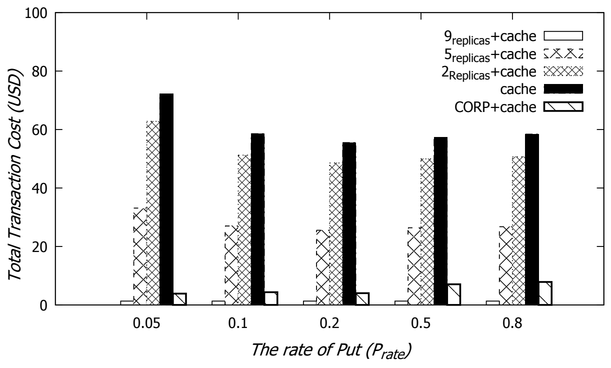

Figure 5: The Network Transmission Cost

Finally, the capacity of replica servers has a major impact on the performance of data replication strategy. Different data objects or applications require different storage services. Storage providers have to place large quantities of storage devices in order to offer uses good data storage services. Therefore, maximizing storage space utilization (SSU: the total available storage space in the entire system) becomes more and more important. The larger the available storage space, the more data objects the system can hold. Lower storage space occupation (SSO: the space size occupied by data objects) for a data object would increase SSU. The total storage space cost (SSC), which fully depends on equation (8), is highly proportional to SSO. Figure 6 shows the results of storage cost (SC) of CORP+cache in comparison with other alternatives. We notice that the SC of CORP+cache falls in between the SC of full GDC replication and the SC of standalone cache mode. This implies that CORP+cache is able to determine the proper number of replicas to decrease TC and NTC.

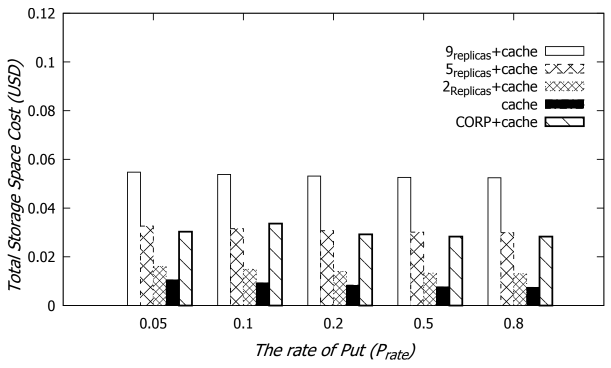

Figure 6: The Storage Space Cost

Figure 7 presents the total system costs (TSC) for different replication approaches. With caching, the more the number of *replica* GDCs, the lower the TSC. CORP+cache can further reduce TSC for all the workloads. Although CORP brings more migration costs for

Table 2: Cost improvement rates of different replication modes with respect to the standalone cache mode for different *Put* rates. (a) 2 static replicas with cache control (b) 5 static replicas with cache control (c) 9 static replicas with cache control (i.e., full GDC replication) (d) CORP+cache.

P_{rate}	0.05	0.1	0.2	0.5	0.8
(a)	8.24%	7.10%	6.96%	7.44%	7.81%
(b)	31.94%	27.96%	27.11%	28.17%	29.93%
(c)	56.16%	48.64%	46.70%	49.08%	50.44%
(d)	74.37%	70.46%	70.03%	67.39%	68.75%

replica placement step, it can reduce the storage cost and network cost by proactively placing data objects at proper locations.

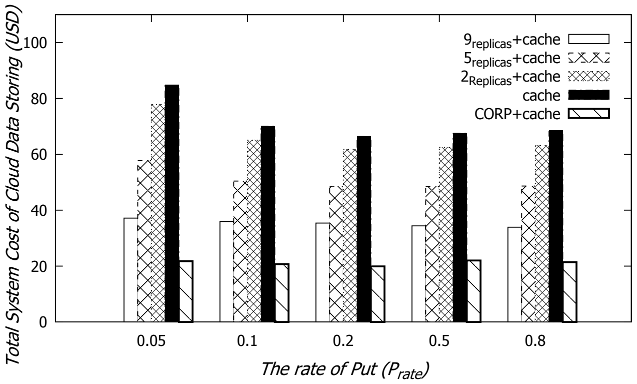

Figure 7: The total system cost.

Cost Improvement: We now investigate the effect of CORP+cache on cost improvement rate with respect to different replication modes, which is defined as.

$$\frac{cost(+cache) - Cost of Replication Mode}{cost(+cache)} \tag{28}$$

, where cost(+cache) means the total system cost of the standalone cache model. Table 2 shows the cost improvement rates of different replication modes with respect to the +cache mode for different *Put* rates. The cost improvement rate against +cache increases as more *replica* GDCs are loaded. As *Put* rate decreases, the cost improvement of CORP+cache becomes higher. It shows that CORP + cache is cost-effective for Get-intensive (lower *Put* rates) workloads like most social networks or cloud computing environments, due to efficient utilization of network communication resources.

