Distributed cache replacement method for geospatial data using spatiotemporal locality-based sequence

Rui LI\textsuperscript{a,b}, Jiapei FAN\textsuperscript{a}, Xinxing WANG\textsuperscript{a}, Zhen ZHOU\textsuperscript{a} and Huayi WU\textsuperscript{a, b,}\*\\
\textsuperscript{a}State key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing (LIESMARS), Wuhan University, 129 Luoyu Road, Wuhan 430079, China; \textsuperscript{b}Collaborative Innovation Center of Geospatial Technology, Wuhan University, 129 Luoyu Road, Wuhan 430079, China\\

(Received 3 July 2015; final version received 18 October 2015)

Specific features of tile access patterns can be applied in a cache replacement strategy to a limited distributed high-speed cache for the cloud-based networked geographic information services (NGISs), aiming to adapt to changes in the access distribution of hotspots. By taking advantage of the spatiotemporal locality, the sequential features in tile access patterns, and the cache reading performance in the burst mode, this article proposes a tile sequence replacement method, which involves structuring a Least Recently Used (LRU) stack into three portions for the different functions in cache replacement and deriving an expression for the temporal locality and popularity of the relevant tile to facilitate the replacement process. Based on the spatial characteristics of both the tiles and the cache burst mode with regard to reading data, the proposed method generates multiple tile sequences to reflect spatiotemporal locality in tile access patterns. Then, we measure the caching value by a technique based on a weighted-based method. This technique draws on the recent access popularity and low caching costs of tile sequences, with the aim of balancing the temporal and spatial localities in tile access. It ranks tile sequences in a replacement queue to adapt to the changes in accessed hotspots while reducing the replacement frequency. Experimental results show that the proposed method effectively improves the hit rate and utilization rate for a limited distributed cache while achieving satisfactory response performance and high throughput for users in an NGIS. Therefore, it can be adapted to handle numerous data access requests in NGISs in a cloud-based environment.

Keywords: spatiotemporal; replacement; access pattern; Least Recently Used (LRU) stack; networked GIS

1. Introduction

With the development of virtual geographic environments, such as Google Earth, Digital City, and CyberTown, the increased activity in networked geographic information services (NGISs) has influenced the development of human–environment relationships (1, 2). Large volumes of tiled geospatial data (tiles) and intensive user requests for them impose a significant burden on NGISs. Some researches have shown that many users request the same data from NGISs, and the redundancy of the requested information causes network congestion and related issues, such as response delays, communication errors, and server overload. A distributed high-speed caching system (DHCS) can cache frequently accessed geospatial data, thereby reducing the bandwidth of input/output requests for data resources and response delays for public users, and obtaining higher cache hit rates and greater data sharing (3). A DHCS also benefits from system scalability and high throughput and is among the most commonly used methods of service acceleration for cloud-based NGISs. However, the core issue in DHCSs is the use of distributed caches of limited capacities to ensure that the most popular data are cached, since cached data change with the varying hotspots. A cache replacement strategy can address this issue.

Considerable research on page cache replacement strategies has been conducted since the tremendous rise in Web traffic beginning in the 1990s. A few companies, such as Esri, Google, and Microsoft, already introduced cache technology in their products. However, these companies employed traditional cache replacement algorithms, such as Least Recently Used (LRU) (4, 5), Least Frequently Used (LFU) (6), first in, first out (7), size-based replacement (8), and other varieties of these algorithms, including Hyper-G (9), size-adjusted LRU (10), greedy dual-size cache replacement (11, 12), LRU-K (13, 14), Lowest Relative Value (15, 16), and Least Grade Replacement (LGR) (17). These algorithms are based either on the temporal locality principle or on the size of the cached data. However, the access pattern of tiles exhibits both temporal and spatial characteristics, and tiles in a layer are identical in size. Therefore, these traditional cache replacement algorithms cannot be directly used in cache replacement methods involving tiles. In the context of NGISs, little research has been conducted on cache replacement methods, especially on DHCSs. Ref. (18) proposed replacing tiles beyond the user’s viewing area based on his/her roaming behavior. Ref. (19) calculated the transition probability of accessed tiles based on access logs, whereby tiles with

*Corresponding author. Email: wuhuayi@whu.edu.cn

© 2015 Wuhan University
the minimum transition probability values are replaced. A Markov model for choosing neighboring tiles has also been proposed (19). The model monitors a user’s k previous movements before arriving at the current tile and predicts his/her next movement in the replacement. However, these proposed cache replacement methods involve obtaining the access probability of a tile through system analyses or lengthy learning processes prior to choosing and replacing the tiles with minimum access probabilities. Access probability can be calculated and updated in near-real time for a tile in a geospatial data set. However, volume tiles in an NGIS can cause a heavy workload and increase the cost of calculating access probabilities.

The latest trend in NGIS research is to study and mine access patterns and modes of user interaction in big data (20–22). User access logs of several NGISs show that geospatial data access exhibits a spatiotemporal locality characteristic, whereby most access concentrates on hotspots that change over time (23–26). Temporal locality in tile access implies that the recently accessed tiles have a higher probability of being accessed again in the near future, whereas the spatial locality in tile access suggests that tiles adjacent to a currently accessed tile tend to take comparable access times (27). The consideration of spatial attributes of tiles, spatiotemporal locality in tile access patterns, and the development of a satisfactory cache replacement method based on users’ requested contents, which have comparable access times and adjacent spatiality, can significantly improve the hit rate and the utilization of a limited cache, and hence help obtain optimal or near-optimal network system performance.

