J Supercomput (2017) 73:4324–4346
DOI 10.1007/s11227-017-2012-z

An enhanced active caching strategy for data-intensive computations in distributed GIS

Shaoming Pan[1,3] · **Yanwen Chong**[1] · **Zhengquan Xu**[1,3] · **Xicheng Tan**[2]

Published online: 20 March 2017
© The Author(s) 2017. This article is an open access publication

Abstract Caching can prepare data for computational tasks in advance by tracking the requirements and behaviors of distributed geographical information systems to reduce network latency and improve computational performance. This paper presents an enhanced method to actively cache data for data-intensive computations that considers both data relationships and the timeliness of those relationships. First, the access correlations, the correlation steps and the times of the correlations are computed based on the behaviors of the computational tasks. Because the influence of historically accessed records will decrease gradually over time, only recently accessed records are used. To track changes in the relationships and prevent cache waste problems, each record is given a different age-based weight. A conditional caching probability can then be computed based on the timeliness relationships, which can be used to find the appropriate data to compute simultaneously. Finally, we present several experiments that compare the proposed method with techniques that use other data placement strategies, active caching strategies and passive caching algorithms. The results show that the proposed model has better performance than other algorithms in all respects. In addition, the proposed model results in a lower cache replacement ratio. The experiments with different data sets on different data scales indicate that the proposed algorithm can also be used in large-scale distributed environments.

✉ Shaoming Pan
pansm@whu.edu.cn

1 State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, Hubei, China

2 Department of Spatial Information and Digital Technology, International School of Software, Wuhan University, Wuhan, China

3 Collaborative Innovation Center for Geospatial Technology, Wuhan, Hubei, China

Keywords Active caching · Distributed computing · Data correlation · Spatial data · Distributed GIS

1 Introduction

With developments in information and communication technology (ICT), geographical information systems (GIS) have been widely used in many fields, including land and resource investigations, weather forecasts and disaster prediction, and urban and road traffic planning [1]. In those cases, GIS must process large amounts of both spatial data and single mapping data; a large amount of those data may be in real time [2]. Thus, several algorithms have been proposed to meet the requirements of data computation in distributed GIS [3–6].

The efficiencies of *locating data* (determining where the data are stored), *transferring data* (obtaining data from a local storage node or remote storage nodes) [7] and *processing data* (computing or analyzing the data) are three important aspects that will affect the data computation and analysis performance.

The distributed parallel spatial index structure R-tree (DPR-tree) [8] and the Hilbert space-filling curve-based multi-tier parallel R-tree (HCMPR-tree) [9] are two typical methods used to improve the performance of *locating data* that have been proposed to improve querying efficiency in the distributed parallel environments. The DPR-tree algorithm uses HCSDP (spatial data partitioning based on the Hilbert curve) [10] technology to divide spatial data, and the HCMPR-tree algorithm provides a new multi-tier parallel spatial indexing structure to obtain better load balancing performance.

Because of its greater computational costs, algorithm parallelization is one of the most important solutions to improve *data processing* performance [11]. Algorithm parallelization partitions the data and performs computations using different nodes, and each node schedules and computes its data simultaneously based on the same procedures. Some computations can be scheduled at the same time to reduce total computational costs. Yao et al. [12] presented a parallel algorithm for buffer analysis based on grid computing that decomposed the computational tasks according to both the map layer and the geographic spatial area. Pang et al. [13] and Fernandez et al. [14] realized parallel computing by dividing and storing related data in the same computing node.

Improving network transmission efficiency and reducing the amount of network transmission data are two aspects of optimizing *data transfer* performance. The data transfer rate strategy is a method to reduce network transmission costs [7] that stores all related data in the same node to reduce the data transfer rate between distributed nodes, thus saving data transfer time costs between nodes [4,15]. Dynamic computation correlation cata placement (DCCP) [4] distributes and stores data that have high dynamic computation correlations in the same data center by considering not only the I/O load but also the capacity of the data centers. Access pattern-based distributed storage algorithm (APSA) [15] also distributes and stores data that have high access correlations in different data centers to allow concurrent access.

Although the algorithms described above have been used to obtain good results, they have several disadvantages that must be considered further. First, the data storage

capacity requirements have increased by a factor of thousands over the past decade with the developments of ICT; therefore, all data must be distributed into many different storage nodes before they can be used for computations or analyses. Although data placement strategies can store some related data in a local storage node, different data calculation and analysis tasks will have different data distribution requirements. For example, urban and road traffic planning will focus on road traffic data; thus, related road traffic network data from an urban area must be stored in the same node. The fixed mode of data placement may not satisfy the requirements of different applications in a real-time system. In addition, with changing data relationships and application requirements, data placement strategies must be adjusted synchronously, potentially to a large number of data migrations between storage nodes and substantially affecting GIS performance.

However, with the rapid increase in network bandwidth, the data transfer time cost is usually less than the data processing time cost. Therefore, preparing the next piece of data while a particular piece of data is being used can reduce total computation time costs; that is, if data transfer and data processing can be performed in parallel, the data transfer time cost can be ignored. Thus, the key issue is to predict and cache in advance the data that will be computed or analyzed during the next step.

In contrast to traditional passive computing algorithms that prepare the current data based on the application's current requirements or some typical active caching strategy that prepare the current data based on the whole historical access information, this article proposes an enhanced active caching strategy for data computations that prepares data in advance by considering both data relationships and the timeliness of those relationships.

This article is organized as follows. Section 2 introduces related studies about caching algorithms based on the application's behaviors and the relationships between the data. A new active computing model based on a data-caching algorithm is presented in Sect. 3. The results of experiments are presented and discussed in Sect. 4. Finally, Sect. 5 provides the conclusions of this study and discusses our future work.

2 Related work

Although actively predicting and caching data have not attracted the attention of researchers in the supercomputing field, caching technology is widely used in information systems because it can be used to improve the quality of service and speed up the response time for users.

GIS is a typical data-intensive application [16] that serves a large number of users in which the pyramid model is used to divide the data into smaller pieces called tiles [17]. The main purpose of caching in GIS is to prefetch the appropriate tiles from storage nodes to prepare the data for the application in advance. Because the server stores large amounts of tiles that can be prefetched, it is difficult to determine which tiles should be prefetched. Many studies have focused on this key problem.

First in first out and least recently used (LRU) are passive caching algorithms that only save data that are currently being accessed; they never proactively prefetch tiles from storage nodes. These algorithms are widely used by Google [18], networked

geographic information systems (NGISs) [19] and NASA [20], improving system performance.

In active caching fields, applications (i.e., Google Earth's Web browser) use historical information to estimate possible tiles that are likely to be used immediately [21–23]. Markov Chain model is a well-known active caching algorithm which use a Markov Chain to predict client's next movements [24,25]. Although these methods have several advantages for GIS, they are primarily used by clients that separately read data from the server in advance for caching based on their own behavior, potentially leading to cache waste (duplication of data units in a cache buffer or data that are cached but will not be used soon) in distributed GIS [26].

Moreover, several global user-driven models have been proposed to address the cache waste problem. These models are primarily used by servers and are based on all of the clients' behaviors. In these models, access to spatial data satisfies intrinsic laws [27] that can be used to determine the relationships between them; those relationships can be used to predict the next data required when a certain piece of data is used [28]. The proposed global user-driven models can generally be categorized as popularity- and correlation-based.

Popularity-based models, such as distributed high-speed caching based on spatial and temporal locality (DCST) [29] and bandwidth hierarchy-based replication (BHR) [30], calculate the popularities of all of the tiles and cache the tiles with higher popularities [31]. DCST uses the election scheme of the United States Congress to select the tiles to cache and uses a steady-state cache hit ratio parameter to limit the tile selection range, thus saving cache space. The main idea of BHR is to keep the required data in the same region as much as possible, thus reducing external–schedule time.