Accuracy Analysis: The proposed method, ordinary CORP, runs on a time slot system. At runtime, CORP is constantly updated. When new access requests arrive in the current time slot, they are getting involved into the time series and the data in the oldest time slot is removed from the time series. However, when a data object is created, there is not enough data in the time series initially (i.e., the number of time slots is zero initially for each data object). Based on the training dataset, the demand for each time slot per ZS is predicted. The output of the prediction model is an integer value, which represents the access requests in the next time slot at a ZS. The predicted value is accompanied with two confidence ranges across the 80 and 95 percent bands. The accuracy of the prediction model is evaluated by different types of error metrics, namely, root mean square deviation (RMSD), normalized root mean square deviation (NRMSD), mean absolute deviation (MAD), and mean absolute percentage error (MAPE). The low 80% and high 80% indicate the limits for the 80% confidence interval for the prediction. The results for the 80% and 95% confidence intervals are given in Table 3 with P_{rate} = 0.05, where the average prediction accuracy (APA) is around 87 percent. With different *Put* rates, the corresponding APA values are approximately consistent with that of P_{rate} = 0.05.

Table 3: Access Number Prediction Accuracy by Various Error Metrics . RMSD : root mean square deviation. NRMSD : normalized root mean square deviation. MAD : mean absolute deviation. MAPE : mean absolute percentage error

Accuracy metric	Predicted	Low 80%	High 80%	Low 95%	High 95%
RMSD	346.71	476.12	571.42	800.01	931.14
NRMSD	0.16	0.31	0.33	0.37	0.41
MAD	131.4	379.81	450.11	534.66	794.52
MAPE	0.13	0.15	0.19	0.18	0.24

Table 4: Δ_{saving}: The cost improvement results for different *Put* rates show that caching has taken an important step to improve the total system costs.

P_{rate}	0.05	0.1	0.2	0.5	0.8
Δ_{saving}	91.33%	84.67%	74.93%	57.18%	48.70%

CORP+cache VS. CORP: We propose CORP+cache to improve the system costs. Thus, in this section we present experiments aimed at evaluating how the total costs are improved by CORP+cache against CORP. Table 4 presents the results of comparing CORP+cache to CORP for different put rates. Δ_{saving} is defined as

$$\frac{cost(CORP) - cost(CORP + cache)}{cost(CORP)} \tag{29}$$

Since the evaluation data come from the social network, each individual data object brings a lot of requests in the initial time slots. It can be observed that the results indicate that the lower the P_{rate} (Get-intensive), the better the cost saving rate (Δ_{saving}) is.

CORP+cache evaluation: In order to evaluate the effectiveness of our proposed approach, we also implemented the Optimal Placement Solution (OPT) based on the same time slot system. OPT knows the exact temporal and spatial data object access patterns. OPT can figure out the optimal object placement for each time slot. The cost effectiveness of CORP(+cache) is highly related to how precise the predicted access number is. OPT(+cache) totally follows up CORP(+cache) model and uses the real object access (*Put* and *Get*) numbers as the inputs in different time slots. Δ_{inc} is defined as

$$\frac{cost(CORP + cache) - cost(OPT + cache)}{cost(CORP + cache)} \tag{30}$$

Table 5 shows the comparisons between CORP+cache and OPT+cache for different *Put* rates. It is evident that CORP+cache only increases 6% ∼ 16% of total system cost compared to OPT+cache.

Table 5: Δ_{inc}: The performance evaluation of CORP+cache compared to OPT+cache.

P_{rate}	0.05	0.1	0.2	0.5	0.8
Δ_{inc}	16.77%	13.84%	10.74%	9.66%	6.23%

Table 6: Δ_{inc}: The performance evaluation of CORP compared to OPT in steady states.

P_{rate}	0.05	0.1	0.2	0.5	0.8
$\Delta_{inc'}$	1.89%	2.83%	1.53%	4.95%	3.26%

In order to measure the cost effectiveness of CORP in steady states (including enough training time slots), we also compare the cost of CORP to that of OPT without caching in steady states. Table 6 presents the cost increase rates ($\Delta_{inc'}$) of CORP compared to OPT for different *Put* rates. We notice that $\Delta_{inc'}$ rates are around 2% ~ 5%. $\Delta_{inc'}$ is defined as

$$\frac{cost(CORP) - cost(OPT)}{cost(CORP)} \tag{31}$$

For simplicity, we ignore the cost of the bookkeeping process based on the following. The size of a Boolean vector δ is O(N) bits, where N is the total number of zones in the system. The numbers of availability zones in most current modern cloud storage systems are from several dozens to several hundreds (i.e., AWS is composed of 69 zones globally). Compared to a common data object's size or a text comment's size in most social media applications, the storage cost of keeping δ is much smaller. On the other hand, consider the inventory management pricing in Amazon S3[1]. The total cost of maintaining the registry of objects is smaller than that of *SC*, *TC*, or *NTC* per object per time slot in one ZS. Therefore, we ignore the bookkeeping cost. In addition, *Get* bucket is not supported in CORP (+cache) to retrieve multiple objects by one access request. Therefore, we do not consider the cost of *List* requests.