However, temporal and spatial localities in tile access patterns are interrelated and interrestricted. Temporal locality reflects the short-term temporal aggregation of access patterns. A cache replacement method can use this feature to define the caching value as well as the validity of cached tiles. However, tiles also exhibit spatial locality, which enables them to establish long-term and fixed spatial relationships with one another. This feature ensures that tiles in a spatial area are accessed sequentially and locally and reflects local stability in tile access pattern. A cache replacement method can exploit this feature to reduce replacement frequency and maintain a certain degree of stability in the DHCS. The best approach to cache replacement for tiles is a “combination” method that considers and balances the temporal and spatial localities of tile access patterns by adapting to changes in the access distribution of hotspots while reducing the frequency of cache replacement. Such an approach can improve the utilization rate of caches and increase DHCS stability. Therefore, we propose in this article a distributed cache replacement method where the tile sequence exhibits distinct spatiotemporal access patterns. The method employs an LRU stack to derive an expression for the frequency and interval of tile access and ranks the tiles according to their latest popularity. By considering the spatial characteristics of both the tile and the cache burst mode in reading data, it then generates tile sequences to reflect spatiotemporal locality in tile access patterns. Moreover, based on the recent access popularity of the tile sequence and its caching cost, it balances the temporal and spatial localities in tile access to realize effective replacement. This results in a high cache hit rate and enables NGISs to improve the service capabilities for a large number of concurrent accesses.

The rest of this article is organized as follows: Section 2 introduces related work in the research area, including research on utilizing access patterns in cache replacement, the ranking mechanism in cache replacement, and caching reading mode for tile replacement. Section 3 contains a presentation of the replacement method for a DHCS based on tile sequences and the patterns of access of tiles which embody spatiotemporal locality and sequentiality. In Section 4, we present simulation results and analyses using the method proposed in Section 3. Section 5 concludes the article and proposes an outlook of our study.

2. Related work
2.1. Utilizing access patterns in cache replacement

Few cache replacement methods have been proposed in the field of NGIS. However, Google Earth has suggested the usefulness of a caching strategy in its cluster-based service (28). However, it has not yet published its detailed cluster-based caching strategies for commercial reasons. Tile access exhibits a locality feature, whereby access tends to be concentrated in a small area and displays certain temporal and spatial patterns. The temporal locality of tile access implies that a tile that is accessed once will be accessed again with a high probability in a short time. The access frequency of a tile, which is the interval between the current time and the time when the most recent user access to the relevant tile, is often employed to represent the temporal locality. LRU is a commonly used cache replacement algorithm based on temporal locality of access (4–7). Tiles that are accessed by users converge to a specific area on the map provided by the NGIS, which means that if a tile is accessed, its neighboring tiles (as well as that tile itself) have a high probability of being accessed.

This is known as spatial locality in tile access patterns. In a pyramid model of tiles, the distance between any tile and a tile being accessed at any given time is introduced to derive an expression for spatial locality in the relevant tile access pattern. However, the distance between accessed tiles is difficult to calculate in an NGIS because the tile at the center of the user’s browsing view changes quickly. Therefore, access times are typically used to measure the distance between tiles, as in the LFU cache replacement algorithm. Following up on LRU and LFU, many researchers proposed cache replacement algorithms (6, 16) that only focused on temporal locality or spatial locality in tile access patterns. Ref. (29) considered both temporal and spatial localities...
in proposing an algorithm based on tile access average interval time (TAIL). This algorithm chose the distance between cached tiles and the tile being accessed at any given time to express spatial locality, and the difference between the current time and the time when the most recent user access to a given tile occurred to express temporal locality. However, the algorithm used accumulated access times for a tile to indicate spatial locality in tile access, without considering a fixed spatial relationship between the tile and its neighboring tiles. Cache replacement aims to efficiently utilize the cache by storing data with high access probabilities. Many recent studies on cache replacement proposed methods based on tracking the access popularity of data, such as the recent usage frequency (RUF) caching policy (30) and the age-based cooperative caching (31). However, these methods depend on massive access histories and cannot quickly track changes in data access popularity. Furthermore, these methods are more computationally complex than LRU and LFU. Therefore, a cache replacement method for tiles that utilizes access patterns should not only reflect the natural geographical and spatiotemporal correlations among the accessed tiles, but also provide an effective mechanism to express the access popularities and caching values of tiles.

2.2. Caching value ranking mechanism in replacement

The caching value ranking mechanism is crucial to an effective replacement method. In general, there are a large number of tiles whereas the capacity of a DHCS is limited. When the memory blocks of a DHCS are occupied by previously accessed, outdated tiles, there is no space to cache the most recently accessed tile. Accordingly, the DHCS should apply a caching value ranking mechanism to delete tiles with low caching values from the memory blocks in order to clear cache space for recently accessed tiles. However, an effective ranking mechanism is critical to providing accurate caching values of tiles in an effective cache replacement method.

Spatiotemporal locality in tile access patterns allows a DHCS to build a suitable cache replacement method. At the same time, since spatiotemporal locality changes with popular hotspots over time, such a method would need to trigger a replacement algorithm to replace outdated hotspot tiles with new ones, which incurs additional cost and can render the DHCS unstable. Therefore, reducing replacement frequency as much as possible is another critical factor in an effective caching value ranking mechanism.

Under ideal conditions, data that will never be accessed again, or will not be accessed for a long time, should be removed from the cache (Optimal Page Replacement Algorithm, OPT) (32). LRU is closest to an OPT in terms of algorithm efficiency. Its various versions are widely used in applications. However, the basic idea underlying LRU is to replace the least recently used tile from the cache during replacement. LRU typically structures a stack to sequentially store accessed tiles from top to bottom and replaces tiles at the bottom of the stack during replacement. LRU reflects the short-term popularity of tile access. It considers the probability of a tile to be accessed again as the inverse ratio of the time interval between the current time and the time when the tile was last accessed. Thus, when ranking the access popularity of a tile in descending order in the stack, the most recently accessed tile is ranked at the top while the least recently accessed tile is ranked at the bottom. That is, the ranking accords with the most recent access time of each tile. LRU is simply implemented, incurs a small system overhead, and is effective. However, while LRU is appropriate for tiles for which the access patterns exhibit significant temporal locality, accessed tiles also exhibit geographical attributes, such as spatial correlation and sequentiality. Within a certain period, access to tiles has an aggregation feature and a spatial locality feature that displays a certain degree of stability in the access pattern. Classic LRU cannot reflect these access features for tiles, and hence cannot maintain stability in ranking and replacing cached tiles. Therefore, an effective caching value ranking mechanism should consider both spatial and temporal localities and balance them to provide the basis for a satisfactory cache replacement method, one that can adapt to changes in the access distribution of hotspots while reducing the frequency of cache replacement.