Correlation-based models such as data replicas based on the fuzzy logic system (FLSDR) [32], global user-driven model for tile prefetching (GUDC) [26] and prefetching scheme based on spatial–temporal attribute prediction (STAP) [33] dynamically cache the related data into a high-speed cache buffer to prepare the data for service in advance. FLSDR selects some data as an optimal replica with a minimum response time considering both the data queue and the data transfer. FLSDR then places the replica into the node from which the replica has the maximum probability of being repeatedly requested. GUDC computes all of the data relationships based on their historical access records and compares the conditional prefetching probability to cache or replace the data. STAP mines the relationships of spatiotemporal data based on their historical access records and then uses the autoregressive integrated moving average model to construct a predictive function to predict users' future behaviors.

Nevertheless, different computational tasks will use different data sets, and high-popularity data may not be needed next. Considering information from a typical historical access log [26] that is used by the application, the data with the highest popularity may not be accessed again. Moreover, the relationships or popularities of all of the data in distributed GIS change continuously during system operation [25]. For that reason, it might not always be appropriate to mine the patterns based on the whole historical access records to guide the replication strategy. Furthermore, we cannot obtain a sufficient number of historical access records if the system has just begun to operate.

Based on these analyses, we propose an enhanced method to actively cache data for data-intensive computations that considers the timeliness of both tile popularities and their relationships (CPR) in distributed GIS. Because we cannot obtain a sufficient number of historical access records and because the influence of historical access records will gradually decrease over time [34], we use only recent records, which can easily and quickly be obtained after the system is started. Each record is given a different weight based on its freshness; thus, we can closely track the changes in the relationships and avoid the cache waste problem. The passive caching strategy LRU is initially used to temporarily save data in the cache buffer, and the cache data will be replaced dynamically based on the most recent relationships.

3 Active caching model for data computations

3.1 Concepts

Figure 1 shows a typical architecture of a distributed GIS in which the spatial data are distributed and stored in M clusters and each cluster is composed of one storage node and several servers. Each server contains one high-speed cache buffer. The servers and storage node in the same cluster are connected by a LAN, and the clusters are connected through the Internet. In a distributed GIS, computational tasks such as remote sensing image correction are performed by clients and dispatched to a server by a load distributor based on the data's location. The server reads the data from a local storage node or remote storage node to perform the computational task.

Denote $D = \{d_1, d_2, \ldots, d_N\}$ as the set of all data that will be used for data computations by clients in a distributed GIS, where N is the total number of data and each element in D is labeled with a natural number $[1, N]$. Based on the analysis presented above, the data computation time costs are composed of the *locating data*

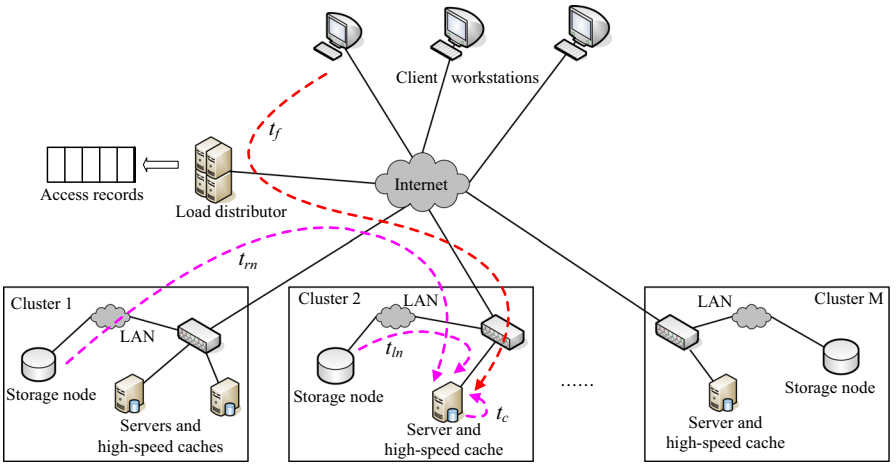

Fig. 1 Key architecture of the distributed GIS

time cost t_f (to find the data and dispatch the task), the *data processing time cost* t_s (to perform the computational task) and the *data transfer time cost* t_o (to obtain the data from the local storage node or remote storage node). The total computation time cost is $t = (t_f + t_s + t_o)$. Because of the pyramid model used in GIS, all of the data are the same size and can be located using the index number (for a certain data set); therefore, t_f and t_s are constants. If a certain piece of data is cached in advance based on when it will be scheduled and computed, then $t_o = t_c$; otherwise, $t_o = t_n$, where t_c is the time cost to obtain the data from the cache and t_n is the time cost to obtain the data from the network. Thus, the computation time costs for a certain piece of data d_i are as follows:

$$t_i = t_f + t_s + \lambda_i t_c + (1 - \lambda_i) t_n \tag{1}$$

where λ_i is a matching indicative factor and $\lambda_i = 1$ indicates that d_i is cached before it is scheduled and computed; otherwise, $\lambda_i = 0$.

Assume both that all of the computational tasks are scheduled synchronously and that the data sequence is chronologically recorded by the load distributor when each piece of data is used by the computational tasks. Let $Q = (q_1, q_2, \ldots, q_L)$ denote the entire sequence, where $q_k \in [1, N]$ denotes the label of the k-th computed piece of data that is scheduled by a certain computational task (i.e., $q_k = i$ indicates that the k-th computed piece of data is $d_i (i = 1, \ldots, N)$), and L is the total number of computations of all of the data. The total computation time costs for all of the computational tasks are as follows:

$$\begin{aligned} T &= \sum_{i=1}^{L} t_{q_i} = L(t_f + t_s) + \sum_{i=1}^{L} (\lambda_{q_i} t_c + (1 - \lambda_{q_i}) t_n) \\ &= L(t_f + t_s + t_n) - h(t_n - t_c) \end{aligned} \tag{2}$$

where $h = \sum_i^L \lambda_{q_i}$ is the total number of cache hits. Based on Eq. (2), the aim of reducing the total computation time costs can be transferred to obtaining a high cache hit rate $r = h/L$, and the key is to find the most appropriate data and actively cache them in advance when a certain piece of data is being computed. If the piece of data is stored in a local storage node, $t_n = t_{ln}$; otherwise, $t_n = t_{rn}$. Because $t_{rn} \geq t_{ln}$ in distributed GIS, adjusting the data placement can also improve the total computation time costs.

3.2 Active caching model

Active caching is a method of finding data that have close relationships with the data being computed and then prefetching and caching them in advance for the next computation. We can compute the relationships between all of the data based on their historical scheduling records considering both global access correlations [26,29] and the timeliness of their access correlations [34]. Because a large amount of spatial data is stored in distributed GIS and it is impossible to dynamically adjust the data placement among all of the clusters for reasons related to various computational tasks, the spatial data will be stored in the storage nodes randomly and evenly.

For a certain period, if d_i is scheduled and computed and d_j is also scheduled and computed after x steps, we denote that there is one x-step correlation from d_i to d_j, and their corresponding correlation weights and correlation steps can be denoted as w_x and s_x, respectively, where $d_i, d_j \in D$, $s_x = x$ and $w_{x-1} > w_x$ $(i, j \in [1, N], i \neq j)$.

Assuming that all of the servers will provide computational services continuously for all clients, all of the servers can process M users' requests simultaneously during a short period of time. Then, M is the largest step between two pieces of data in a schedule, and $x \leq M$. In general, denote $Q_k = (q_{k1}, q_{k2}, \ldots, q_{kM})$ as the sub-access vector of all of the data labels that were scheduled chronologically by the load distributor at a given moment. For $\forall d_i,\ d_j \in D\,(i, j \in [1, N])$, the access correlations, the correlation steps and the correlation times between d_i and d_j can be separately computed as follows based on typical data correlation mining algorithm [26] within the vector Q_k:

$$M_k(i, j) = \sum_{x=1}^{M-1} \sum_{y=x+1}^{M} v_{kx,ky}(i, j) w_{y-x} \qquad i, j \in [1, N] \tag{3}$$

$$E_k(i, j) = \sum_{x=1}^{M-1} \sum_{y=x+1}^{M} v_{kx,ky}(i, j) s_{y-x} \qquad i, j \in [1, N] \tag{4}$$

$$F_k(i, j) = \sum_{x=1}^{M-1} \sum_{y=x+1}^{M} v_{kx,ky}(i, j) \qquad i, j \in [1, N] \tag{5}$$

where $v_{kx,ky}\,(i, j) = 1$ when $q_{kx} = i$ and $q_{ky} = j$ or $q_{kx} = j$ and $q_{ky} = i$, otherwise $v_{kx,ky}\,(i, j) = 0$.