Due to space limitations, this paper does not compare CORP + cache with other reactive replication approaches. Further, their frameworks are different and they do not make distinction between the 'between GDC' network cost and the 'within GDC' network cost. Although we do not present the detailed access latency analysis, the results show that the average access waiting time (AAWT) of CORP+cache is around 5 ms ~ 50 ms for different *Put* rates. However, the AAWT of the stand-alone cache mechanism and the full *GDC* replication are approximately 20 ms ~ 100 ms and 45 ms ~ 55 ms, respectively. The focus in this paper is on the trade-off between local data store and remote data access, whereas other realistic considerations and comparisons are left for future work.

6 Conclusions

We proposed the CORP and CORP+cache algorithms to realize a proactive provisioning replication model by using ARIMA for cloud datastores. Our approach employed a mixture of ARIMA and analytic schemes to analyze data access workload patterns in a predefined window length to predict workload distribution for the next time interval. The CDP can dynamically deploy required data replicas in the distributed storage system in responding to the predicted data access requests for the next interval. We presented an evaluation of the effect of the CORP+cache based on cost improvement via trace-driven CloudSim simulator toolkit and realistic workload traces from Twitter. Simulations showed that, with caching, as the number of *replica* GDCs increases, the TSC decreases. CORP+cache is capable of further reducing the TSC against the static replication mode with caching. The TSC of CORP + cache is around 70% ~ 75% lower than that of the standalone cache mode and 20% lower than that of the full GDC replication with cache control for different P_{rate}.

We also presented the cost effectiveness of +cache for CORP. The results show that +cache can improve CORP to effectively reduce the total system cost for the whole workload. In order to further evaluate CORP+cache, we compared it to the optimal placement solution (OPT+cache) in the same time slot system based on known temporal and spatial access patterns. Compared to OPT+cache, CORP+cache increases only 6% ~ 16% of TSC of OPT+cache. Furthermore, we also evaluated the cost effectiveness of CORP in steady states. By comparing CORP and OPT, simulation results show that CORP only increases 2% ~ 5% of the system cost of OPT. In other words, the TSC of CORP is highly close to that of OPT for the time slot system in a steady state.

The simulation results also showed that the TSC of CORP+cache is improved better in lower P_{rate}. It implies that CORP+cache is cost-effective for most social networks with Get-intensive workloads. This work can be further extended by employing consistency protocols to get high consistency among different replicas.

[1] https://docs.aws.amazon.com/AmazonS3/latest/API/API_InventoryConfiguration.html

References

[1] M. Arlitt and T. Jin. 2000. A Workload Characterization Study of the 1998 World Cup Web Site. *Netwrk. Mag. of Global Internetwkg.* 14, 3 (May 2000), 30–37. https://doi.org/10.1109/65.844498

[2] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. 2010. Finding a Needle in Haystack: Facebook's Photo Storage. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10).* USENIX Association, Berkeley, CA, USA, 47–60.

[3] R. N. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya. 2015. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Transactions on Cloud Computing* 3, 4 (Oct 2015), 449–458. https://doi.org/10.1109/TCC.2014.2350475

[4] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 2011. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Softw. Pract. Exper.* 41, 1 (Jan. 2011), 23–50. https://doi.org/10.1002/spe.995

[5] R. N. Calheiros, R. Ranjan, and R. Buyya. 2011. Virtual Machine Provisioning Based on Analytical Performance and QoS in Cloud Computing Environments. In *2011 International Conference on Parallel Processing*. 295–304. https://doi.org/10.1109/ICPP.2011.17

[6] E. Caron, F. Desprez, and A. Muresan. 2010. Forecasting for Grid and Cloud Computing On-Demand Resources Based on Pattern Matching. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science.* 456–463. https://doi.org/10.1109/CloudCom.2010.65

[7] Ruay-Shiung Chang, Hui-Ping Chang, and Yun-Ting Wang. 2008. A dynamic weighted data replication strategy in data grids. In *2008 IEEE/ACS International Conference on Computer Systems and Applications.* 414–421. https://doi.org/10.1109/AICCSA.2008.4493567

[8] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta Ljung. 2016. *Time Series Analysis: Forecasting and Control* (5 ed.). Wiley, Hoboken NJ. https://doi.org/10.2307/2284112

