Computers and Geosciences 112 (2018) 1–8

Contents lists available at ScienceDirect

Computers and Geosciences

journal homepage: www.elsevier.com/locate/cageo

Research paper

A dynamic replication management strategy in distributed GIS

Shaoming Pan [a], Lian Xiong [b], Zhengquan Xu [a,*], Yanwen Chong [a], Qingxiang Meng [c]

[a] State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, Hubei, China
[b] School of Communication and Information Engineering, Chongqing University of Posts and Telecommunications, Chongqing, China
[c] School of Remote Sensing and Information Engineering, Wuhan University, Wuhan, Hubei, China

ARTICLE INFO

Keywords:
Data replication
Replication management
Geospatial data
Data popularity
Distributed GIS

ABSTRACT

Replication strategy is one of effective solutions to meet the requirement of service response time by preparing data in advance to avoid the delay of reading data from disks. This paper presents a brand-new method to create copies considering the selection of replicas set, the number of copies for each replica and the placement strategy of all copies. First, the popularities of all data are computed considering both the historical access records and the timeliness of the records. Then, replica set can be selected based on their recent popularities. Also, an enhanced Q-value scheme is proposed to assign the number of copies for each replica. Finally, a reasonable copies placement strategy is designed to meet the requirement of load balance. In addition, we present several experiments that compare the proposed method with techniques that use other replication management strategies. The results show that the proposed model has better performance than other algorithms in all respects. Moreover, the experiments based on different parameters also demonstrated the effectiveness and adaptability of the proposed algorithm.

1. Introduction

Distributed geographic information system (GIS) can store large amounts of geospatial data (Pan et al., 2017) by using more storage nodes and also, can deal with more users by dispensing requests to different servers. Moreover, distributed GIS can improve the service response time by accessing data from local storage node through local server and also, distributed GIS can reduce the risk of a single point failure. Scheduling, organizing and managing all those data in distributed environment is the key way to achieve the above benefits (Rui et al., 2017) and most of such applications are adopting this management approach (Boulos, 2005; Bell et al., 2007).

The performance of distributed system is decided by two aspects: the load balance of whole system and the service response time. One of primary method to realize load balance in distributed GIS is data placement strategy. The Access Pattern-based Storage Algorithm (APSA) (Pan et al., 2015) mines the relationship of all data and then separately stores the related data into different storage nodes so as to get a higher whole parallel access probability and avoid to access data from remote nodes. Also, Dynamic Computation Correlation based Placement strategyaaa (DCCP) (Wang et al., 2016) places the dataset considering both the I/O load and the capacity load according to their computation correlations.

Since the relationship among all data will change continuously with the running of system (Rui et al., 2012), mining their correlations in advance based on their historical relationships cannot dynamically track their characteristics. Furthermore, there are huge amount of data which will be accessed and stored, dynamically adjusting data among all storage nodes will lead to a storm of data transferring and reduce the availability of network bandwidth (Chervenak et al., 2009).

To address this problem, data replication is an important optimization strategy to keep some hotspot data as replicas in high-speed cache so as to avoid getting data from remote storage nodes repeatedly, and which can significantly improve system performance (Rui et al., 2017). In fact, some researches show that data replication strategy is the only way to improve the performance of load balance (Amjad et al., 2012) and service response time (Xiong et al., 2016). Obviously, not all geospatial data need to be used to create copies due to the massive dataset in GIS and the minimal access frequency of part of geospatial data. Thus, in order to clearly describe our method, we define that *replica* is a sub-dataset of all geospatial data and each replica has one or more *copies* which are stored into different places (Manifestly, all geospatial data will be split into two sub datasets which are the replica and the others, where only the data belongs to replica sub dataset can create copies so as to limit the total amount of copies to save storage space).

* Corresponding author.
E-mail address: xuzq@whu.edu.cn (Z. Xu).

https://doi.org/10.1016/j.cageo.2017.11.017
Received 14 March 2017; Received in revised form 9 October 2017; Accepted 22 November 2017
Available online 2 December 2017

For centralized system, there are only one cache buffer and each replica has only one copy. Examples of such strategies include the Popularity model, Markov Chain model and the Passive model. The popularity model (Shi et al., 2005) calculates the popularity of all data based on their history access records and then selects those data with higher probabilities as replicas. Markov Chain model uses a basic Markov Chain model (Li et al., 2010) or some improved Markov Chain model to cache optimum data as replicas (Rui et al., 2012). The passive model just keeps the data being accessed as replicas, such as Least Frequently Used (LFU) and Least Recently Used (LRU).

Unfortunately, unlike centralized system, data replication in distributed system must consider two more key problems: how many copies should be produced for each replica and where all copies will be placed to meet the requirements of load balance? Furthermore, dynamically tracking the characteristics of all data with the running of system is also the key of the solutions.

2. Related work

Bandwidth Hierarchy based Replication (BHR) (Park et al., 2004) and the modified BHR (Sashi and Thanamani, 2011) are two typical methods which use dynamic replication strategy to reduce data access latency and to increase resource utilization rate. The main idea of BHR is to keep the required data in the same region as much as possible to reduce the external-schedule time. BHR selects the best site to keep replicas based on all nodes' bandwidth distribution.

Hierarchical Replication Strategy (HRS) (Chang et al., 2007) is similar to BHR and is also used in the data grids. HRS aims to get an enough higher hit ratio of required data within a region so as to reduce the internal-schedule time. Rather than BHR which computes the popularities of all replicas based on the cluster level, HRS computes the popularities of all replicas at site level and can more accurately define the real characteristics of system.

Due to the special requirements and characteristics of GIS, Global User-Driven Caching method (GUDC) (Pan et al., 2017) is one of the typical data caching modes for GIS. GUDC computes all data relationship based on their historical access records and then compares the conditional prefetching probability from uncached datasets so as to select some data as replicas and then, store them into cache buffer. Meanwhile, GUDC uses the same method to delete the replicas from cache buffer to save space.

Collaboration Model for Replicas in Distributed Caching (CMRDC) (Rui et al., 2017) designed three aspects of data replication strategy for distributed GIS which are replica dataset selection, the number of copies for each replicas and replicas placement. CMRDC uses a steady-state cache hit ratio to estimate the size of replica dataset and then computes the popularities of each data and selects a given number of the higher popularity data as replica datasets.

Obviously, those methods mainly designed for GIS are very effective and provide some far better strategies to improve system performance in distributed GIS. But their effectiveness depends on two preconditions: the enough large of historical access records and the enough stability of their relationships.

