RESEARCH ARTICLE

# Distributed Storage Algorithm for Geospatial Image Data Based on Data Access Patterns

**Shaoming Pan**[1,3], **Yongkai Li**[2], **Zhengquan Xu**[1,3]*, **Yanwen Chong**[1]

**1** State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, Wuhan, Hubei, China, **2** Computer School of Wuhan University, Wuhan, Hubei, China, **3** Collaborative Innovation Center for Geospatial Technology, Wuhan, Hubei, China

* xuzq@whu.edu.cn

**Citation:** 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

**Editor:** Matjaz Perc, University of Maribor, SLOVENIA

**Received:** March 13, 2015

**Accepted:** June 23, 2015

**Published:** July 16, 2015

**Data Availability Statement:** All relevant data is available via Figshare (http://dx.doi.org/10.6084/m9.figshare.1416051).

**Funding:** This research was sponsored by the National Nature Science Foundation of China (Project No. 41271398) (SMP), the National Key Basic Research and Development Program of China (Project No. 2011CB302306) (ZQX), the Fundamental Research Funds for the Central Universities (Project No. 2042014kf0263 (SMP), 2042014kf0242 (YWC)) and supported by the Fund of SAST (Project No. SAST201425) (YWC). The funders had no role in study design, data collection

## Abstract

Declustering techniques are widely used in distributed environments to reduce query response time through parallel I/O by splitting large files into several small blocks and then distributing those blocks among multiple storage nodes. Unfortunately, however, many small geospatial image data files cannot be further split for distributed storage. In this paper, we propose a complete theoretical system for the distributed storage of small geospatial image data files based on mining the access patterns of geospatial image data using their historical access log information. First, an algorithm is developed to construct an access correlation matrix based on the analysis of the log information, which reveals the patterns of access to the geospatial image data. Then, a practical heuristic algorithm is developed to determine a reasonable solution based on the access correlation matrix. Finally, a number of comparative experiments are presented, demonstrating that our algorithm displays a higher total parallel access probability than those of other algorithms by approximately 10–15% and that the performance can be further improved by more than 20% by simultaneously applying a copy storage strategy. These experiments show that the algorithm can be applied in distributed environments to help realize parallel I/O and thereby improve system performance.

## Introduction

Declustering is one of the most effective methods in the field of parallel I/O and can be widely used to improve system performance by splitting and distributing large files among multiple storage nodes to speed up access to data. The Google file system (GFS) is a well-known distributed file system in which each large file is divided into several blocks of fixed size. Each block (approximately 64 megabytes (MB)) is then stored in multiple different storage nodes to enhance concurrency and system performance [1]. Moreover, a number of other similar systems, such as RAID (Redundant Array of Independent Disks) systems [2] and geospatial information systems (GISs) [3], have been developed, all of which use declustering technologies for the distributed storage of large files.

and analysis, decision to publish, or preparation of the manuscript.

**Competing Interests:** The authors have declared that no competing interests exist.

However, it is clearly imperative that we be able to store not only large files but also small files. With the rapid development of geospatial information technology and the widespread application of the Digital Earth system [4], an increasing number of small image files, most less than 64 MB in size, are being produced [5].

In fact, large amounts of small geospatial image files are currently stored in the Digital Earth system. Based on the multi-resolution pyramid approach to global satellite remote sensing images, remote sensing images are divided into image files of different resolution ratios, and each file is typically less than 64 MB. Examples of such systems include World Wind, Google Earth, Microsoft TerraServer [6], and the NASA Earth Observing System [7]. World Wind divides remote sensing images into small files, and these files are typically less than 1 MB in size [8,9]. Google Earth performs a similar type of processing; it splits files into slightly larger files, but the file sizes remain below 64 MB [10,11].