2.3. Cache reading mode for tile replacement

A cache strategy that matches the cache reading mode can boost performance. In general, the cache uses static random access memory (SRAM) for cached data distributed in a matrix. When the CPU retrieves data from the cache, the row decoder first fixes the row address, following which the column decoder fixes the column address. It thus retrieves the unique cell address (row and column addresses) of the requested cached data in memory and sends this address to the data bus through the RAM interface. In general, SRAM efficiently reads data in burst mode, where it locks a memory row when reading data and then quickly sweeps through all columns, simultaneously reading bits of data in the entire row. If the CPU retrieves requested data items that are side by side in memory (that is, they are in the same row but different columns), burst mode can improve memory reading efficiency.

A pyramid model provides a suitable management and cache method for geospatial data (33). In a pyramid model of tiled geospatial data, access to tiles exhibits a sequential feature. When a user navigates, the client application calculates the coordinates of the center tile of the browsing view at the time based on its longitude and latitude. It then requests a tile by providing its coordinates \((\ell(x, y))\) to the server, where \(x\) and \(y\) represent the coordinates of the relevant tile, and \(\ell\) represents the layer.
number in the tile pyramid model. The server returns multiple tiles to form the user’s browsing view at the time according to the request. Thus, tiles neighboring an accessed tile in the browsing view have similar access times and are sequentially accessed in the server. If tiles accessed in a sequence can be stored in consecutive memory spaces, a cache replacement method that considers a tile sequence to be a replaced object instead of a tile can attain high efficiency and low system cost by taking advantage of spatiotemporal locality and sequential features in tile access patterns as well as memory reading performance in burst mode. However, the generation and ranking of a tile sequence in the cache is an issue that needs to be addressed by a satisfactory cache replacement method for NGISs.

3. Methodology

The crucial aspect of a cache replacement method is how to reflect the spatiotemporal locality and sequentiality in patterns of tile access, and its use of the cache burst mode in efficiently reading data to accelerate the extraction of tile data. Such a method should ensure the stability of the DHCS at low cost while efficiently carrying out replacement. In this article, we develop a method to express the access temporal locality of a tile and its popularity based on an LRU stack. Since both the neighboring tiles and the cached data in memory are spatially correlated, we use the LRU stack to build multiple tile sequences, where tiles in the same sequence are locally spatiotemporally relevant. We then use the resource cost of a tile sequence and its recent access popularity as weights to facilitate effective replacement. Our method employs an LRU stack to reflect high temporal locality in tile access patterns and generates a tile sequence to reflect spatial relevance in tile access. In the replacement process, our method exploits the sequentiality of reading cached data as well as the caching cost of the tile sequence to balance temporal and spatial localities in order to reduce replacement frequency and attain a stable caching system.

3.1. Self-adaptive framework of DHCS

Geospatial data are inherently distributed, which need a reasonable distributed caching strategy to effectively use the limited cache. A DHCS can help in cloud enabling an NGIS, and the distributed framework can easily expand and improve service performance by adding new cache server nodes (34).

We propose a self-adaptive method for a DHCS, where each cache server runs independently to automatically request tasks based on its current service processing capability. In a DHCS, all cache servers are clustered. A cluster-based caching administrator (CCA) exists at the point of entry of a DHCS, as shown in Figure 1.

In this framework, the CCA maintains two lists. One is a cache-mapping list, Cache (S, T), where tile T is cached in server S. In this case, tiles are distributed and stored in the multiple cluster-based cache servers using simple key-value pairs to improve the efficiency of distributing tasks in the DHCS. The other list is a task request queue, Queue (S). When a cache server S sends a request to a CCA to assign a task to it according to its status and service processing capability, Queue (S) adds the request at its end. The CCA distributes request tasks one by one based on Queue (S). This task distribution method is self-adaptable and load-balancing. The flow of task distribution for the cluster-based caching service in the DHCS framework is as follows:

Step 1: The application server receives the user’s request for tile T and forwards the request to the CCA.

Step 2: The CCA searches for tile T in Cache (S, T). If it finds tile T (a cluster-based cache hit), the CCA adds the server that cached tile T to a temporary cache server list,Cached (s), which is passed to Step 3. Step 2 is looped until all servers that have cached tile T are found and added to list Cached (s). If the CCA does not find tile T (a cluster-based no-cache hit), it proceeds to Step 4.

Step 3: For each server in Cached (s), the CCA searches for it in Queue(S). If it finds a server St in Queue (S) (a server hit), it forwards the request for tile T to server St. The request task is thus completed. If no server is found in Queue (S) (a server no-hit), it proceeds to Step 4.

Step 4: The CCA forwards the requested task to Shead, the first cache server in Queue (S). Shead requests tile T from the cluster-based, object-based storage device (OSD).

Step 5: The cluster-based OSD system returns tile T to cache server Shead and the application server.

Step 6: The cache server Shead sends a request to the CCA to update the caching index of tile T in Cache (S, T). The cache server Shead executes the cache replacement algorithm if the number of currently cached tiles exceeds the replacement threshold value.