Because newer access information has a greater influence on the total access correlations [34], our enhanced model will consider both the different weight of access correlations within sub-access vector Q_k and the different weight of access correlations among all sub-access vectors $\{Q_1, Q_2, \ldots, Q_G\}$, where $Q = (Q_1, Q_2, \ldots, Q_G)$ and G is the total number of sub-access vectors. Thus, the total access correlations $M(i, j)$, their total correlation steps $E(i, j)$ and the correlation times $F(i, j)$ between d_i and d_j can be stated as follows:

$$M(i, j) = \sum_{k=1}^{G} M_k(i, j) w_{G-k+1} \qquad i, j \in [1, N] \tag{6}$$

$$E(i, j) = \sum_{k=1}^{G} E_k(i, j) w_{G-k+1} \qquad i, j \in [1, N] \tag{7}$$

$$F(i, j) = \sum_{k=1}^{G} F_k(i, j) w_{G-k+1} \qquad i, j \in [1, N] \tag{8}$$

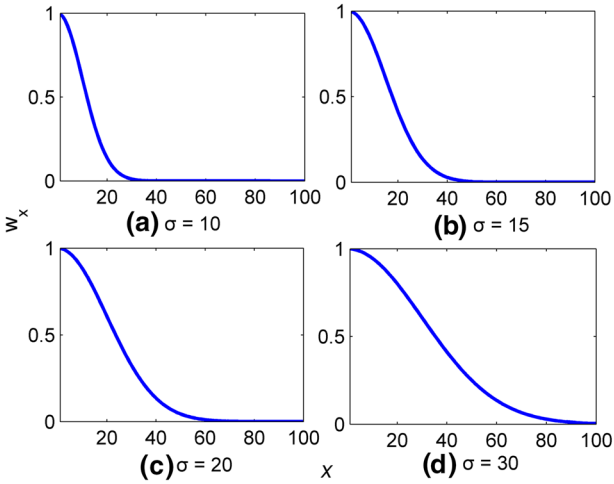

Fig. 2 *Decay curves* based on different decay coefficients

respectively. Because the influence of historically accessed records will decrease gradually over time, only recently accessed records are used to track changes in the relationships and prevent cache waste problems, and so, the access correlations, the correlation steps and the and correlation times within each sub-access vector need to be given a different age-based weight. Thus, the weight is a decay function of access steps x or sub-access vectors steps (i.e., denote the access steps between sub-access vector Q_k and Q_G as $G - k + 1$) and which can be defined as follows:

$$w_x = e^{-(x^2/2\sigma^2)} \tag{9}$$

where σ is the decay coefficient, and $x \in [1, G]$. Obviously, selecting a different decay coefficient will lead to a different amount of historical access information and different weights being used. Figure 2 shows several typical decay coefficient values and the corresponding decay curves.

As shown in Fig. 2, only 20–100 recent sub-access vectors will be used to compute the total access correlations, their total correlation steps and the correlation times based on Eqs. (6), (7) and (8). After that, the average correlation steps between d_i and d_j can easily be computed:

$$\bar{E}(i, j) = \frac{E(i, j)}{F(i, j)} \qquad i, j \in [1, N] \tag{10}$$

A close access relationship is determined by two aspects: (1) if the data are computed simultaneously and (2) if their access distance is short when they are computed simultaneously. Thus:

$$\vec{P}(i, j) = \frac{M(i, j)}{\bar{E}(i, j)} = \frac{M(i, j)}{E(i, j)} \times F(i, j) \qquad i, j \in [1, N] \tag{11}$$

either can indicate the age-based total caching probability for d_j when d_i is being computed or simply represents the probability that d_j will be computed in the next movement and which consider the difference of access correlations not only within a sub-access vector but also among sub-access vectors. Thus, for $\forall d_i \in D$, the age-based total caching probability of all other data can be obtained based on Eq. (11), and from that, we can find the largest element to predict its corresponding data when d_i is being computed. Thus, we can obtain a high cache hit rate when data are scheduled and computed to reduce the total computation time costs.

Furthermore, some computational tasks will always use some data portfolios to compute and find the destination; for example, navigation path planning will use the neighboring blocks one by one. Thus, an active caching strategy can use those data portfolios to obtain data more accurately. For example, if $(d_i\, d_j\, d_k)$ is a portfolio, the active caching strategy can produce a very precise estimation and actively cache the data d_k when the data d_i have just been computed and the data d_j are being computed. Thus, for $\forall d_i \in D$, let $\left\{A_1(i), A_2(i), \ldots, A_{C_i}(i)\right\}$ denote the set of all data portfolios for data d_i, where C_i is the total number of portfolios, each portfolio $A_n(i) = \left(d_{n1} d_{n2} \ldots d_{na_n} d_i\right)$ is a sub-vector of Q_k ($k \in [1, G]$, $n \in [1, C_i]$) and ends with the data d_i, and $a_n + 1$ is the length of $A_n(i)$. Then,

$$\vec{P}(A_n(i), j) = \frac{M(A_n(i), j)}{E(A_n(i), j)} \times F(A_n(i), j) \quad i, j \in [1, N],\ n \in [1, C_i] \tag{12}$$

can indicate the age-based total caching probability for d_j based on data portfolio $A_n(i)$ [26]. $\vec{P}(i, j)$ is clearly a special case of $\vec{P}(A_n(i), j)$ in which $\left(d_{n1} d_{n2} \ldots d_{na_n}\right)$ is null ($a_n = 0$) and $P(A(i), D) = (\vec{P}(A_n(i), j))_{C_i \times N}$ is the age-based total conditional caching matrix for all data portfolios of d_i.

Similarly, the data portfolios also have characteristics of timeliness and using some very old portfolios will also lead to obtain a wrong prediction. Thus, finding a valid data portfolio set for a certain data d_i is the key for $P(A(i), D)$. Thus, let $\xi_k(i)$ be the popularity of d_i based on Q_k ($k \in [1, G]$). The total popularities of d_i can be stated as follows:

$$\xi(i) = \sum_{x=1}^{G} \xi_k(i) w_{G-x+1} \qquad i, j \in [1, N] \tag{13}$$

where $\xi_k(i) = \sum_{x=1}^{M} v_{kx,kx}(i, i)$. The average popularity of all of the data can be computed as follows: $\bar{\xi} = \sum_{i=1}^{N} \xi(i)/N$ based on Q and w_x. Several studies have shown that only 20% of data will be requested repeatedly [25,26]; thus, the data with popularities higher than $\bar{\xi}$ are selected as the elements of the popular data set D_p. Based on Q and D_p, the age-based total conditional caching vector $P_v(i, D_p)$ and the age-based average conditional caching probabilities can be stated and computed easily as follows:

$$\bar{P}_v(i, D_p) = \frac{\sum_{k=1}^{N_p} \vec{P}(i, k)}{N_p} \tag{14}$$

where N_p is the total number of elements in the popular data set D_p. Thus, the data for which the age-based total conditional caching probabilities are higher than the age-based average conditional caching probability can be grouped together with d_i as a data portfolio. Moreover, we can select additional data into the portfolio to obtain a sufficiently large portfolio set (fewer than M elements) so as to get $A(i)$ which consider only the newest access information.

3.3 Active caching strategy

In distributed GIS, computations are proposed by clients, distributed to the server by the load distributor based on the data location and executed by the server. The load distributor records the historical access records and schedules servers to actively cache data in advance. The procedures of our active caching strategy are as follows:

Step 1 Each server independently saves data to the high-speed cache buffer and replaces data in the buffer based on the LRU strategy when the system is beginning to operate. Set $s = 1$ and compute w_x based on the parameter value of decay coefficient and Eq. (9). Set $X = 2\sigma$ as the max number of sub-access vector which will be used to compute age-based total popularities and total conditional caching probabilities (the area of decay curves is less than 5% of total area when $x > 2\sigma$).

Step 2 The load distributor chronologically records an index of all of the data that are computed by all of the clients, and we can then obtain their historical scheduling sub-access vector $Q_s = (q_{s1}, q_{s2}, \ldots, q_{sM})$ and add Q_s to the end of Q and update Q. It is clear that $d_{q_{sM}}$ is the data being computed.