However, conventional declustering technologies, which play an important role in the field of distributed storage, still encounter difficulties in handling large numbers of small files [12], and further research on this issue is required [13]. To this end, a technology for the merging of small files has been proposed [14]. In the field of data storage, merging technologies are primarily used to reduce the numbers of files and the size of their metadata. HDWebGIS (WebGIS based on Hadoop) [15] is one typical example that is based on a proposed merging method that organizes and merges small files that are associated with similar spatial locations together into a single large file and then creates an index that is used to access the individual small files through middleware. Likewise, with the diffusion and application of cloud technology, the Hadoop distributed file system (HDFS), as one of the most prominent distributed file systems currently extant, must solve the problem of small file storage. Dong divides the small files that are stored on HDFS into three categories: structurally related, logically related and independent files [16]. Structurally related or logically related small files can be merged together and stored as a single large file to improve the performance of HDFS. Unfortunately, however, the cited study provides only a basic criterion for such merging; no specific method for merging small files based on their relationship is proposed.

Most previous studies have considered only the combination of small files into larger ones, followed by the distributed storage of each merged large file base on RAID technology. In fact, however, a particular block must be found and read from storage when a certain small file is requested, and this block cannot be prefetched when many requests for small files that belong to different merged files are issued simultaneously. Moreover, this process cannot be run in parallel, even when the small files are stored in the same storage node.

Given these challenges, this paper employs several strategies to organize and store small geospatial image data files into storage nodes in an attempt to optimize I/O parallelism performance in distributed environments. In this context, it is very important to understand, analyze and estimate the relationships among geospatial image data files that have a high probability of being requested simultaneously. To accomplish this goal, we analyze the data access patterns (**DAPs**) of geospatial image data files, which imply the relationships among these files, and then propose a new method of distribution on these DAPs to ensure that related small files (files with a higher probability of being requested simultaneously) are stored in different storage nodes to facilitate parallel requests.

## Overview of DAPs

DAPs are widely used in various fields for prefetching and caching [17]. James designed and implemented a Probability Graph (PG) to automatically predict future accesses based on DAPs, thereby greatly reducing the required cache size [18]. Thomas also proposed a

Partitioned Context Modeling (PCM) approach, which was developed based on graph-based modeling, to improve the accuracy of predicting the next file to be accessed [19].

Hotmap is a typical DAP analysis system that is designed to analyze geospatial data access patterns (G-DAPs) based on the historical access log information produced by a GIS after a long period of operation (the server records the information related to geospatial image data files, such as the file name and image location, in chronological order when end users (clients) request image files from the server) [20]. According to the Hotmap model, a G-DAP satisfies Zipf's law [21], which is described by Eq (1):

$$F_i = \theta / i^{\alpha} \tag{1}$$

where $F_i$ is the number of accesses to the $i$th geospatial data file, $\theta$ is a constant and $\alpha$ is a parameter of Zipf's law. Zipf's law indicates which geospatial data files will be accessed more frequently. A number of advancements in caching and prefetching based on G-DAP utilization have been reported [22–24].

As mentioned by Thomas, DAPs can also be used to organize and adjust disk layouts [19]. Therefore, a new algorithm must be developed to solve the problem of small file storage in GIS applications.

## Distributed Storage of Geospatial Data Based on DAPs

By analogy with the Random distributed Storage Algorithm (**RSA**) and the Location-based distributed Storage Algorithm (**LSA**), which have been employed by several researchers [15,16], we refer to the algorithm proposed in this paper as the Access Pattern-based distributed Storage Algorithm (**APSA**). Table 1 summarizes the different storage strategies used in these algorithms.

### 3.1 Description of APSA

We first provide some basic definitions of objects used by the algorithm.

First, let $F = \{f_1, f_2, \ldots, f_N\}$ be the set of natural files, which includes all of the original small geospatial image data files (for brevity, we henceforth refer to small geospatial image data files simply as small files). Each element in $F$ is labeled with a natural number [1, $N$], and $N$ is the total number of small files. The natural numbers [1, $N$] can then be defined as a natural file vector $I_o = (1,2,\cdots, N)$ based on the natural sequence of these files.