Unfortunately, as a typical dynamic system, the relationships or popularities of all data are changed continuously (Rui et al., 2012) and thus mining the patterns based on historical access records to guide the replication strategy may not always work. Furthermore, we can not obtain a large enough historical access records when the system has just started.

To address above mentioned problems, the proposed method in this article, i.e., Access-related Dynamic Data Replication strategy (ADDR), also considers the three aspects of data replication strategy for distributed GIS. ADDR only uses the recent limited records to mine data relationships. Moreover, each record has a different weight based on their freshness (Chang and Chang, 2008) and thus, we can track the changes of relationships closely. Then, an enhanced Q-value scheme (Burghes et al., 1982) which considers both the fairness and capabilities of all nodes is implemented to get the number of copies for each replica and to guide the replicas placement among all nodes. Also, the replicas will be adjusted dynamically.

3. Replication management strategies

3.1. System architecture

Fig. 1 gives a typical cooperation model for replication in distributed GIS, where raster data are split into different resolution ratio image files due to pyramid model and each of them are separately stored as a single geospatial data. All geospatial data are distributed stored in M clusters and each cluster has storage node and server. Each server holds one high speed cache buffer. The server and storage node in the same cluster are connected by LAN and clusters are connected through internet. In distributed GIS, users' access requests are distributed by load distributor to different servers based on the length of access queue and the cached data set.

Since there exist massive geospatial data stored in distributed GIS and part of them only have lower access frequency (Pan et al., 2017), only part of geospatial data need to be selected as a female parent to create copies so as to save storage space. Thus, based on the above analysis, ADDR must solve three problems: selecting the appropriate sub-geospatial dataset as replicas set, allotting storage space for each replica (i.e., deciding the number of copies for each replica) and placing all copies into suitable locations.

In this paper, we proposed a complete solution to meet the above three requirements and we explained these three mechanisms in detail in the following.

3.2. Replica set selection algorithm

Firstly, let $D = \{d_1, d_2, \cdots, d_N\}$ be the set of all data stored in distributed GIS, where N is the size of dataset. Each element in D is labeled with a natural number coding scheme and the size of data d_m is s_m ($m \in [1\ \ N]$). Assuming the historical access records will be chronologically recorded and which can be denoted as $H = (\{a_1, t_1\}, \{a_2, t_2\}, \cdots, \{a_K, t_K\})$, where K is the total number of accessing to all data, $a_j (j \in [1\ \ K])$ denotes the label of j-th accessed data and $t_j (j \in [1\ \ K])$ denotes the time of j-th accessing.

Dividing historical access records into several sub-parts $H = (H_1, H_2, \cdots, H_L)$ based on a given time interval τ, where L is the total number of sub-parts and $H_i = (\{a_{1_i}, t_{1_i}\}, \{a_{2_i}, t_{2_i}\}, \cdots, \{a_{K_i}, t_{K_i}\})(i \in [1\ \ L])$ satisfy the conditions: $t_{K_i} - t_{1_i} \le \tau$ and $t_{1_{(i+1)}} - t_{1_i} > \tau$, where K_i is the total number of the accessing to all data in i-th sub-part. For $\forall H_i \in H$, let $\lambda_{m,i}$ denotes the total access count of data $d_m (m \in [1\ \ N])$ during the time of i-th sub-part $[t_{1_i}\ \ t_{K_i}](i \in [1\ \ L])$, then the popularity of data d_m can be stated as follows:

$$\xi_m = \sum_{i=1}^{L} \frac{\lambda_{m,i} \times f(L-i)}{K_i} \tag{1}$$

where $f(x)$ is a decay function. Therefore,

$$\overline{\xi} = \sum_{m=1}^{N} \xi_m \Big/ N \tag{2}$$

indicates the average popularity and then, the data which the popularity is higher than the average popularity can be selected as the member of replica set. Furthermore, let $C = \{c_1, c_2, \cdots, c_M\}$ be the set of all server in distributed GIS and M is the total number of servers. The bandwidth of server c_i is b_i and the high-speed cache buffer size is μ_i ($i \in [1\ \ M]$) and so, τ can be estimated:

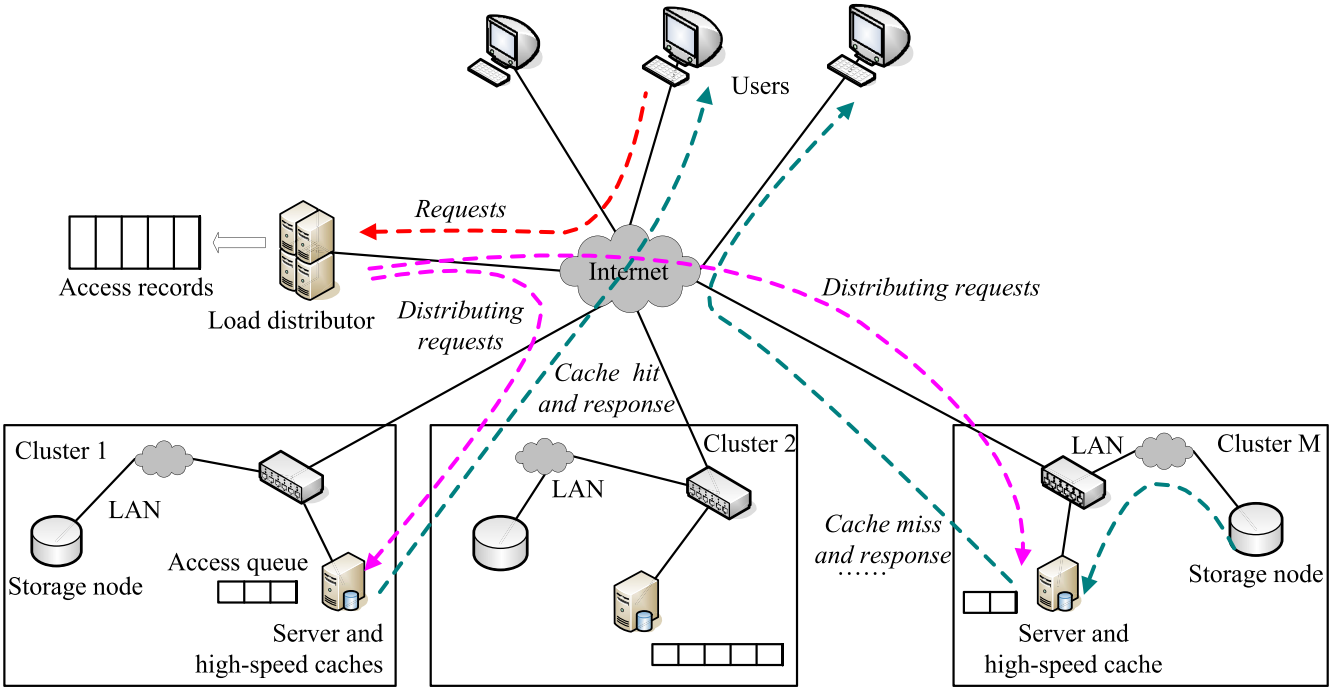

Fig. 1. Cooperation model for replication in distributed GIS.