Step 3 Compute the popularities for all data D based on Q_s. It is clear that only the accessed data set based on Q_s needs to be computed and the popularities for all other data are zero.

Step 4 Compute the total popularities of all data and average popularity of all of the data based on Eq. (13), Find the data portfolio set based on Eqs. (13) and (14) and then the age-based total conditional caching probability matrix can then be computed based on Eq. (12), where G can be set as X.

Step 5 Let $U(q_{sM}) = (\mu_1(q_{sM}), \mu_2(q_{sM}), \ldots, \mu_{C_{q_{sM}}}(q_{sM}))$ denote the matching indicator of all data portfolios. If $A_i(q_{sM})$ is a subsequence of Q_s, set $\mu_i(q_{sM}) = 1$; otherwise, set $\mu_i(q_{sM}) = 0$.

Step 6 Compute the age-based total conditional caching probabilities for all of the data as follows:

$$P_s(q_{sM}, D) = \begin{bmatrix} \sum_{l=1}^{C_{q_{sM}}} \vec{P}(A_l(q_{sM}), 1)\mu_l(q_{sM}) \\ \sum_{l=1}^{C_{q_{sM}}} \vec{P}(A_l(q_{sM}), 2)\mu_l(q_{sM}) \\ \vdots \\ \sum_{l=1}^{C_{q_{sM}}} \vec{P}(A_l(q_{sM}), N)\mu_l(q_{sM}) \end{bmatrix}^T = U(q_{sM}) \cdot P(A(q_{sM}), D) \quad q_{sM} \in [1, N] \tag{15}$$

Thus, we can find the data with the highest degrees of correlation with the data $d_{q_{sM}}$ and then prefetch and cache the corresponding data (i.e., if the second one is the largest element in $P_s(q_{sM}, D)$, then d_2 will be prefetched and cached).
Step 7 Set $s = s + 1$ and repeat Steps 2–7 until the computational tasks have been completed.

Similar to GUDC [26], more than one piece of data can be selected and actively cached based on the total conditional caching probabilities to increase the data-caching speed at the beginning of system operation.

3.4 Algorithm analysis

The computational complexity of calculating the total caching probabilities of all of the data based on Eq. (11) is approximately $O(N^3G)$. Because a distributed GIS contains a large amount of data and many sub-access vectors, it is both impossible and unnecessary to compute the total caching probabilities of all of the data each time by recalculating the access correlations, the correlation steps and the correlation times based on Eqs. (3), (4) and (5) when a piece of data is requested. Indeed, the historical results can be reused, and only the newest value based on the newest sub-access vector needs to be calculated. Thus, the computational complexity is approximately $O(M^3)$, and it is possible to calculate the total caching probabilities because of the limited number of clusters scheduled by a single load distributor in a real distributed GIS. Moreover, the layered physical network topology can be used by configuring many clusters to decentralize the computational services.

Furthermore, a small w_x makes little contribution to the total conditional caching probabilities; therefore, we can set $w_x = 0$ when $x > 2\sigma$. Thus, only limited historical access information will be used to compute the age-based total conditional caching probabilities because most values of w_x are zero. Only a tiny fraction of $M_k(i, j)$, $E_k(i, j)$, and $F_k(i, j)$ needs to be stored for the next computation, and the required memory is approximately $O\left(M^2X\right)$, where X is the number of w_x with nonzero values.

4 Simulations and experiments

4.1 Simulation design

To illustrate the performance of the proposed algorithm, we designed a typical earth observation system, which is called GlobeSIGht [27]. The application uses SRTM90 (90-meter-resolution global terrain data files from the Shuttle Radar Topography Mission) data for terrain analysis computations [35]. The simulation parameters are listed in Table 1.

As shown in Fig. 1, each computation center has one local storage node and can obtain data from remote storage nodes through the network with a bandwidth of 10–100 Mbps. The historical data access record is produced by GlobeSIGht [27] based on a Zipf-like law [26]. All of the experiments are measured using the average computation time cost, which represents the average computation time for one piece of data. In

Table 1 Simulation parameters

Parameter	Value
Number of clusters (M)	2–20
Number of nodes in each cluster	1
High-speed space in each node	300–3000
Size of each datum	≈44 KB
Size of the data set (N)	50,000–500,000
Connectivity bandwidth	10–100 Mbps
Number of clients (users)	100
Number of data accessed (L) by all clients	3,000,000–30,000,000

this simulation, the terrain analysis computation time is approximately 0.05 s, and the latency of transmitting data over the network, disk and cache is based on the bandwidth, the disk I/O speed, the cache I/O speed and the size of the data. No additional latency is considered in the simulation. Based on the size of a single piece of data, the number of cached data for each server is limited to 300–3000 (i.e., 13–130 MB) based on the cache buffer size. For simplicity, the experiments assume that all of the computation centers and storage nodes have the same abilities.

Experiments are performed using different passive caching strategy (PC) algorithms (such as LRU), data placement strategy (DP) algorithms (such as the DCCP algorithm [4]) and active caching strategy (AC) algorithms (such as the GUDC algorithm [26] and CPR). The PC algorithms store the data in the storage nodes randomly and then obtain and cache the data from the storage nodes based on the behaviors of the applications. The DP algorithms store related data in the same storage node in advance and then obtain data from a local storage node or remote storage nodes based on their locations. The AC algorithms store all of the data in the storage nodes randomly and then predict and cache the related data from the storage node in advance while certain data are being computed. Because of the limited cache buffer size, the AC methods save cache space by using the LRU strategy to delete cached data from the cache buffer.

In addition, several caching strategies that are described in Sect. 2 are used to compare the performance with that of the proposed CPR algorithm, and several experiments are performed using the CPR algorithm based on different active caching parameters.

To illustrate the performance of the proposed algorithm, which actively caches data by considering both the data's popularity and their relationships (labeled AC_CPR in the figures), we compare that algorithm with the following methods:

1. An optimal method (Best) that uses the DP algorithm to place the data in some storage nodes (labeled DP_Best) and uses the same strategy to schedule the computations. In this case, all of the data computation centers can obtain the needed data from their local storage node. This method clearly cannot be implemented in practice and can only be used either for a comparative analysis or as a reference.
2. A PC method that uses DCST [29] to cache the data in advance and uses LRU to replace the cached data (labeled PC_DCST).
3. A PC method that uses LRU to replace the cached data (labeled PC_LRU).

4. A method that uses the DCCP [4] data placement strategy to place the data and does not use active caching (labeled DP_DCCP).
5. An AC method that uses GUDC [26] to cache the data in advance and does not use a data placement strategy (labeled AC_GUDC).

Because selecting different decay coefficients will lead to different amounts of historical access information and different weights, Serdar [21] gives a detailed proposal for the navigation depth; thus, we set $\sigma = 15$ in the simulations. Furthermore, an experiment that uses different values of σ is performed.

4.2 Experiments and results

4.2.1 Experiments using different computation algorithms

Figures 3, 4 and 5 show the average computation time costs, average cache hit ratios and average cache replacement ratios for all of the algorithms using 10 computation centers and 600 pieces of cached data in each server. In this experiment, AC_CPR randomly places all of the data in storage nodes and then uses the CPR strategy to actively cache the data. DP_DCCP and DP_Best place all of the data in storage nodes based on their own strategies. Neither approach uses a caching strategy.