This distributed cluster-based caching approach is self-adaptive and cooperative. Based on the service processing capability and the cache capacity of each cache server, the method attempts to assign tiles with the highest access probabilities to available servers with the highest service processing capability to obtain the fastest access response time for these “hotspot” tiles. The method is a simple and efficient one for task partitioning and cache replacement.

3.2. Structure of an LRU stack

The proposed cache replacement method structures an LRU stack to take advantage of the spatiotemporal locality of tile access. Two key issues must be considered when structuring an LRU stack: the utilization of the LRU stack to express the spatiotemporal locality of and the spatial relationship among tiles, and the utilization of the LRU stack to balance the time cost (temporal
locality) and the space cost (spatial locality) of tile sequences in the cache replacement process. Thus, the proposed method should consider the function of the LRU stack as well as the method for the generation and ranking of tile sequences in the stack.

3.2.1. Components of LRU stack

The proposed method employs an LRU stack to embody the temporal locality of tiles and generates a tile sequence to embody their spatial locality and sequentiality. It divides an LRU stack into three portions for different functions. The first portion is the tile receiving pool, which receives the most recently accessed tiles, filters the ones with highest access probabilities, and adds them to the stack. It thus stores hotspots in this pool. The second portion is the tile sequence pool, which collects accessed tiles that are not popular and will be structured into sequences. The size of the pool ranges from 0 to BUFFER_MAX. When the memory occupied by the tiles in the sequence pool is BUFFER_MAX, the tiles are structured into tile sequences of varying lengths in the pool. The third portion of the LRU stack in our method is the replacement queue pool, which stores and ranks tile sequences to be replaced in the near future.

Therefore, based on an LRU stack structure, our method can implement spatiotemporal locality and sequentiality for tiles, generate tile sequences, and rank the sequences for the replacement operation.

3.2.2. Expression of tile access popularity

In general, an LRU stack reflects the temporal locality of tile access, which considers the most recent past as the immediate future. Based on the LRU in descending order, tiles that were recently accessed are ranked higher than those that were accessed earlier. The LRU stack also reflects the short-term popularity of tiles and stores the latest hotspots. The distance between the top of the stack and any given tile in the stack can be expressed as the “least recently accessed time interval.” Therefore, a tile with a lower least recently accessed time interval is more popular and will be accessed again with a high probability in a short time. If a tile is at the bottom of the stack, this means that it is less popular than any other tile in the stack and will have the highest least recently accessed time interval in the stack. Thus, the popularity of a tile is inversely proportional to its least recently accessed time interval as expressed in Equation (1) below.Replacing the tile with the highest least
recently accessed time interval value can simplify the replacement process and improve its efficiency.

$$\text{Popularity} = \frac{1}{\text{Least Recently Accessed Time Interval}} \quad (1)$$

### 3.3. Replacement method

#### 3.3.1. Replacement flow

The replacement flow includes maintaining the most recently accessed tile in the LRU stack and identifying its popularity, generating tile sequences, and ranking them, and triggering the replacement process under a certain condition. In a strict LRU stack, when a new request for a tile is received, the tile is always placed on top of the LRU stack to identify it as the most recently accessed one. If the tile is accessed before, it is moved to the top. However, this creates overhead due to frequent movement, and increased difficulty of tile sequence generation and ranking in both the sequence pool and the replacement queue. We propose a simple method to reduce tile movement in the stack using marks to indicate whether a tile is the most recent hotspot. The replacement flow of this method is shown in Figure 2.

**Step 1:** When a request arrives for tile B, if tile B is in the stack, its access flag is marked as NEW to indicate that it is the most recent hotspot, instead of moving it to the top of the stack. This is called a cache hit.

**Step 2:** If tile B is not in the stack, this is known as a cache miss. Then, if the receiving pool is not full, tile B is placed at the top; otherwise, a slot is emptied for tile B, and the algorithm proceeds to Step 3.

**Step 3:** The system checks tile C at the bottom of the receiving pool. If tile C’s access flag is NEW, it is moved to the top of the stack and its access flag is marked as OLD. Conversely, if tile C’s access flag is OLD, this means that tile C has the highest least recently accessed time interval value and is not recently accessed. Tile C is then moved to the sequence pool to create space for tile B in the receiving pool. Tile B is placed at the top of stack.

**Step 4:** When the size of the sequence pool is BUFFER_MAX, it is time to trigger tile sequence generation. The tiles in the sequence pool are structured into different sequences and moved to the replacement queue pool, where they are ranked.

**Step 5:** When the number of cached tiles exceeds the replacement threshold value, it is time to trigger cache replacement. Our method replaces tiles ranked at the bottom of the replacement queue pool with recently accessed tiles.

Thus, our method efficiently uses the LRU stack while reducing the movement of tiles.

#### 3.3.2. Tile sequence generation and ranking

In Section 2, we stated the need for a weighting method to calculate the caching value of a generated tile sequence in order to rank it. A satisfactory ranking method should replace the tile sequence with a lower caching value simply while making the replacement stably. Thus, in this article, we take into account both the caching cost and the recent access popularity in calculating caching value, and name it a combined caching cost and recent access popularity (CCR) weight.