$$\tau = \left(\sum_{i=1}^{K} \frac{M \times s_{a_i} \times f(K-i)}{B}\right) \Big/ \sum_{i=1}^{K} f(K-i) \tag{3}$$

where $B = \sum_{i=1}^{M} b_i$ presents the total bandwidth of distributed GIS and Eq. (3) gives the average service time and obviously, newer of access information has greater contributions to estimate τ.

Since newer of access information has larger influence to popularities or estimate the time interval (Chang and Chang, 2008), set $f(x) = e^{-\left(x^2/2\sigma^2\right)}$, where σ is decay coefficient. Based on above description, selecting the different decay coefficient will use different number of history access information as well as each access information will have different weight and thus, some typical selections can be given in Fig. 2, where the decay coefficient σ vary from 5 to 25 and thus, only the latest 10–70 sub-parts will be used to compute the popularity of each data based on Eq. (1).

3.3. Number of copies selection algorithm

Considering each data has different popularity, some of them may be accessed simultaneously by lots of users and so, each replica may have different number of copies. Obviously, only limited storage space can be allocated for the copies of all replicas and so, deciding the number of copies for each replica is a typical limited seats assignment problem which can be solved by Q-value scheme (Burghes et al., 1982), where the number of copies of each replica is the seats of each class and the total seats of all class is the total storage space for the copies of all replicas. For the simplicity of description, we also denote $D = \{d_1, d_2, \cdots, d_N\}$ as the obtained replica set in the previous sections and then, Q-value function can be stated as follows:

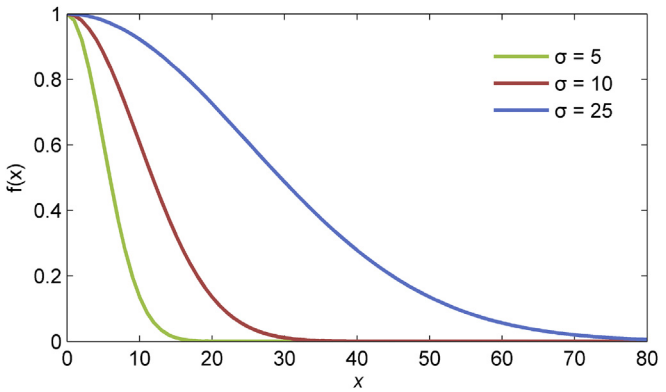

Fig. 2. Decay curves based on different decay coefficient.

$$Q_m = \frac{\xi_m^2}{R_m \times (R_m + 1)} \tag{4}$$

where Q_m is the Q-value of data d_m and R_m is the number of copies of data $d_m (m \in [1\ N])$.

But using Q-value scheme to solve the number of copies selection problem remains two disadvantages in distributed GIS: 1) when Q-value algorithm procedure is just beginning, R_m is zero and Q_m can not be computed based on Eq. (4). So, Q-value scheme assign one copy for each replica for the first step at the beginning and which will not work when the high-speed cache buffer is not large enough to hold all replicas; 2) if there is very big difference of popularity (i.e., the number of seats in each class) among all replicas, some replicas will occupy all high-speed cache buffer and all the other replicas will never have any one copy. Since there are limited servers in distributed GIS, assigning copies for each replica more than the number of server is a waste of cache buffer and may also be unfair. In order to address above problems, an enhanced Q-value algorithm is proposed which uses modified J-value to assign seats at the beginning (Michael and Caulfield, 2010) and try to assign seats to all replicas. And also, some replicas which have very high popularities will be stopped to assign more copies even when they have a larger J-value or Q-value based on the number limitation of servers. Set

$$J_m = \left\lfloor \frac{100 \times \xi_m}{R_m + 1} \right\rfloor \tag{5}$$

as J-value of data d_m, where the symbol $\lfloor . \rfloor$ can set same J-value to the

data which have similar popularities so as to rapidly assign copy to all replicas. Thus, an enhanced Q-value procedure which has several steps can be stated as follows:

1) Denote $C_{size} = \sum_{i=1}^{M} \mu_i$ as the total high-speed buffer size and C_{left} as the surplus high-speed buffer size; Set $R_m = 0 (m \in [1\ \ N])$ and $C_{left} = C_{size}$;
2) Find the largest popularity from all data and assume it as ξ_{x_1}, set $R_{x_1} = 1$ and $C_{left} = C_{left} - s_{x_1}$;
3) Let $J = \{J_1, J_2, \cdots, J_N\}$. Find the largest element from J and if we have more than one element, selecting the one which the number of copies is the smallest. Assuming the selected element is J_{x_2}, then set $R_{x_2} = R_{x_2} + 1$ and $C_{left} = C_{left} - s_{x_2}$. If $C_{left} < \min(s_m)$, exit the procedure;
4) Repeat steps 3–4 until there are two replicas which the number of copies is not zero and at least one of them is 1. Then the replica set can be divided into two sub-set $\tilde{D}_1$ and $\tilde{D}_2$, where each element in $\tilde{D}_1$ has at least one copy and each element in $\tilde{D}_2$ has no copy;
5) Recalculate J;
6) For $\forall d_i \in \tilde{D}_1$ and $\forall d_j \in \tilde{D}_2$, if $J_i > J_j$ then schedule Q-value procedure to assign one seat to one element in $\tilde{D}_1$; Otherwise, find the largest element from J and assume it as J_{x_3}, where $d_{x_3} \in \tilde{D}_2$, then set $R_{x_3} = R_{x_3} + 1$ and $C_{left} = C_{left} - s_{x_3}$, move d_{x_3} to $\tilde{D}_1$. If $C_{left} < \min(s_m)$, exit the procedure;
7) If $R_m > M$, set $C_{left} = C_{left} + (R_m - M)s_m$, $R_m = M$ to save cache space and set $\xi_m = 0$ to stop to assign more copies for d_m;
8) Repeat steps 5–8 until $\tilde{D}_2 = \Phi$. Obviously, we have $\tilde{D}_1 = D$ and each replica has at least one copy;
9) Schedule Q-value procedure to assign surplus seats to elements in $\tilde{D}_1$;
10) If $R_m > M$, set $C_{left} = C_{left} + (R_m - M)s_m$, $R_m = M$ to save cache space and set $\xi_m = 0$ to stop to assign more copies for d_m; Repeat steps 9–10 until $C_{left} < \min(s_m)$.