As shown in Fig. 3, the performance of all of the algorithms remains stable throughout the experiment, and AC_CPR performs better than the others. Although the performance of AC_CPR is worse than the optimal method, AC_CPR is the closest to the optimal strategy. Although the performance improvement of AC_CPR for the average computation time costs appears unremarkable, the average cache hit ratio is improved by approximately 9.5–93.8%, and the average cache replacement ratio is

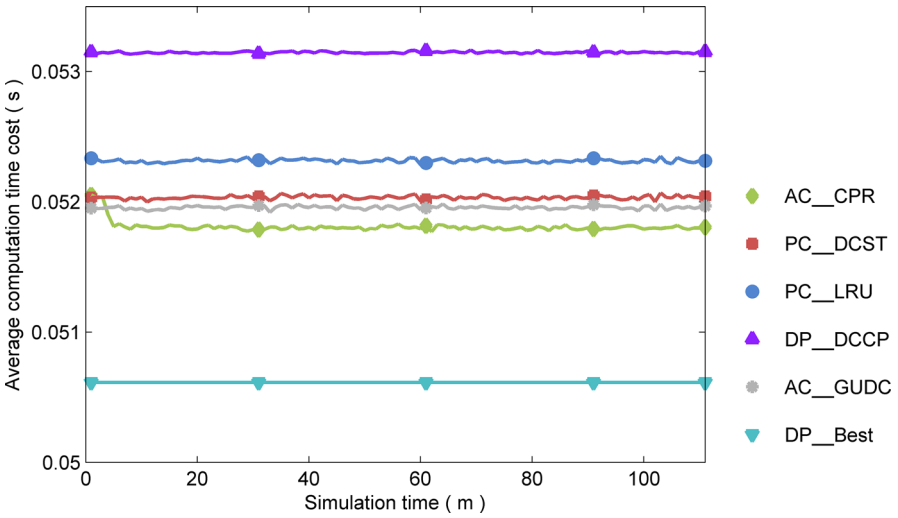

Fig. 3 Comparison of the average computation time costs obtained by different algorithms using 10 computation centers

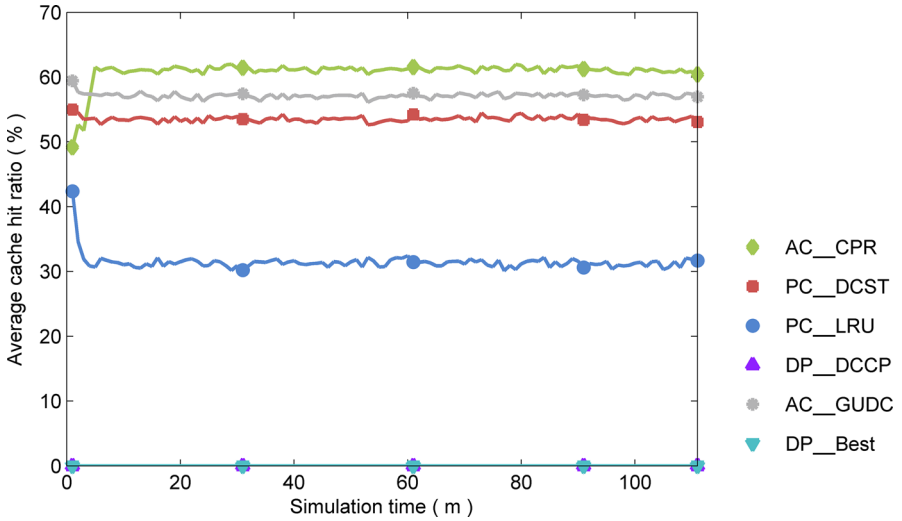

Fig. 4 Comparison of the average cache hit ratios obtained by the algorithms using 10 computation centers

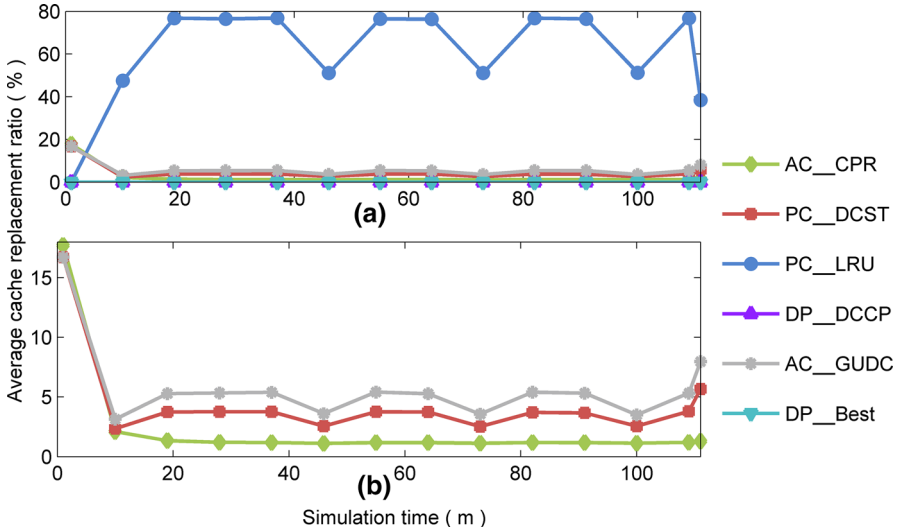

Fig. 5 Comparison of the average cache replacement ratios obtained by the different algorithms using 10 computation centers

reduced by approximately 59.69–71.15%, except for the DP methods, which have a cache buffer size of zero. The average computation time costs include the locating data time cost, the data processing time cost and the data transfer time cost. However, the proposed method cannot improve the performance of the data processing. The average locating data time cost and data transfer time cost can be estimated from the difference

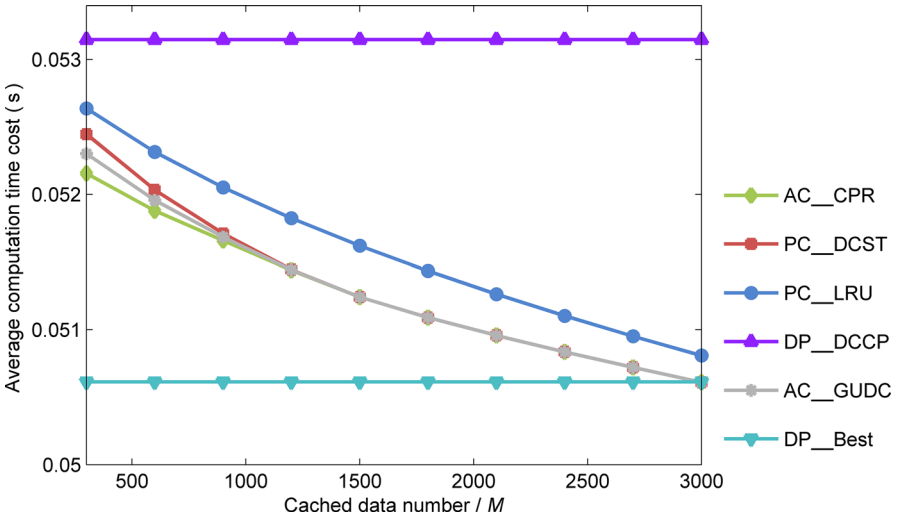

Fig. 6 Comparison of the average computation time costs obtained by different algorithms for 10 computation centers with 300–3000 pieces of cached data

between the average computation time costs of DP_Best and the other algorithms; thus, the performance improvement of AC_CPR is approximately 12–72%.

Moreover, AC_CPR uses LRU to passively cache data at the very beginning of system operation; thus, the average computation time costs and average cache hit ratios are lower and the average cache replacement ratio is higher. However, AC_CPR can quickly cache the appropriate data once a sufficient amount of historical access information is obtained, and the performance then remains stable.

The computation performance can be improved further by increasing the cache buffer size (Fig. 6).

As shown in Fig. 6, the performance of the DP algorithms (DP_DCCP and DP_Best) did not change, whereas the performances of the AC and PC algorithms improved with increasing cache buffer size. DP_DCCP and DP_Best have no cache strategies, and they always obtain data from a local storage node or network shares. However, the AC and PC algorithms use the cache buffer to store data that are prefetched from other storage nodes in advance. A larger cache buffer size indicates the greater possibility of a cache hit; thus, we can obtain higher computation performance because the cache I/O is faster than both the disk I/O and the network I/O.

Figure 6 also shows that active data-caching strategies can achieve better performance than passive data-caching strategies because they will predict the computational tasks' behavior and prepare data for the tasks in advance. The CPR strategy provides clear performance advantages over the other algorithms even when the cache buffer is very small. When the cache buffer is large enough, active data-caching strategies can approach the performance of the optimal strategy.

To check the performance of all of the algorithms with different numbers of computation server centers, an experiment is conducted with 600 pieces of cached data and between 2 and 20 computation centers. The results are shown in Fig. 7.