Tile sequencing improves the reading efficiency of cached tiles. Tile sequencing places spatiotemporally related tiles in neighboring cache segments in order for them to be read with fewer CPU instructions than otherwise, thereby enabling a quicker response to users’ requests. The replaced tiles also exhibit spatiotemporal locality of access. However, our proposed replacement method focuses on structuring tile sequences accesses to

**Figure 2. Flow of replacement process.**
which exhibit spatiotemporal locality, and appropriately ranking tile sequences in the replacement queue based on their caching values. Accessed hotspots change over time, and users perform different operations for varying roaming reasons at different times. Therefore, the lengths of tile sequences that exhibit access spatiotemporal locality can vary. A tile with a high access frequency will cause tiles neighboring it to be accessed with high frequencies as well, due to which the accessed area will become greatly concentrated. The tile also has a more stable spatial relationship in access with neighboring tiles in a smaller spatial area. Thus, the generated tile sequence with a higher access popularity is shorter and has a lower caching cost than the one with a lower access popularity. Hence, by caching shorter tile sequences with higher access popularities and higher access spatial localities, our method will assist in the efficient use of a limited cache. Our proposed “Seq-Len cache-replacement algorithm” uses the length of a tile sequence as a weight to measure its caching cost. The length of the tile sequence is also an indicator of spatial locality. Function \( f(\text{caching cost}, RT) \) is used to calculate the caching cost for tile sequence RT. It is inversely proportional to the length of tile sequence RT, as Length (RT) is a function that determines the number of tiles in tile sequence RT, i.e. the length of sequence RT, as Equation (2).

\[
\begin{align*}
\text{Length} (RT) &= 1 / f(\text{caching cost}, RT) \\
&= \text{CCR}(\text{the most recently replaced tile sequence}) \\
&= f(\text{recent access popularity}, RT) + f(\text{caching cost}, RT)
\end{align*}
\] (4)

According to Ref (35), data with a higher caching cost should be replaced first and it is also considered as an optimal online caching replacement method (12). Longer tile sequences with lower caching costs are removed from the cache first. Meanwhile, to prevent less popular, shorter sequences from staying in the cache for a long time, the proposed method expresses the caching value using a temporal weighting function \( f(\text{recent access popularity}, RT) \) to measure the recent access popularity of tile sequence RT. As mentioned in Section 2.1, recent access popularity can be used to indicate temporal locality in tile access. That is, less popular, shorter sequences will have a lower recent access popularity value than more recent, more popular, and longer sequences, and will thus have a higher probability of removal from the cache. This method can thus prevent less popular, shorter sequences from remaining in the cache and causing caching pollution. To uniformly express the weight CCR of the caching value, the recent access popularity value for a generated tile sequence is determined by the CCR value of the most recently replaced tile sequence, as Equation (3). However, the CCR value of the most recently replaced tile sequence increases over time:

\[
\begin{align*}
\text{CCR} (\text{the most recently replaced tile sequence}) &= f(\text{recent access popularity}, RT) \\
&= \text{CCR}(\text{the most recently replaced tile sequence}) \\
&= f(\text{recent access popularity}, RT)
\end{align*}
\] (3)

When the Seq-Len method is triggered to generate tile sequences, it traverses the sequence pool and sorts tiles in the same cache row into a sequence according to the replacement flow mentioned in Section 3.3.1, and forms multiple tile sequences of varying length. For each sequence, Seq-Len also assigns a CCR value related to the recent access popularity and caching cost of the tile sequence. When a new sequence RT is added to the replacement queue, its CCR value is set as:

\[
\begin{align*}
\text{CCR} (RT) &= f(\text{recent access popularity}, RT) + f(\text{caching cost}, RT)
\end{align*}
\] (4)

Newly generated sequences are ranked based on their CCR value according to the sequences already in the replacement queue. Sequences with lower CCR values move to the bottom of the stack, to be replaced first. The tile sequences generated simultaneously in the sequence pool have the same values of recent access popularity but different values of caching cost when their CCR values are calculated.

The recent access popularity value identifies temporal locality, as in LRU, and the value of the caching cost identifies the access spatial locality. Longer sequences have lower access spatial localities and will have lower caching costs than shorter sequences generated simultaneously. Thus, longer sequences will be replaced earlier. However, the most recently accessed longer sequences will have higher values of recent access popularity and will be replaced after shorter sequences with lower recent access popularity values, which were accessed earlier in time. The function CCR (RT) considers and balances the temporal and spatial localities of sequences in the replacement queue. If the length of a sequence is 1 (that is, there is only one tile in the sequence), the Seq-Len algorithm is the identical to the LRU algorithm for a tile. After tile sequences are generated and moved to the replacement queue, the sequence pool is empty and ready to receive new tiles.

4. Simulation and analysis

4.1. Simulation design

4.1.1. Simulation environment

We conducted a user log-driven simulation and applied a discrete-event mechanism to simulate access requests in cluster-based servers. The simulation was performed in a network simulation environment. The network architecture is shown in Figure 1, and the network configuration and simulation parameters are set as listed in Table 1. It includes the network units (servers, clients, CCA, and network cloud) and their capacities (cache capacity, network bandwidth, processing power, etc.), the geospatial data and their access patterns, algorithms that were simulated and compared, and the simulation configuration (simulation time, random seeds, and results).

In the simulation, large-scale users accessed DHCS through a network cloud. Each user’s access bandwidth was 1 Mbps. A CCA with sufficient processing power was placed at the point of entry of the distributed system to prevent forwarding bottlenecks, as shown in Figure 1.
Distributed high-speed cache servers were connected using a 1000 Mbps switch to form a fast Ethernet. DHCS cache capability is an important factor in our proposed cache replacement method. The relative size of the cache is the ratio of the cache size to the total size of requested tiles. Therefore, we varied the relative size of the cache from 10 to 80% in our simulations to assess the performance of our method.

4.1.2. Simulation data
To evaluate the service capacity for a large number of user requests, we simulated to initiate users request for the tiles at the clients and made the pattern of arrival of access requests followed a Poisson distribution with the arrival rates according to the access peaks shown in Figure 3 (approximately 1000 to 2500 requests per second, based on user access logs from “TIANDITU”).