Based on steps 3 and 6, the enhanced Q-value algorithm can more effectively assign the limited cache buffer to some very hot spot data rather than traditional Q-value scheme which simply assign one copy for each replica. Also, based on the steps 7 and 10, the enhanced Q-value algorithm can avoid all seats to be occupied by a few data which have very high popularities.

As shown in above steps, in step 1 initial value to all parameters are assigned and in steps 2–5 copies for all replicas are assigned one by one based on their J-value until copies for at least two replicas are assigned. Where first copy for the most popular replica is assigned in step 2 and more copies are assigned based on their J-value in steps 3–4. Each time one copy is created, the J-value of all replicas have to be updated based on Eq. (5) due to their number of copies are changed and this process is individually shown in step 5. Obviously, the more of copies indicate a smaller of J-value and Q-value when their popularities are same and so, each replica is assigned at least one copy in steps 6–8. After that the left copies are assigned in steps 9–10 based on Q-value algorithm until all storage space is occupied or each replica has assigned M copies. In order to clearly describe this algorithm process, an example is given as follows (Table 1), where the access sequence is "GGGGEAFBCCDABCDDDCDDDABCDDDCDDDG" (Madison and Batson, 1976) and only C, D, G will be selected as the elements of replica set based on the algorithm in section 3.2. In addition, we assume $M = 2$, $\mu_i = 2$ (data unit) and then we have $C_{size} = \sum_{i=1}^{M} \mu_i = 4$ (data unit).

Table 1
A simple example of copies selection procedure.

Steps	Process	Middle results
1	Initialize parameters	R(C, D, G)=(0 0 0); ξ(C, D, G)=(0.194 0.387 0.161); $C_{left} = C_{size} = 4$.
2	Create first copy	R(C, D, G)=(0 1 0); J(C, D, G)=(19 19 16); $C_{left} = 3$.
3	Create second copy	R(C, D, G)=(1 1 0); $C_{left} = 2$; $\tilde{D}_1 = \{C, D\}$; $\tilde{D}_2 = \{G\}$.
4	All replicas assigned at least one copy	R(C, D, G)=(1 1 1); J(C, D, G)=(9 19 16);$C_{left} = 1$; $\tilde{D}_1 = \{C, D, G\}$; $\tilde{D}_2 = \Phi$.
5	Calculate Q-value	Q(C, D, G)=(0.018 0.075 0.013);
6	Assign the last seat to data D which has the largest Q value	R(C, D, G)=(1 2 1); $C_{left} = 0$;
7	Exit procedure	Here C, G have one copy and D has two copies.

$$\psi(m,n) = \sum_{i=1}^{L} \frac{\lambda_{m,i} \times \lambda_{n,i} \times f(L-i)}{K_i} \tag{6}$$

Let $P_i = (p_i(m,n))_{N\times N}$ is a copies placement strategy for i-th server. If data d_m and d_n have a copy stored in high-speed cache buffer of i-th server, then $p_i(m,n) = 1$, otherwise $p_i(m,n) = 0$. Based on above copies placement strategy, we can obtain total correlations of all replicas stored in high-speed cache buffer of i-th server:

$$\psi(P_i) = \frac{\sum_{m=1}^{N}\sum_{n=1}^{N} \psi(m,n) \times p_i(m,n)}{\sum_{m=1}^{N}\sum_{n=1}^{N} p_i(m,n)} = \frac{P_i \cdot \Pi}{\|P_i\|_0} \tag{7}$$

Where $\Pi = (\psi(m,n))_{N\times N}$ is correlations matrix of all replicas. Considering different server has different service capability, the normalized total correlations for i-th server is:

$$\overline{\psi}(P_i) = \frac{\psi(p_i)}{b_i} = \frac{P_i \cdot \Pi}{b_i \times \|P_i\|_0} \tag{8}$$

Since the high-speed cache buffer of each server is limited, the placement strategy based on Eq. (8) must satisfy the following condition:

$$s.t. \quad \Gamma(P_i) = \sum_{m=1}^{N}\sum_{n=1}^{N} p_i(m,n) \times (s_m + s_n) \le \mu_i \tag{9}$$

Similarly, we can get the total correlations of all servers and their copies number conditions.

$$\overline{\psi}(P) = \sum_{i=1}^{M} \frac{P_i \cdot \Pi}{b_i \times \|P_i\|_0} \tag{10}$$

$$s.t. \quad \Re(P_i, m) = \sum_{i=1}^{M}\sum_{j=1}^{N} p_i(m,j) \le R_m \quad 1 \le m \le N \tag{11}$$

Since the aim of load balance is to distribute the requests averagely into all servers, the problem of obtaining an optimal copies placement strategy for all replicas can be transfer to find an optimal solution so as to get a uniform distribution of popularities, and which can be expressed as follows:

$$\begin{aligned} P^* &= \arg\min_{P} \left(\sum_{i=1}^{M} |\overline{\psi}(P) - \overline{\psi}(p_i)| \right) \\ s.t. \quad & \Gamma(P_i) \le \mu_i \\ & \Re(P_i, m) \le R_m \quad 1 \le m \le N, 1 \le i \le M \end{aligned} \tag{12}$$

Due to the massive data set in distributed GIS, we can also use a locally approaching search algorithm to determine a reasonable solution based on correlations matrix Π (Pan et al., 2015) which was developed

3.4. Copies placement strategy

In order to realize load balance among all servers, copies placement strategy should place all copies of replicas which have higher probability simultaneously into different nodes. Thus, the correlations of two replicas d_m and d_n $(m, n \in [1\ \ N])$ can be given firstly as follows:

from CM ordering algorithm (Gibbs et al., 1976).

In a word, by the above three algorithms, ADDR can decide which data can be selected as replicas set based on their recent popularities and also, the number of copies for each replica can be produced using an enhanced Q-value scheme even though the cache space is extremely limited or the popularity distribution is extremely uneven. Finally, a reasonable copies placement strategy can be found for the requirement of load balance.