Let $C = \{c_1, c_2, \ldots, c_m\}$ denote the set of storage nodes, where $m$ is the total number of all storage nodes. For simplicity, each of the $N$ small files is stored in $m$ storage nodes on average (an uneven grouping can be transformed into an even grouping by copying select small files that have higher request rates; this process is demonstrated in section 4, and a related experiment is detailed in section 5.4).

Finally, let $\tilde{F} = \{\tilde{F}_1, \tilde{F}_2, \ldots, \tilde{F}_m\}$ be the set of grouped storage files. Each group of small files will be stored in one of the storage nodes, and each small file in $F$ will belong to one and only one group. In other words, the element $\tilde{F}_i = \{\tilde{f}_{i1}, \tilde{f}_{i2}, \ldots, \tilde{f}_{in}\}$ in $\tilde{F}$ is a group of $n$ small

**Table 1. Storage strategies used by various algorithms.**

| Algorithms | Storage strategies |
|---|---|
| RSA | Randomly |
| LAS | Based on their locations |
| APSA | Based on their relationships |

doi:10.1371/journal.pone.0133029.t001

files that will be stored in the $i$th storage node, $c_i$. Here, $m \times n = N$, and $n$ is called the storage length.

Furthermore, for $\forall \tilde{F}_i \in \tilde{F}$ $(i \in [1, m])$, if we assume that the small files $\tilde{f}_{i1}, \tilde{f}_{i2}, \ldots, \tilde{f}_{in}$ are labeled by $t_{i1}, t_{i2}, \cdots, t_{in}$ in $F$ and are denoted by $T_i = (t_{i1}, t_{i2}, \cdots, t_{in})$ $(t_{ij} \in [1, N], i \in [1, m], j \in [1, n])$, then $T = (T_1, T_2, \cdots, T_m)^T = (t_{ij})_{m\times n}$ defines the map from set $F$ to set $\tilde{F}$, and each $T_i$ is a restriction of $T$ on $\{f_{t_{i1}}, f_{t_{i2}}, \ldots, f_{t_{in}}\}$. Therefore, the map $T : F \Rightarrow \tilde{F}$ denotes a storage distribution rule that defines how the small files are assigned to storage nodes on average. If the storage distribution vector is denoted by $I=(t_{11},t_{12},\cdots,t_{1n},t_{21},t_{22},\cdots,t_{2n},\cdots,t_{m1},t_{m2},\cdots,t_{mn})$, then the **APSA** key can be converted to construct an $N \times N$ permutation matrix, $B$, that satisfies the condition $I = I_oB$. We can then achieve our goal of distributing all small files across all storage nodes on average.

## 3.2 Access correlation matrix

To construct $B$ (or to find an optimal $T$), we must analyze the historical access log information [20], which reflects the small files' access patterns and can be used to compute the relationships among the small files.

Let $R = \{f_{a_1}, f_{a_2}, \ldots, f_{a_M}\}$ denote the chronological access sequence of small files that is obtained from the historical access log information recorded by the server of the Digital Earth system. The vector $A = (a_1, a_2, \cdots, a_M)$ can then be defined as the geospatial data file access vector. Here, $M$ is the total number of accesses to all small files in $F$, and the natural number $a_i \in [1, N]$ $(i = 1,2,\cdots, M)$ denotes the label of the $i$th requested file from $F$ (i.e., $a_i = k$ $(i = 1,\ldots, M)$, which indicates that the $i$th requested file is $f_k$ $(k \in [1, N])$).