4.2. Simulation evaluation
Our simulations evaluated the performance of the proposed cache replacement method from two perspectives:

4.2.1. Simulations for the size of sequence pool
The size of the sequence pool is another important factor in generating tile sequences because it affects the efficiency of the cache replacement strategy. We varied the sequence pool size in our simulations to determine the appropriate size. Figure 4 shows the average response times for different relative sizes of caches for varying sequence pool sizes. It shows that the optimal sequence pool size is 20% of the relative size of the cache. If the sequence pool size is too small, the strategy cannot generate longer tile sequences and, therefore,
cannot take the physical advantages of reading data from cache to improve system server efficiency. If the sequence pool size is long, generating and sorting tile sequences, as well as filling the replacement queue pool, requires more time, and the shorter tile sequences are prematurely replaced. Furthermore, the strategy cannot take advantage of the sequencing and continuous caching. Therefore, for all remaining simulations in this experiment, the sequence pool size was set to 20% of the relative size of the cache.

4.2.2. Caching performance evaluation

Cache hit rate is often used to determine whether cache replacement can adapt to hotspot changes, especially given a small cache. If the cache replacement method has a higher cache hit rate, the method is highly sensitive to the popularity of accessed tiles and its mechanism for ranking tiles according to access popularity is simple and effective. It also indicates that the cache replacement method can satisfactorily use limited cluster-based caching to reflect access aggregation for a large number of access requests.

Cache hit rate is the ratio of the requested tiles retrieved from the cache and the total number of the requested tiles, as shown in Equation (5):

$$\text{Cache hit rate} = \frac{\text{tiles retrieved from cache}}{\text{total number of tiles requested}}$$  \hspace{1cm} (5)

Figure 5 shows a comparison of the tile request cache hit rates of the five aforementioned methods for different cache sizes. The cache hit rate of any method increased with cache size. RUF (using frequency of tile access) always recorded the worst cache hit rate, whereas LRU (using temporal locality of tile access) was superior to LFU (using spatial locality of tile access). TAIL and Seq-Len (using both temporal and spatial localities of tile access) were superior to RUF, LFU, and LRU. However, Seq-Len performed slightly better than TAIL. This indicates that access to tiles in NGIS is context-relevant, especially for tiles in temporal and spatial contexts. RUF with just access frequency, for instance, could not reflect this context relevance, and thus it exhibited the worst performance in terms of cache hit rate. However, temporal locality in tile access (as in LRU) reflects access popularity and context relevance better than spatial locality (as in LFU) because a tile and its neighbors have similar access times, and accesses to them are aggregative. In an analogous manner, using both temporal and spatial localities (Seq-Len and TAIL) can improve caching performance to a greater extent than temporal or spatial locality only.

The results show that the cache hit rate of Seq-Len is 15–40% higher than that of RUF, 3–15% higher than that of LRU, 5–24% higher than that of LFU, and 1–6% higher than the cache hit rate of TAIL. In particular, using a small cache (relative size of the cache less than 30%), Seq-Len performs 5–50% better than the other methods. It means that Seq-Len can accommodate hotspot changes and reflect changes in access popularity better than RUF because it has better convergence. With regard to reflecting temporal and spatial localities, Seq-Len can utilize a limited distributed cache better than LFU, LRU, and TAIL. Although both Seq-Len and TAIL consider temporal and spatial localities in tile access patterns, they apply the different strategies. Seq-Len creates tile sequences by taking advantage of spatiotemporal locality and sequential features in tile access patterns, whereas TAIL employs the accumulated access time for a tile to calculate the spatial locality of tile access. However, Seq-Len is as simple and effective as LRU. It can reduce algorithmic complexity and system cost and can concurrently replace multiple tiles to improve replacement efficiency.

4.2.3. Service performance for intensive access

4.2.3.1. Average request response time. Cluster-based caching aims to improve concurrent service capability and reduce response time for intensive user access. The average request–response time indicates the performance
of a distributed cluster-based system in case of a large number of user requests. It is the average value of the response time to all requests and indicates the data service performance of NGISs.

Figure 6 shows that the average request response time of any method decreases with an increase in cache size. RUF records the worst performance, LRU has a similar average request response time to that of LFU, Seq-Len exhibits the best average request response time, and TAIL records the second-best response time. Seq-Len’s average request response time is 50–180 ms less than that of TAIL, 100–230 ms less than that of LRU, 100–280 ms less than that of LFU, and 150–370 ms less than that of RUF. In particular, given a small cache (relative size of the cache $\leq 40\%$), Seq-Len’s average request response time is 200–300 ms less than the other methods because it does not continually change in the face of continually changing hotspots over time. This indicates that Seq-Len is more stable and robust than the other methods because it determines the hotspot changes for a tile sequence rather than for a tile, and considers and balances the temporal and spatial localities for a tile sequence in the replacement process. On the contrary, RUF and TAIL, for instance, judge the hotspot changes based on the accumulated access times for a given tile.

These results also indicate that Seq-Len can accelerate access response for users by exploiting the physical advantages of cache in reading data. Furthermore, because Seq-Len caches and replaces tiles that are spatially correlated in a sequence, it follows the navigation mode of tiles (whereby once a tile has been requested, the server will send a tile sequence whose center is the requested tile). Thus, Seq-Len always provides the fastest tile return for any cache size.

4.2.3.2. Throughput. Higher throughput indicates that the user will have a better roaming experience when an NGIS deals with the intensity of concurrent access, particularly in a cloud-based environment. It also indicates the workload capacity and service capacity of NGISs. Figure 7 shows that the throughput of any method increases with increase in cache size. For all cache capacity conditions, Seq-Len’s throughput is 10–25%, 9–15%, 9–14%, and approximately 5–12% higher than that of RUF, LFU, LRU, and TAIL, respectively. Seq-Len is able to utilize most of the bandwidth with increasing cache size. When the relative size of the cache is up to 40%, Seq-Len’s utilization of bandwidth is up to 80–96%. This indicates that Seq-Len can reflect the accessed aggregative tiles in time as well as tiles popular at any given time, which enables it to translate more data per unit interval than other methods. Furthermore, Seq-Len caches tiles are spatiotemporally associated and exhibit sequentiality in access patterns. It enables Seq-Len to extract data more quickly than other methods using sequentiality in cache reading. Thus, Seq-Len can accommodate access to current hotspots by handling more access traffic per unit time in a cloud-based NGIS.