4. Simulations and results

4.1. Simulation design

OptorSim (Zhang et al., 2016) was developed to mimic any network topology and replication strategy which is widely used as the simulation platform by lots of researchers (Chang et al., 2007; Wang et al., 2016). In this paper, we also use OptorSim to evaluate the performance of different strategies for dynamic replication management algorithms, where the simulation parameters are shown in Table 2.

To fairly evaluate our strategy comparing with the other dynamic replication strategies which are mentioned in above, the Cache-Hit Ratio (CHR) can be used as the metric. Besides, the Load Imbalance Ratio (LIR) and Cache Replacement Ratio (CRR) will also be checked in the simulations, where LIR is the ratio between the difference of server load and average load. Besides, we set $\sigma = 5$ in the simulations. Meanwhile, we assume that users' access behaviors satisfy Zipf law (Shi et al., 2005; Newman, 2005) and the distribution parameter α is 0.750 (Rui et al., 2017). Moreover, all of the experiments have been repeated about 6 times and all the reported results are averages.

4.2. Simulation results

4.2.1. Experiments using different replication algorithms

In this experiment, CMRDC and GUDC train their models using a historical access records and then, prepare replications in high-speed cache in advance before system is started to operate, and no other algorithms have pre-caching strategy. All algorithms will get replicas set, create copies for all replicas, place all copies to all servers and replace copies from cache based on their own strategy during the simulations. Fig. 3 shows CHRs for all of the algorithms using 6 servers and caching 1% data in each server.

As shown in Fig. 3, ADDR, BHR and HRS can get almost the same high CHRs when system just begins. Although the performance of ADDR is worse than BHR at the very beginning of system operation, ADDR can get a high CHR once enough access records have been collected and also, ADDR can quickly track the changes of users' behaviors and none of other algorithms can dynamically track this change and respond quickly. Thus, ADDR can get a better CHR performance than the other algorithms even if the experiment lasts longer, no matter users' access behaviors change or not.

Moreover, the results were analyzed by Analysis of Variance (ANOVA) and we have the F-value $\gg 1$ and the P-value < 0.002 which indicate that the probability of the experimental results affected by random initialization is less than 0.2% and the simulation results

Table 2
Simulation parameters.

Parameter	Value
Number of cluster (M)	2–20
Number of nodes in each cluster	1
High-speed space at each node (μ_i)	1%–10% of data set
Size of single data (s_i)	44 KB
Size of data set (N)	10,000–100,000
Connectivity bandwidth (b_i)	100Mbps
Number of jobs (users)	300
Number of data accessed (K) by all jobs	3,000,000–30,000,000

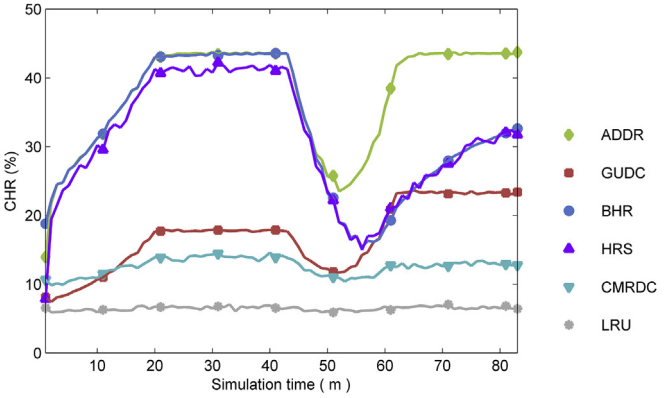

Fig. 3. Comparative CHRs obtained from all algorithms during simulation period.

are efficient.

To check the performance of all of the algorithms with different cache buffer size, an experiment is conducted with 6 servers and between 1% and 10% of cached data. The results are shown in Fig. 4 (CHRs), Fig. 5 (CRRs) and Fig. 6 (LIRs).

As shown in Fig. 4, the performance of CMRDC and GUDC does not change when cache buffer size is larger than 6%, whereas the performances of all of other algorithms improve with increasing cache buffer size. CMRDC and GUDC only select part of data as copies to save high-speed cache space, and this strategy will lead to a waste of part high-speed cache space when the high-speed cache space is large enough. But the resulting benefits are that they can get a lower CRRs (Fig. 5) and LIRs (Fig. 6) by reducing the cache usage.

Throughout the experiment, ADDR can get a better average CHR performance higher than others about 10.2%–96.8% as well as a better average CRR performance lower than most of the other algorithms about 94%, except for BHR, which can get a as lower CRRs as ADDR and their difference of average CRRs are less than 0.4%. Although the performance of LIRs fluctuates with increasing cache buffer size, all of the algorithms can get very low LIRs and all of them are less than 0.4% and the difference can be ignored. This occurs because we design the whole simulations based on load balancing data scheduling strategy as mentioned in Fig. 1.

In addition, an experiment was conducted in which the number of servers varies from 2 to 20 and caching 1% data in each server. The results are shown in Fig. 7 (CHRs), Fig. 8 (CRRs) and Fig. 9 (LIRs).

As shown in Fig. 7, the CHR performance of all algorithms improves with an increasing number of servers. This occurs because more of the servers indicate larger of the total cache buffer space (the total cache buffer size is $M \times 1\%$) and a larger cache buffer space indicates more of copies can be produced so as greater possibility of a cache hit can be

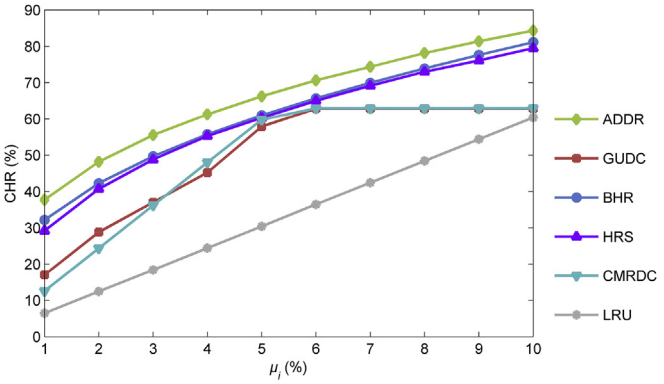

Fig. 4. Comparative CHRs obtained from all algorithms with different cache buffer size.