[9] Navneet Kaur Gill and Sarbjeet Singh. 2016. A Dynamic, Cost-aware, Optimized Data Replication Strategy for Heterogeneous Cloud Data Centers. *Future Gener. Comput. Syst.* 65, C (Dec. 2016), 10–32. https://doi.org/10.1016/j.future.2016.05.016

[10] Rob Hyndman and Yeasmin Khandakar. 2008. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software, Articles* 27, 3 (2008), 1–22. https://doi.org/10.18637/jss.v027.i03

[11] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. 2012. Empirical Prediction Models for Adaptive Resource Provisioning in the Cloud. *Future Gener. Comput. Syst.* 28, 1 (Jan. 2012), 155–162. https://doi.org/10.1016/j.future.2011.05.027

[12] Denis Kwiatkowski, Peter Phillips, Peter Schmidt, and Yongcheol Shin. 1992. Testing The Null Hypothesis of Stationarity Against The Alternative of A Unit Root. How Sure Are We That Economic Time Series Have Unit Root? *Journal of Econometrics* 54 (10 1992), 159–178. https://doi.org/10.1016/0304-4076(92)90104-Y

[13] Rui Li, Shengjie Wang, Hongbo Deng, Rui Wang, and Kevin Chen-Chuan Chang. 2012. Towards social user profiling: unified and discriminative influence model for inferring home locations. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '12*. 1023–1031. https://doi.org/10.1145/2339530.2339692

[14] Najme Mansouri and Mohammad M. Javidi. 2018. A new Prefetching-aware Data Replication to decrease access latency in cloud environment. *Journal of Systems and Software* 144 (2018), 197–215.

[15] Yaser Mansouri and Rajkumar Buyya. 2018. Dynamic replication and migration of data objects with hot-spot and cold-spot statuses across storage data centers. *J. Parallel and Distrib. Comput.* (12 2018). https://doi.org/10.1016/j.jpdc.2018.12.003

[16] Y. Mansouri, A. N. Toosi, and R. Buyya. 2019. Cost Optimization for Dynamic Replication and Migration of Data in Cloud Data Centers. *IEEE Transactions on Cloud Computing* 7, 3 (July 2019), 705–718. https://doi.org/10.1109/TCC.2017.2659728

[17] M. Mao, J. Li, and M. Humphrey. 2010. Cloud auto-scaling with deadline and budget constraints. In *2010 11th IEEE/ACM International Conference on Grid Computing*. 41–48. https://doi.org/10.1109/GRID.2010.5697966

[18] Rashedur M. Rahman, Ken Barker, and Reda Alhajj. 2005. Replica placement in data grid: considering utility and risk. *International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II* 1 (2005), 354–359 Vol. 1.

[19] Kavitha Ranganathan and Ian Foster. 2001. Identifying Dynamic Replication Strategies for a High-Performance Data Grid. In *Grid Computing — GRID 2001*, Craig A. Lee (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 75–86.

[20] P. N. Shankaranarayanan, A. Sivakumar, S. Rao, and M. Tawarmalani. 2014. Performance Sensitive Replication in Geo-distributed Cloud Datastores. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. 240–251. https://doi.org/10.1109/DSN.2014.34

[21] Matthew Sladescu, Alan Fekete, Kevin Lee, and Anna Liu. 2012. Event Aware Workload Prediction: A Study Using Auction Events. In *Web Information Systems Engineering - WISE 2012*. Springer Berlin Heidelberg, Berlin, Heidelberg, 368–381.

[22] Dawei Sun, Guiran Chang, Changsheng Miao, and Xingwei Wang. 2013. Analyzing, modeling and evaluating dynamic adaptive fault tolerance strategies in cloud computing environments. *The Journal of Supercomputing* 66, 1 (01 Oct 2013), 193–228. https://doi.org/10.1007/s11227-013-0898-7

[23] Da-Wei Sun, Gui-Ran Chang, Shang Gao, Li-Zhong Jin, and Xing-Wei Wang. 2012. Modeling a Dynamic Data Replication Strategy to Increase System Availability in Cloud Computing Environments. *Journal of Computer Science and Technology* 27 (03 2012). https://doi.org/10.1007/s11390-012-1221-4

[24] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. 2009. Wikipedia Workload Analysis for Decentralized Hosting. *Comput. Netw.* 53, 11 (July 2009), 1830–1845. https://doi.org/10.1016/j.comnet.2009.02.019

[25] Q. Wei, B. Veeravalli, B. Gong, L. Zeng, and D. Feng. 2010. CDRM: A Cost-Effective Dynamic Replication Management Scheme for Cloud Storage Cluster. In *2010 IEEE International Conference on Cluster Computing*. 188–196. https://doi.org/10.1109/CLUSTER.2010.24