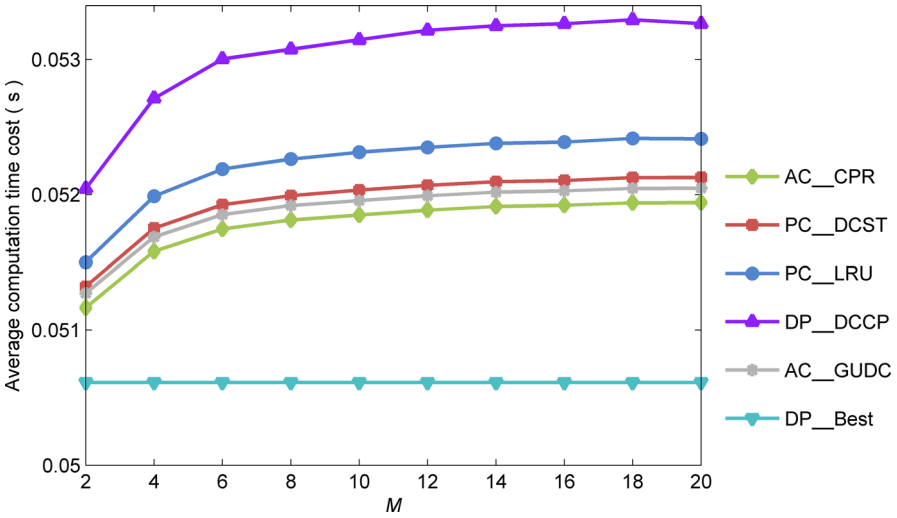

Fig. 7 Comparison of the average computation time costs obtained by different algorithms for 2–20 servers and 600 pieces of cached data

Similar to the previous analysis, DP_Best has the best performance. However, the performances of all of the other algorithms decrease with an increasing number of computation server centers. More computation server centers indicate that fewer data will be stored in the local storage node. Thus, more of the computation data must be obtained from remote storage nodes, and the performance will inevitably decrease. The results shown in Fig. 7 indicate that active caching algorithms provide good computation performance and have lower degradation rates than the other methods with more than 10 computation server centers; thus, the proposed algorithm can be used in large-scale distributed GIS and will have more advantages.

Another important aspect of verifying the adaptability of the algorithm for data-intensive computations is testing the stability of the algorithm's performance on different data scales. Thus, an experiment was conducted in which the number of data varies from 50,000 to 500,000. The results are shown in Fig. 8.

Figure 8 shows the change in performance with the increasing size of the data sets for the algorithms. With the exception of DP_Best, AC_CPR always provides the best performance with an increasing amount of data. In addition, the performances of all of the algorithms decrease with increasing amount of data, with the exception of DP_Best. Larger data sets indicate either that more data will be obtained from remote storage nodes (for the DP algorithms) or that more choices (hard to active caching) are needed to predict the next computation step (for the AC algorithms). The results indicate both that AC_CPR can achieve nearly the same stability as the DP_DCCP method and that the two algorithms have the best adaptability for large-scale environments.

In addition, an experiment was conducted using different data sets. The results from using the NLT Landsat-7 data [27] are shown in Fig. 9a, and the results from using the SRTM90 data are shown in Fig. 9b. The NLT Landsat-7 data set is larger than the SRTM90 data set.

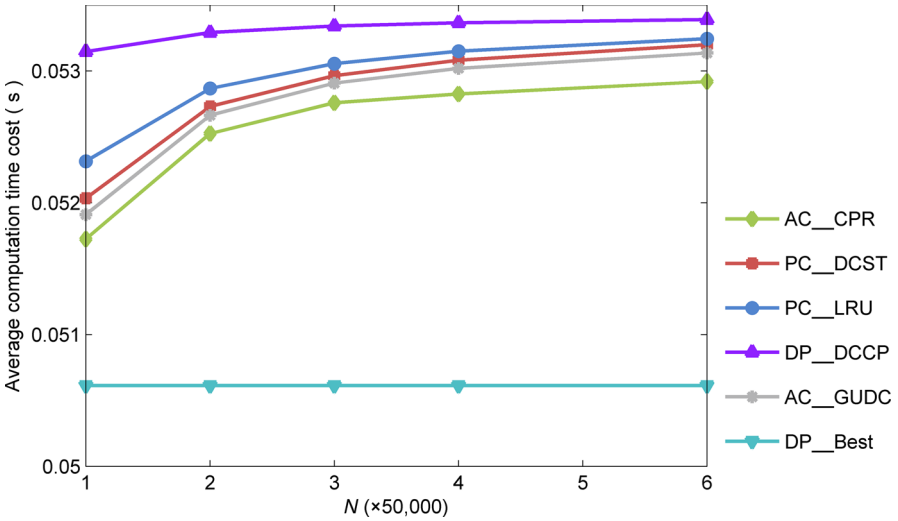

Fig. 8 Comparison of the average computation time costs obtained by different algorithms using 10 computation centers when the data size varies from 50,000 to 500,000

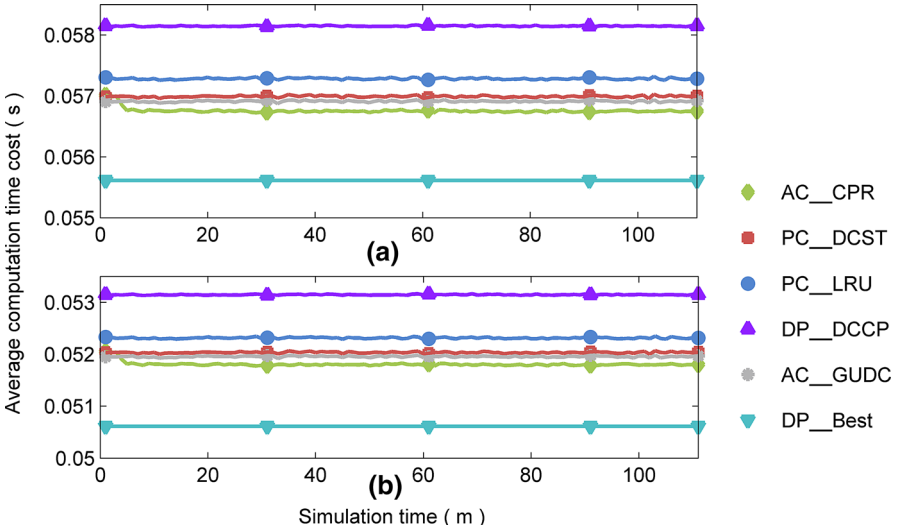

Fig. 9 Comparison of the average computation time costs obtained by different algorithms using 10 computation centers and the NLT Landsat-7 data set (**a**) and the SRTM90 data set (**b**)

The experiments show that the same algorithm provides different results for different data sets. This occurs because different data sets and different computational tasks require different data processing time costs to process the data, and the same algorithm will have a different average computation time cost based on Eq. (2). However, the performance of DP_Best provides a uniform reference standard. AC_CPR always performs better than DP_DCCP for the different data sets (Fig. 9).

4.2.2 Experiments using different parameters

As discussed in Sect. 3.3, the proposed active caching algorithm can cache multiple data during each computation scheduling period; therefore, the speed of data replacement can increase when the computing tasks change. Thus, an experiment was conducted to demonstrate the performance improvement using the proposed active caching algorithm with 10 computation centers. The experimental results for all of the algorithms are shown in Fig. 10.

Figure 10 shows both that performance improves by increasing the number of caching steps when the cache buffer size is relatively small and that this performance improvement can almost be neglected when the cache buffer size is sufficiently large. This occurs because a large cache buffer can store large amounts of data, and there is no need to delete cached data to save cache space. Thus, AC_CPR can cache multiple data to increase the data replacement speed at the beginning of system operation when the cache buffer size is small, and it only caches small amounts of data to reduce the computational complexity and scheduling times when the cache buffer size is large.

Moreover, the access to spatial data satisfies several intrinsic laws [26,27], which may change based on different users' behaviors or application tasks. To demonstrate the change in performance of AC_CPR and to validate the adaptability of the proposed method with different application behaviors, an experiment was performed using different distribution laws in which the distribution parameters vary significantly from approximately 0.600–0.950 [36]. The results are shown in Fig. 11.