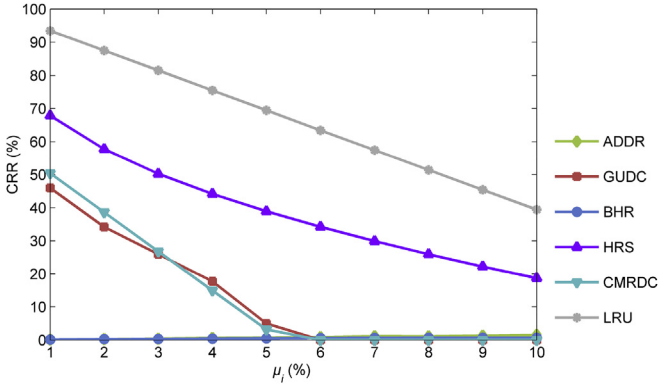

Fig. 5. Comparative CRRs obtained from all algorithms with different cache buffer size.

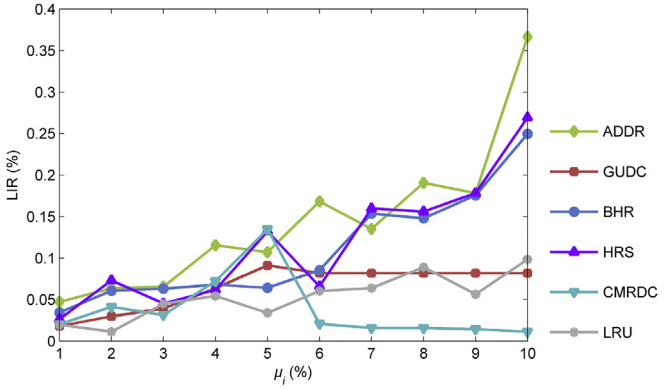

Fig. 6. Comparative LIRs obtained from all algorithms with different cache buffer size.

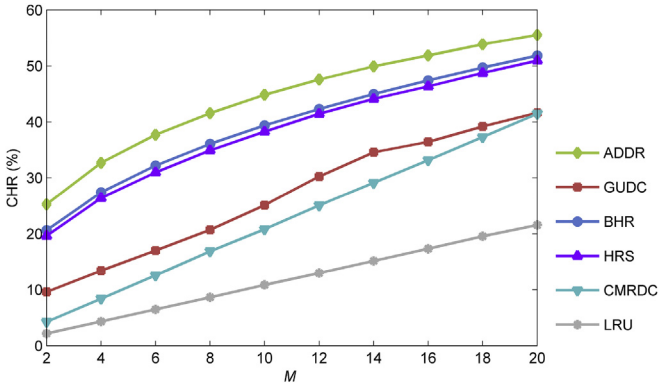

Fig. 7. Comparative CHRs obtained from all algorithms with different number of servers.

obtained. Besides CHRs, the CRR performance (Fig. 8) of most of the algorithms reduced due to less and less data need to be replaced to save cache space, except for ADDR and BHR, which the CRR performance remains stable to get a as higher CHR as possible by fast-up the copies replacement throughout the experiment even when their CRRs increased slowly, their CRRs are much lower than the other algorithms and such increase can also be ignored. Moreover, the LIR performance (Fig. 9) of all algorithms can still remain stable throughout the experiment due to the data scheduling strategy and the analysis mentioned above.

4.2.2. *Experiments using different parameters*

Obviously, the performance of our proposed algorithm will be affected by the choice of decay coefficient and access distributions, and so, some experiments were also conducted to demonstrate adaptability

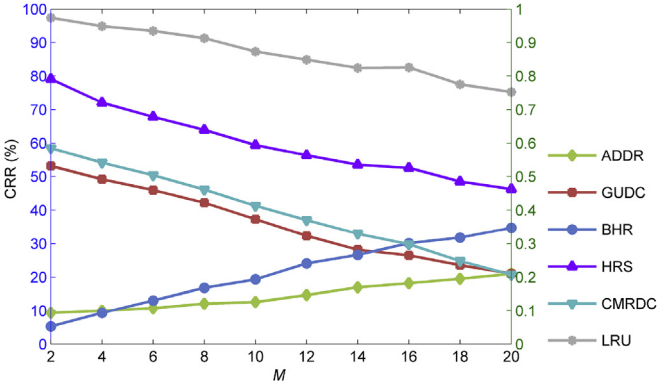

Fig. 8. Comparative CRRs obtained from ADDR, BHR (right axis) and other algorithms (left axis) with different number of servers.

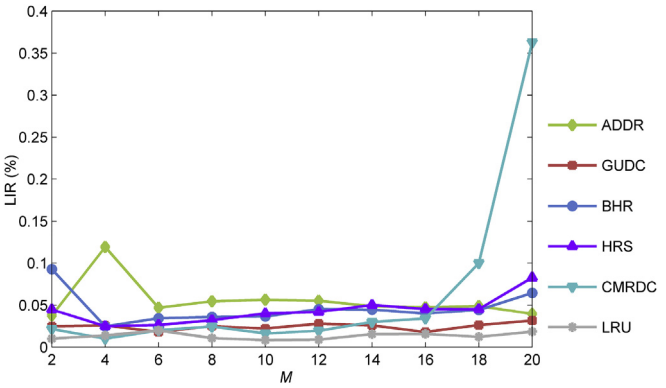

Fig. 9. Comparative LIRs obtained from all algorithms with different number of servers.

of ADDR.

Since the above experiments had confirmed the improvement of the performance of ADDR in CHRs, CRRs and LIRs, only the performance changes of ADDR are considered in the next experiments. In addition, the relationships among CRRs, LIRs and CHRs are also confirmed based on Figs. 4–9, only the CHRs will be evaluated in the next experiments.

Because different decay coefficients σ will lead to different amounts of sub-parts of H_i ($i \in [1\ \ L]$) be used to compute the popularity of each data based on Eq. (1). Obviously, the use of too many sub-parts of H_i will reduce the effect of the timeliness, and some invalid features will be obtained. Then, the data which is no longer popular may be selected as replica and stored in cache as copies. Also, the use of too little sub-parts of H_i can't get enough information to judge which data is popular and which one is not, and this situation is similar with the system which is just started at the very beginning as shown in Fig. 3 (during 0–30 m). As shown in Fig. 10, a small decay coefficient ($\sigma = 1.5$) and a large decay coefficient ($\sigma = 25$) all lead to a low CHR performance and $\sigma = 5.0$ is the best choice in our designed simulation, where 10–20 sub-parts will be used to compute data popularities based on Fig. 2.