4.2.3.3. Maximum number of concurrent. NGISs involve dealing with intensity of concurrent access, particularly in a cloud-based environment. A higher maximum number of concurrent can support high-intensity tile requests. The maximum number of concurrent is the number of concurrent requests that a NGIS can accept at a unit time. It indicates the concurrent
processing performance of a NGIS in a cluster-based environment. The distribution of access rush hours is shown in Figure 3, where the access arrival peak is approximately 2500 requests per second according user access logs. Figure 8 shows the maximum number of concurrent of a single server in the cluster-based environment under different cache capacity conditions. The maximum number of concurrent of any method increased with increase in cache size. For all cache capacity conditions, as shown in Figure 8, Seq-Len’s maximum number of concurrent is 40–53%, 25–30%, 10–25%, and approximately 5–8% higher than that of RUF, LFU, LFU, and TAIL, respectively. In particular, given a small cache, Seq-Len performs well.

Repeated access to the same tiles exhibits access bias. Seq-Len takes advantage of this characteristic to reduce the reading cost for tiles, thereby improving service capacity for concurrent access. When the relative size of the cache was 60%, Seq-Len successfully replied to all of the 2500 requests per second with six cluster-based servers as designed in the simulation.

5. Conclusions
Tile access exhibits sequential and spatiotemporal locality due to the spatial attributes of the tiles. In this article, we proposed a distributed cache replacement method for geospatial data, which considers the fixed characteristics of tile access patterns, uses the physical characteristics of cache burst mode in reading data, generates the tile sequences with spatiotemporal localities based on an LRU stack, balances the temporal and spatial localities in tile access patterns through a weighted method, and ranks the tile sequences in the replacement queue for an effective cache replacement strategy. Experiment results show that our proposed method outperforms the classic cache replacement methods that utilize access patterns.

Furthermore, the size of the replacement queue is an important factor to consider in a cache replacement strategy because it affects replacement frequency and the threshold value and is closely related to the size of the tile replacement sequence in the replacement method proposed here. Therefore, in future, we will investigate the relationship between dynamic changes in the replacement queue and the tile replacement sequence to make our method adapt quickly to popularity changes and obtain a better performance for NGISs.

Acknowledgments
The authors would like to thank the National Geomatics Center of China and “TIANDITU”.

Funding
This work was supported by the National Natural Science Foundation of China [grant number 41371370] and the National Basic Research Program of China [grant number 2012CB719906].

Notes on contributors
Rui Li is an associate professor in the State key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing (LIESMARS), Wuhan University. Her main research interests include spatiotemporal computing and data mining, and Web GIS.

Jiabei Fan is currently a master’s degree candidate of the LIESMARS, Wuhan University. She majors in spatial data mining and Web GIS.

Xinxing Wang is a postgraduate of the LIESMARS, Wuhan University. He majors in parallel and distributed computing, and networking communication.

Zhen Zhou is currently a master’s degree candidate of the LIESMARS, Wuhan University. He majors in spatial data mining and Web GIS.

Huayi Wu is a full professor and associate director of the LIESMARS, Wuhan University. His research interests include geospatial data sharing and service, geospatial data mining, and geographic information engineering and applications.

References
(1) Gong, J.H. Man-Earth Relationships Based on Virtual Geographic Environments. In Proceedings of 6th National Conference Cartography GIS Conference, Wuhan, Hubei, China, 2006.
(2) Li, D.R.; Shan, J.; Shao, Z.F.; Zhou, X.R.; Yao, Y. Geomatics for smart cities – concept, key techniques, and applications. Geo-spatial Inf. Sci. 2013, 16 (1), 13–24.
(3) Qin, X.; Zhang, W.; Wang, W.; Wei, J.; Zhong, H.; Huang, T. On-line Cache Strategy Reconfiguration for Elastic Caching Platform: A Machine Learning Approach. In Proceedings of the IEEE 35th Annual Computer Software and Applications Conference, Munich, German, Jul 18–22, 2011; pp 523–534.
(4) Boulos, M.N.K. 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. 2005, 4 (12), 2269–2272.
(5) Bell, D. G.; Kuehnel, F.; Maxwell, C.; Kim, R.; Kasraie, K.; Gaskins, T.; Hogan, P.; Coughlan, J. NASA World Wind: Open Source GIS for Mission Operations. In Proceedings of 2007 IEEE Aerospace Conference. IEEE: Big Sky, Montana, 2007; pp 1–9.
(6) Lee, D. Choi, J.; Kim, J.H.; Noh, S.H.; Min, S.L.; Cho, Y.; Kim, C.S. LRFU: a spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE Trans. Comput. 2001, 50 (12), 1352–1361.
(7) Dan, A.; Towsley, D. Approximate Analysis of the LRU and FIFO Buffer Replacement Schemes. In Proceedings of 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Boulder, USA, May 22–25, 1990; pp 143–152.
(8) Podlipnug, S.; Böszörmenyi, L. A survey of web cache replacement strategies. ACM Comput. Surv. 2003, 35 (4), 374–398.
(9) Wong, K.Y. Web cache replacement policies: a pragmatic approach. IEEE Network 2006, 20 (1), 28–34.
(10) Cheng, K.; Kambayashi, Y. LRU-SP: A Size-adjusted and Popularity-aware LRU Replacement Algorithm for Web Caching. In Proceedings of the IEEE 24th Annual International Computer Software and Applications Conference, Taipei, October 25–27, 2000; pp 48–53.
(11) Cao, P.; Irani S. Cost-aware WWW Proxy Caching Algorithms. In Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems, Monterey, CA, December 1997; pp 193–206.