As shown in Fig. 11 and considering the results in Fig. 6, for which the distribution parameter is 0.600, the performance of AC_CPR improves with an increase in the distribution parameter. This occurs because a larger distribution parameter represents

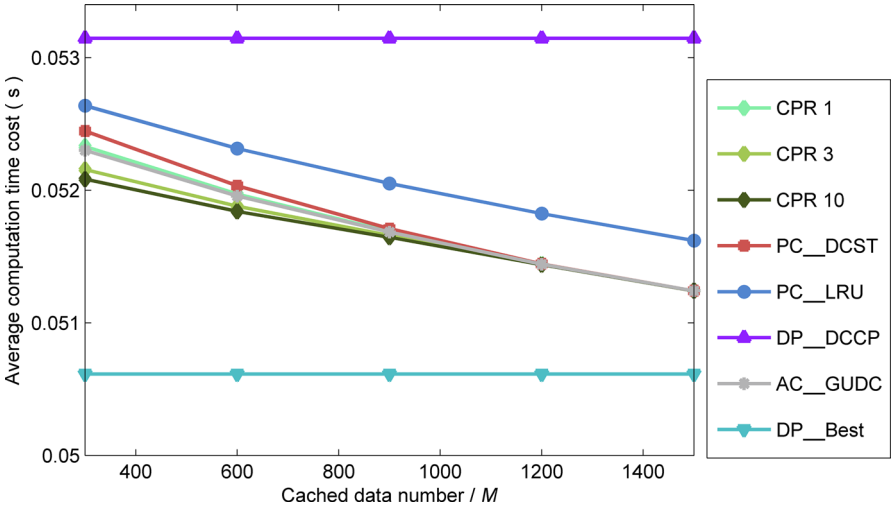

Fig. 10 Comparison of the average computation time costs obtained by different algorithms using 10 computation centers and 600–1500 pieces of cached data

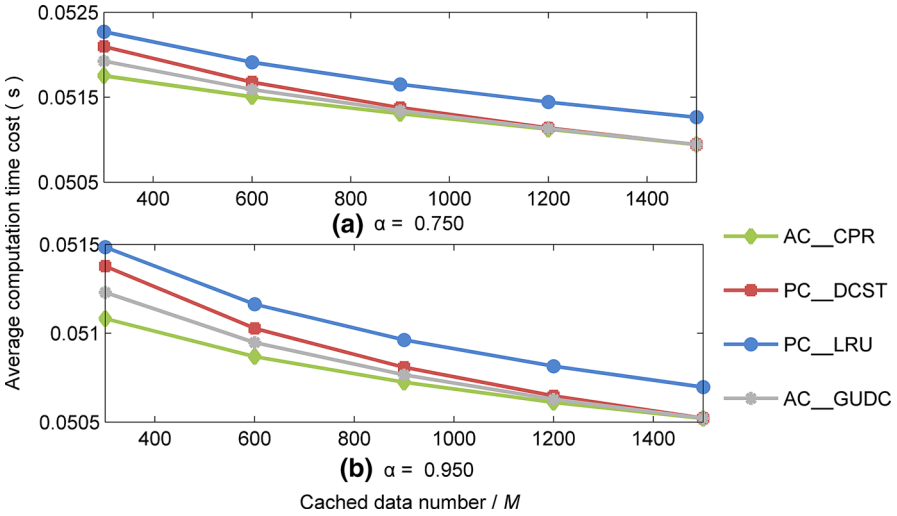

Fig. 11 Comparison of the average computation time costs obtained by different algorithms with 10 computation centers when the distribution parameter α varies from 0.750 to 0.950 and the cache buffer size varies from 300 to 1500

a more concentrated access distribution and therefore fewer data that will be used repeatedly must be cached. The results also show that the proposed algorithm can adapt to all kinds of application behavior and unlike data placement strategies, there is no need to adjust the algorithm's strategy when the computational task behavior changes. Thus, we can obtain the data's access distribution parameter by statistically computing the application's historical computation behavior dynamically. We can then obtain both a low computation time cost and a low computational and communication overhead by dynamically adjusting and using an appropriate cache buffer size based on the access distribution parameter. This strategy will be considered in future work.

Because different decay coefficients σ will lead to different amounts of historical access information and the use of different weights, an experiment was performed using decay coefficients from approximately 10–6000. The results are shown in Fig. 12.

As shown in Fig. 12, the performance of AC_CPR improves with an increase in the decay coefficient when the decay coefficient is less than 3000 because a larger decay coefficient indicates that more historical access records will be used; thus, the access correlation can be mined accurately. However, the use of too many records will reduce the effect of the timeliness, and some invalid features will be obtained. A greater number of records indicate a larger computational overhead; thus, decay coefficients of 15–30 are good choices to obtain higher average computation time costs and lower computational overhead when the number of computation centers is 10. The performances of all of the other algorithms remain stable, which indicates that the timeliness of the historical records has no effect.

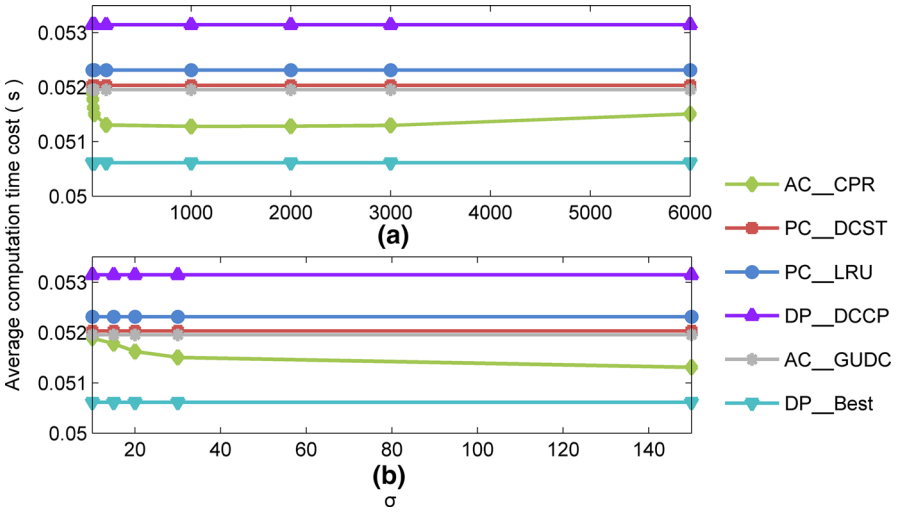

Fig. 12 Comparison of the average computation time costs obtained by different algorithms using 10 computation centers when the decay coefficient σ varies from 10 to 6000 (**a**) and from 10 to 150 (**b**)

4.3 Discussion

It is difficult for data placement strategies to synchronously adjust data distributions between storage nodes to meet the requirements of computational tasks caused by changes in applications, and active caching strategies can adapt to these dynamic characteristics by preparing data for computational tasks in advance. The experiments showed that the proposed algorithm can achieve better performance than other algorithms in all respects, can meet the requirements of large-scale distributed GIS and can adapt to dynamic environments. Computational performance can be further improved by using an appropriate cache buffer size and caching an appropriate amount of data during each computation scheduling period.

The proposed algorithm assumes that all of the storage nodes have the same storage capacity, all of the computation centers have the same computational capacity and transmission bandwidth, all of the data can be distributed to all of the storage nodes evenly, and each computation center can obtain data from any storage node in the same amount of time. However, some systems have different storage capacities and computational capacities; thus, the data placement strategy and active caching strategy should be combined to place the data in the appropriate storage node to reduce the total computation time cost and adapt to the computation centers' abilities. These issues will be considered in future studies.

5 Conclusions and future work

Instead of reading data from storage for computational tasks in real time, active prefetching and caching data from remote storage nodes through a network may be

used to significantly improve the quality of service and reduce the average computation time cost in distributed GIS. However, it is difficult to find the appropriate data to cache in advance because of massive data sets and the behaviors of different computational tasks.

This paper proposed an integrated algorithm for a data-caching strategy that is based on the computational tasks' historical behaviors, which imply timeliness relationships. The aim of CPR is to prepare and hold in the cache the data that are most likely to be computed immediately based on the cache buffer size. Due to the different cache buffer sizes, a flexible strategy can be used either to obtain high performance of the average computation time cost by caching more data when the cache buffer space is small or to reduce the computational complexity and scheduling times by caching only small amounts of data when the cache buffer size is sufficiently large.