In fact, the value of σ defines a minimum access information unit in which all users access behaviors are stable and this time interval can be used to obtain enough access information to mine users' inner access patterns. Obviously, the size of time interval is decided by lots of parameters based on Eq. (3) and CHR performance which will decide how many data have to be requested from servers. Based on Table 2, CHR performance and Eq. (3), we can estimate that the total access information length of all sub-parts is about [600 1000] when $\sigma = 5.0$. Meanwhile, research shows that users navigation depths is about 5–10 (Serdar and Veysi, 2012) and then, users access behaviors may change. Thus, the total access information length of all sub-parts must be less than the sum of all

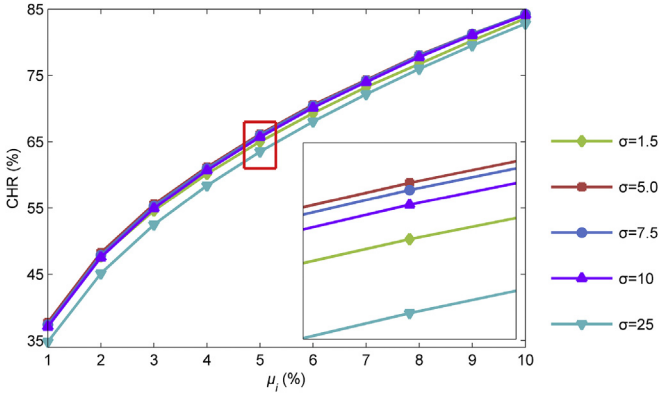

Fig. 10. Comparative CHRs obtained from ADDR with different decay coefficient.

users' navigation depths, i.e. 1500. As shown in Fig. 2, the total access information length of all sub-parts is larger than 1500 when $\sigma = 25.0$ as well as the total access information length of all sub-parts is not enough to mine access patterns when $\sigma = 1.5$.

It is clear that different distribution parameter represents different concentration degree of user access, although they all satisfy Zipf law (Shi et al., 2005; Newman, 2005). Some researches give an approximate range of the access distribution parameter in about 0.600–0.950 (Rui et al., 2017) and thus, an experiment was performed using different distribution parameters.

As shown in Fig. 11, the performance of CHR improves with an increase of the distribution parameter. This occurs because a larger distribution parameter represents a more concentrated access distribution and therefore, fewer data must be selected as copies. The results also show that the proposed algorithm can adapt to all kinds of application behaviors.

4.2.3. *Experiment based on real system*

The Instructional Workload Trace (IWT) is recorded by a typical real distributed file system which is deployed in the University of California at Berkeley (Roselli et al., 2000) and includes two groups of Hewlett-Packard series 700 workstations running HP-UX 9.05, and totally includes twenty machines in these two groups. In order to check the performance of all mentioned algorithms in a real system, an experiment based on IWT is also proposed to demonstrate adaptability of ADDR in real system which is shown in Fig. 12.

Similar to the experimental results shown in Fig. 4, ADDR can also get a better average CHR performance than others throughout the experiment, although the performance advantages of ADDR can almost be ignored by comparing with BHR and HRS when the cache buffer size is larger than 3%. As shown in Fig. 11, a larger distribution parameter

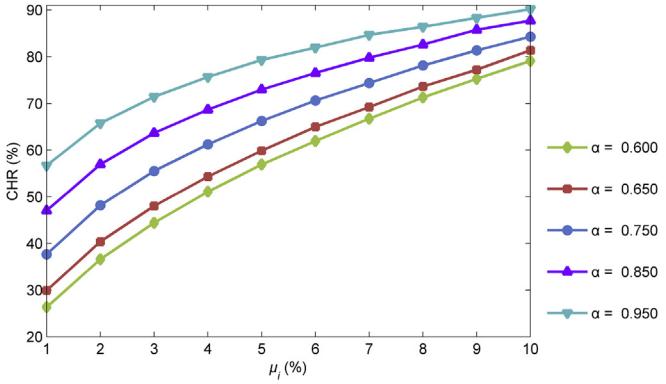

Fig. 11. Comparative CHRs obtained from ADDR with different distribution of access behavior.

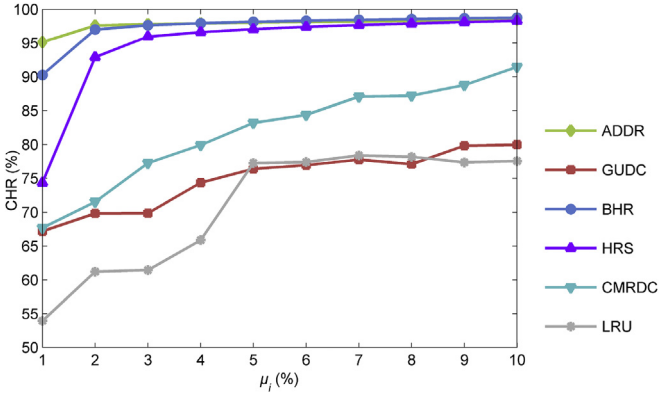

Fig. 12. Comparative CHRs obtained from algorithms based on real system's access records.

represents a higher CHR performance or the corresponding performance can be obtained by using less cache buffer size. In this experiment, user access behaviors are highly centralized and thus, ADDR, BHR and HRS can almost obtain the best performance when the cache buffer size is larger enough and their CHR performance cannot be further improved by expanding the cache buffer size. Thus, the experimental result based on IWT indicates that ADDR can also achieve a better performance than other algorithms in real environment.

4.3. Discussions

Due to the massive data storage requirement, geospatial data have to be separately stored into different storage nodes in distributed GIS which can not only meet the requirements of load balance among servers, but also reduce the risk of single node failure. Unfortunately, users' access behaviors will change continuously with the system running and the traditional data placement strategies, such as Redundant Array of Independent Disks (RAID) (Zhu et al., 2006) and DCCP methods, cannot meet the requirements of these dynamic characteristics for their data distributions among storage nodes need to be synchronously adjusted. The experiments show that the proposed algorithm can achieve better CHR as well as smaller CRR performances than other algorithms in all respects, and can adapt to dynamic environments. And the experiment based on IWT further demonstrated that the proposed algorithm can be used for real systems.

Obviously, the premise of using the algorithm is that users' access behaviors satisfy some intrinsic laws, for example Zipf law (Shi et al., 2005; Newman, 2005) which is a typical kind of access patterns in GIS (Pan et al., 2017; Rui et al., 2012; Wang et al., 2016), where users' access to data is extremely unbalanced and may most of requests are sent to small amounts of data.

Moreover, although the experiment results show that ADDR can be used in real system, our simulations considered only the ideal conditions, such as the identical capabilities of servers and the unique types of geospatial data. In real situations, some other external factors, such as the difference of disk I/O speed and cache space as well as the distance among service nodes, must be considered in our future works.