(12) Jin S.; Bestavros, A. Popularity-aware Greedy Dual-size Web Proxy Caching Algorithms. In Proceedings of the IEEE 20th International Conference on Distributed Computing Systems, Taipei, April 10–13, 2000; pp 254–261.

(13) O’Neil, E.J.; O’Neil, P.E.; Weikum, G. The LRU-K page replacement algorithm for database disk buffering. ACM SIGMOD Rec. 1993, 22 (2), 297–306.

(14) O’Neil, E.J.; O’Neil, P.E.; Weikum, G. An optimality proof of the LRU-K page replacement algorithm. J. ACM 1999, 46(1), 92–112.

(15) Cakiroglu, S.; Arikan, E. Replace Problem in Web Caching. Proceedings of IEEE Symposium on Computers and Communications. June 30–July 3, 2003; pp 425–430.

(16) Padmapriya, V.; Thenmozhi, K. Web caching and response time optimization based on eviction method. Int. J. Innovative Res. Sci. Eng. Technol. 2013, 2 (4), 1171–1177.

(17) Tu, S.; He, X.F.; Li, X.F.; Ratcliiff, J.I. A Systematic Approach to Reduction of User-perceived Response Time for GIS Web Services. In Proceedings of the 9th ACM International Symposium on Advances in Geographic Information Systems. Atlanta, GA, May 28–June 1, 2001; pp 47–52.

(18) Kang, Y.K.; Kim, K.C.; Kim, Y.S. Probability-based Tile Pre-fetching and Cache Replacement Algorithms for Web Geographical Information Systems. Proceedings of Advances in Databases and Information Systems. Vilnius, Lithuania, September 25–28; Springer: Berlin, 2001; pp 127–140.

(19) Wang, F. Design and Implementation of Web-based GIS for Forest Fragmentation Analysis. Dissertation, West Virginia University: Morgantown, WV, 2002.

(20) Fisher, D. How We Watch the City: Popularity and Online Maps. In: Proceedings of Workshop on Imaging the City, ACM CHI 2007 Conference. San Jose, CA, USA, April 28–May 3, 2007.

(21) Liu, X.H.; Han, J.Z.; Zhong, Y.Q.; Han, C. D.; He, X. B. Implementing WebGIS on Hadoop: A Case Study of Improving Small File I/O Performance on HDFS. In Proceedings of IEEE International Conference on Cluster Computing and Workshops. New Orleans, LA, August 31–September 4, 2009; pp 1–8.

(22) Xia, J.; Yang, C.; Gui, Z.; Liu, K.; Li, Z. Optimizing an index with spatiotemporal patterns to support GEOSS Clearinghouse. Int. J. Geogr. Inf. Sci. 2014, 28 (7), 1459–1481.

(23) Talagala, N.; Asami, S.; Patterson, D.; Futernick, B.; Hart, D. The art of massive storage: a web image archive. Computer 2000, 33 (11), 22–28.

(24) Fisher, D. Hotmap: looking at geographic attention. IEEE Trans. Visual. Comput. Graphics 2007, 13 (6), 1184–1191.

(25) Wang, H.; Pan, S.M.; Peng, M.; Li, R. Zipf-like distribution and its application analysis for image data tile request in Digital Earth (in Chinese). Geomatics Inf. Sci. Wuhan Univ. 2010, 35 (3), 356–359.

(26) Yang, C.W.; Goodchild, M.; Huang, Q.Y.; Nebert, D.; Raskin, R.; Xu, Y.; Bambucus, M.; Fay, D. Spatial cloud computing: how can the geospatial sciences use and help shape cloud computing? Int. J. Digital Earth 2011, 4 (4), 305–329.

(27) Li, R.; Wang, X.X.; Wang, J.J.; Wu, H.Y. Simulation and Analysis of Cluster-based Caching Replacement Based on Temporal and Spatial Locality of Tile Access. In Proceedings of Modern Accelerator Technologies for Geographic Information Science. Springer Science Business Media: New York, 2013; pp 169–181.

(28) Butler, D. Virtual globes: the wide-world. Nature 2006, 439 (7078), 776–778.

(29) Wang, H.; Yu, Z.W.; Zeng, W.; Pan, S.M. massive spatial data cache replacement policy based on tile lifetime and popularity (in Chinese). Geomatics Inf. Sci. Wuhan Univ. 2009, 34 (6), 667–670.

(30) Kang, S.J.; Lee, S.W.; Ko, Y.B. A Recent Popularity Based Dynamic Cache Management for Content Centric Networking. In Proceedings of the 4th International Conference on Ubiquitous and Future Networks, IEEE: Phuket, Thailand, July 4–6, 2012; pp 219–224.

(31) Ming, Z.X.; Xu, M.W.; Wang, D. Age-based Cooperative Caching in Information – Centric Networks. In Proceedings of IEEE International Conference on Computer Communications. Orlando, FL, USA, March 25–30, 2012; pp 268–273.

(32) Sun, Z.X. Operating System Tutorial (in Chinese); Higher Education Press: Beijing, 2004.

(33) Li, R.; Zhang, Y.F.; Xu, Z.Q.; Wu, H.Y. A Load-balancing method for network GISs in a heterogeneous cluster-based system using access density. Future Gener. Comput. Syst. 2013, 29 (2), 528–535.

(34) Yang, C.W.; Wong, D.W.; Yang, R.; Kafatos, M.; Li, Q. Performance-improving techniques in web-based GIS. Int. J. Geogr. Inf. Sci. 2005, 19 (3), 319–342.

(35) Young, N. The K-server dual and loose competitiveness for paging. Algorithmica 1994, 11 (6), 525–541.