The performance of the proposed method was demonstrated through a series of experiments. The results demonstrate that the proposed algorithm can provide better performance than other algorithms in all respects. The CPR can also be used in large-scale distributed GIS. Regardless of how the computing tasks are changed, the CPR can automatically adapt and obtain good performance.

In the future, the following areas of improvement can be considered: (1) differences between the servers' abilities and between the storage nodes are important factors that will significantly affect the average computation time cost and the algorithm's computational overhead and communication overhead; thus, a combined algorithm that considers different application behaviors and differences in the computation centers' abilities will be a focus of future work; (2) cache replacement is another important issue that must be researched further; and (3) metaheuristic algorithms such as the earthworm optimization algorithm (EWA) [37], the Monarch butterfly optimization (MBO), elephant herding optimization (EHO) and the moth search (MS) [38] algorithm can be used to reduce the complexity of finding all fixed data combinations to solve the problems and should be studied further.

Acknowledgements This work was partially supported by the National Natural Science Foundation of China (Grant Nos. 41671382, 41271398, 61572372 and 51277167), LIESMARS Special Research Funding and the Fund of SAST (Project No. SAST201425) and "CAST Innovation Fund": the Study of Agent and Cloud-Based Spatial Big Data Service Chain. The funders had no role in the study design, the data collection and analysis, the decision to publish or the preparation of the manuscript.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

1. Goodchild MF (1992) Geographical information science. Int J Geogr Inf Syst 6(1):31–45
2. Zhao LJ, Chen LJ, Rajiv R, Kim-Kwang RC, He JJ (2016) Geographical information system parallelization for spatial big data processing: a review. Cluster Comput 19:139–152
3. Wang L, Chen D, Hu Y, Ma Y, Wang J (2013) Towards enabling cyber infrastructure as a service in clouds. Comput Electr Eng 39(1):3–14

4. Wang T, Yao SH, Xu ZQ, Jia S (2015) DCCP: an effective data placement strategy for data-intensive computations in distributed cloud computing systems. J Supercomput. doi:10.1007/s11227-015-1511-z
5. Eidsvik J, Shaby BA, Reich BJ, Wheeler M, Niemi J (2014) Estimation and prediction in spatial models with block composite likelihoods using parallel computing. J Comput Graph Stat 23(2):295–315
6. Matthias K, Dorit H (2016) Parallel inference for massive distributed spatial data using low-rank models. Stat Comput. doi:10.1007/s11222-016-9627-4
7. Fuller SH, Millett LI (2011) The future of computing performance: game over or next level? Committee on sustaining growth in computing performance. National Research Council, Washington, DC
8. Yu B, Hao ZX (2010) Research of distributed and parallel spatial index mechanism based on dpr-tree. Comput Technol Dev 20(6):39–42
9. Zhao YC, Li CM, Zhao CY (2007) Research on the distributed parallel spatial indexing schema based on r-tree. Geogr GeoInf Sci 23(6):38–41
10. Zhao YC, Meng LK, Lin ZY (2006) Spatial data partitioning towards parallel spatial database system. Geomat Inf Sci Wuhan Univ 31(11):962–965
11. Foster I (1995) Designing and building parallel programs. Addison Wesley Publishing Company, Boston (reading)
12. Yao Y, Gao J, Meng L, Deng S (2007) Parallel computing of buffer analysis based on grid computing. Geospatial Inf 5(1):98–101
13. Pang L, Li G, Yan Y, Ma Y (2009) Research on parallel buffer analysis with grided based hpc technology. In: IEEE International Geoscience and Remote Sensing Symposium, IGARSS 2009, vol 4. p IV–200
14. Fernandez J, Marroquinguzman M, Wilson RA (2014) Optimization approaches to mpi and area merging-based parallel buffer algorithm. Boletim de Ciências Geodésicas 20(2):237–256
15. 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. doi:10.1371/journal.pone.0133029
16. Yang C, Wu H, Huang Q, Li Z, Li J (2011) Using spatial principles to optimize distributed computing for enabling physical science discoveries. Proc Natl Acad Sci 106(14):5498–5503
17. Jarukasemratana Sorn, Murata Tsuyoshi (2013) Web caching replacement algorithm based on web usage data. New Gener Comput 31(4):311–329. doi:10.1007/s00354-013-0404-z
18. Boulos MN (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(12):2269–2272
19. Shi Xuan, Kindratenko Volodymyr, Yang Chaowei (2013) Modern accelerator technologies for geographic information science. Springer, New York
20. Bell DG, Kuehnel F, Maxwell C, Kim R, Kasraie K, Gaskins T, Coughlan J (2007) NASA world wind: opensource GIS for mission operations. In: Aerospace Conference, pp 1–9
21. Yeşilmurat Serdar, İşler Veysi (2012) Retrospective adaptive prefetching for interactive web GIS applications. Geoinformatica 16:435–466. doi:10.1007/s10707-011-0141-8
22. Park D-J, Kim H-J (2001) Prefetch policies for large objects in a web-enabled GIS application. Data Knowl Eng 37:65–84 (ISSN: 0169-023X)
23. Lee DH, Kim JS, Kim SD, Kim KC, Kim Y-S, Park J (2002) Adaptation of a neighbor selection Markov chain for prefetching tiled web GIS data. In: Proceedings of the Second International Conference on Advances in Information Systems, vol 2457. pp 213–222, ISBN: 3-540-00009-7
24. Yunjin Li, Zhong E, Wang E, Huang Y (2010) Markov model in prefetching spatial data. Bull Surv Mapp 7:1–4
25. Rui Li, Guo Rui Xu, Zhenquan Feng Wei (2012) A prefetching model based on access popularity for geospatial data in a cluster-based caching system. Int J Geogr Inf Sci. doi:10.1080/13658816.2012.659184
26. 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. doi:10.1371/journal.pone.0170195
27. Hao Wang, Shaoming Pan, Ming Peng (2010) Zipf-like distribution and its application analysis for image data tile request in digital earth. Geomat Inf Sci Wuhan Univ 35(3):356–359
28. Xia Jizhe, Yang Chaowei, Liu Kai, Gui Zhipeng, Li Zhenlong, Huang Qunying, Li Rui (2015) Adopting cloud computing to optimize spatial web portals for better performance to support digital earth and other global geospatial initiatives. Int J Digit Earth 8(6):451–475. doi:10.1080/17538947.2014.929750

29. Rui Li, Wang X, Shi X (2014) A replacement strategy for a distributed caching system based on the spatiotemporal access pattern of geospatial data. ISPRS Int Arch Photogramm Remote Sens Spat Inf Sci 40(4):133–137
30. Sashi K, Thanamani A (2011) Dynamic replication in a data grid using a modified bhr region based algorithm. Futur Gen Comput Syst 27(2):202–210
31. 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
32. Tao Wang, Yao Shihong Xu, Zhengquan Pan Shaoming (2016) Dynamic replication to reduce access latency based on fuzzy logic system. Comput Electr Eng. doi:10.1016/j.compeleceng.2016.11.022
33. Xiong L, Xu Z, Wang H et al (2016) Prefetching scheme for massive spatiotemporal data in a smart city. Int J Distrib Sens Netw 2016(2):1. doi:10.1155/2016/4127358
34. Jianliang Liu, Lin Yang, Mingyang Guo, Lu Xu (2014) The relevance principle of I/O references. J Comput Res Dev 51(Suppl):48–56
35. D'Urso MG, Trotta S (2015) Comparative assessment of linear and bilinear prism-based strategies for terrain correction computations. J Geod 89(3):199–216
36. Krashakov SA, Teslyuk AB, Shchur LN (2006) On the universality of rank distributions of website popularity. Comput Netw Int J Comput Telecommun Netw 50(11):1769–1780
37. Wang G G, Deb S, Coelho L D S (2015) Earthworm optimization algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Int J BioInspired Comput
38. Wang Gai Ge (2016) Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. Memetic Comput. doi:10.1007/s12293-016-0212-3