5. Conclusions and future works

As a typical service-intensive application, the requirement of reducing service response time is the key problem for distributed GIS. Replication strategy is one of effective solution for GIS which selects and keeps some copies in high speed cache in advance to avoid the delay of reading data from disks. Due to the complexity and variability of user behaviors, mining users' behavior based on a large enough historical information cannot track this change in time and also is impossible when

system is just started.

The solution proposed in this article is based on the timeliness of users' behaviors, and some experiments are conducted to check the CHRs, CRRs and LIRs performances comparing with lots of other typical algorithms. The results show that our proposed algorithm can achieve the best CHR performance and the performance improvement is approximately 10.2%–96.8%. Also, ADDR and BHR can achieve almost the same CRR performance and the average CRR performance is lower than most of the other algorithms about 94%. Due to the simulation strategy, all algorithms can achieve perfect load balancing performance and their LIRs are lower than 0.4% and basically, this imbalance can be ignored.

The probability of accessing the same cluster which holds the requested data by different users will be rather high in our algorithm, leading to possible remote data scheduling problems. A comprehensive consideration of both loading balancing and remote data scheduling efficiency will be an important future work. Moreover, a more complex model considering the heterogeneity of the nodes or servers and a mixed model considering multiple types of geospatial data will also be our future studies.

Acknowledgements

The research is supported by National Key Research and Development Program of China under Grant No. 2017YFB0504202 and by the National Natural Science Foundation of China (No. 41671382, 41271398, 61572372).

References

Amjad, T., Sher, M., Daud, A., 2012. A survey of dynamic replication strategies for improving data availability in data grids. Future Gen. Comput. Syst. 28 (2), 337–349.

Bell, D.G., Kuehnel, F., Maxwell, C., Kim, R., Kasraie, K., Gaskins, T., 2007. NASAWorld Wind: opensource GIS for mission operations. In: Aerospace Conference, pp. 1–9.

Boulos, M.N., 2005. Web GIS in practice III: creating a simple interactive map of England's strategic health authorities using google maps api, google earth KML, and MSN virtual earth map control. Int. J. Health Geogr. 4, 2269–2272.

Burghes, D.N., Huntley, I., McDonald, J., 1982. In: Horwood, E. (Ed.), Applying Mathematics: a Curse in Mathematical Modelling (Chichester, West Sussex, and New York).

Chang, R., Chang, J., Lin, S., 2007. Job scheduling and data replication on data grids. Future Gen. Comput. Syst. 23 (7), 846–860.

Chang, R.S., Chang, H.P., 2008. A dynamic data replication strategy using access-weights in data grids. J. Supercomput. 45 (3), 277–295.

Chervenak, A., Schuler, R., Ripeanu, M., Amer, M., Bharathi, S., Foster, I., 2009. The globus replica location service: design and experience. IEEE Trans. Parallel Distrib. Syst. 20 (9), 1260–1272.

Gibbs, N.E., Poole, W.G., Stockmeyer, P.K., 1976. An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM J. Numer. Analysis 13 (2), 236–250.

Li, Y., Zhong, E., Wang, E., Huang, Y., 2010. Markov model in prefetching spatial data. Bull. Surv. Mapp. 7, 1–4.

Madison, A.W., Batson, A.P., 1976. Characteristics of program localities. Commun. ACM 19, 285–294.

Michael, J., Caulfield, 2010. Apportioning representatives in the United States congress - jefferson's method of apportionment. Convergence. https://doi.org/10.4169/loci003163.

Newman, M., 2005. Power laws, pareto distributions and Zipf's law. Contemp. Phys. 46 (5), 323–351.

Park, S.M., Kim, J.H., Ko, Y.B., Yoon, W.S., 2004. Dynamic grid replication strategy based on internet hierarchy. Int. Workshop Grid Coop. Comput. Lect. Note Comput. Sci. 3033, 838–846.

Pan, S., Li, Y., Xu, Z., Chong, Y., 2015. Distributed storage algorithm for geospatial image data based on data access patterns. PLoS One 10 (7), e0133029. https://doi.org/10.1371/journal.pone.0133029.

Pan, S., Chong, Y., Zhang, H., Tan, X., 2017. A global user-driven model for tile prefetching in web geographical information systems. PLoS One 12 (1), e0170195. https://doi.org/10.1371/journal.pone.0170195.

Roselli, D.S., Lorch, J.R., Anderson, T.E., 2000. A comparison of file system workloads. In USENIX annual technical conference. General Track 41–54.

Rui, L., Rui, G., Xu, Z.Q., Feng, W., 2012. A prefetching model based on access popularity for geospatial data in a cluster-based caching system. Int. J. Geogr. Inf. Sci. https://doi.org/10.1080/13658816.2012.659184.

Rui, L., Feng, W., Wu, H.Y., Huang, Q.Y., 2017. A replication strategy for a distributed high-speed caching system based on spatiotemporal access patterns of geospatial data. Comput. Environ. Urban Syst. Part B 163–172.

Sashi, K., Thanamani, A., 2011. Dynamic replication in a data grid using a modified bhr region based algorithm. Future Gen. Comput. Syst. 27 (2), 202–210.

Serdar, Y., Veysi, İ., 2012. Retrospective adaptive prefetching for interactive Web GIS applications. Geoinformatica 16, 435–466. https://doi.org/10.1007/s10707-011-0141-8.

Shi, L., Gu, Z., Wei, L., Shi, Y., 2005. Quantitative analysis of Zipf 'S law on web cache. Lect. Notes Comput. Sci. 3758, 845–852.

Wang, T., Yao, S., Xu, Z., Jia, S., 2016. DCCP: an effective data placement strategy for data-intensive computations in distributed cloud computing systems. J. Supercomput 72 (7), 2537–2564.

Xiong, L., Xu, Z., Wang, H., 2016. Prefetching scheme for massive spatiotemporal data in a smart city. Int. J. Distributed Sens. Netw. 2 (1) https://doi.org/10.1155/2016/4127358.

Zhang, Y., Sun, X., Baowei, W., 2016. Efficient algorithm for k-barrier coverage based on integer linear programming. China Commun. 13 (7), 16–23.

Zhu, Y.F., Jiang, H., Qin, X., Fend, D., Swanson, D., 2006. Exploiting redundancy to boost performance in a RAID-10 style cluster-based file system. Clust. Comput. 9 (4), 433–447.