Because the storage length is $n$, we divide $A$ into several $n$-element sub-vectors; then, $A$ can be written as $A = (S_1, S_2, \cdots, S_l)$, where $S_i = (a_{i1}, a_{i2}, \cdots, a_{in})$ $(a_{ij} \in [1, N], 1 \le i \le l, 1 \le j \le n)$ is an $n$-element sub-vector in $A$ and $l$ is the total number of $n$-element sub-vectors. The set of all sub-vectors of vector length $n$ is denoted by $S = \{S_k: k \in [1, l]\}$. For $\forall S_k \in S$, let $\tilde{S}_k = \{a_{k1}, a_{k2}, \cdots, a_{kn}\}$ denote the set of all $n$ elements of sub-vector $S_k$. We can then define the access correlation function for all small files as shown in Eq (2); this function represents the relationship between any pair of small files:

$$R_{S_k}(i,j) = \begin{cases} 1 & i \in \tilde{S}_k,\ j \in \tilde{S}_k,\ i \neq j \\ 0 & Others \end{cases} \quad 1 \le i \le N,\ 1 \le j \le N,\ 1 \le k \le l \tag{2}$$

Here, $R_{S_k}(i,j)$ denotes the access correlation between $f_i$ and $f_j$ during a short period of access time. Therefore, $R_{S_k}(i,j)$ indicates whether the geospatial data files $f_i$ and $f_j$ are both likely to be requested within a short period of time. If $R_{S_k}(i,j) = 1$, then we consider that $f_i$ and $f_j$ have one storage conflict, and we define the following:

$$R_S(i,j) = \sum_{k=1}^{l} R_{S_k}(i,j) \quad 1 \le i \le N,\ 1 \le j \le N \tag{3}$$

$R_S(i, j)$ represents the total number of storage conflicts between $f_i$ and $f_j$. A larger value of $R_S(i, j)$ indicates a higher level of storage conflict or a higher total concurrent access probability **(TCAP)** between the small files $f_i$ and $f_j$, corresponding to a higher probability that these files will be assigned to different storage nodes to achieve a higher total parallel access probability **(TPAP)** in the case that they are requested simultaneously.

For all small files in $F$, we can obtain an $N \times N$ matrix based on $S$ as follows:

$$R_S = (R_S(i,j))_{N\times N} \quad 1 \le i \le N,\ 1 \le j \le N \tag{4}$$

Here, $R_S$ is called the access correlation matrix and represents the concurrent access correlations among the small files. This matrix has the following properties (as proven in the appendices):

1. $R_S$ is a symmetric matrix, i.e., $R_S{}^T = R_S$;
2. $\sum_{i=1}^{N}\sum_{j=1}^{N} R_S(i,j) = (n-1)M \equiv K_S$;
3. $P(f_i) \cong \sum_{j=1}^{N} R_S(i,j) / \sum_{i=1}^{N}\sum_{j=1}^{N} R_S(i,j) = \sum_{j=1}^{N} R_S(i,j)/\mathrm{K_S} \quad i \in [1,N]$; and
4. $P_S(f_j|f_i) \cong R_S(i,j) / \sum_{j=1}^{N} R_S(i,j) = K_S R_S(i,j)/P(f_i) \quad i,j \in [1,N]$.

Here, $P(f_i)$ is the concurrent access probability of $f_i$ in $S$, and $P_S(f_j|f_i)$ is the conditional probability that $f_j$ will belong to the same $S$ to which $f_i$ belongs.

## 3.3 Mathematical model of APSA

Let $T_i = (t_{i1}, t_{i2}, \cdots, t_{in})$ be an arbitrary row vector in $T$. Then, according to $T_i$, we can store $n$ geospatial data files $\tilde{F}_i = f_{T_i} = \{f_{t_{i1}}, f_{t_{i2}}, \ldots, f_{t_{in}}\}$ in the $i$th storage node $c_i$. If any file in $f_{T_i}$ is requested, then $f_{T_i}$ is requested (in other words, the storage node $c_i$ is busy and is unable to serve any other clients), and $P(f_{T_i})$ denotes the concurrent access probability of the set of small files $f_{T_i}$. On the basis of the 3rd property of $R_S$ stated in section 3.2, we can define the following:

$$P(f_{T_i}) = \sum_{j=1}^{n} P(f_{t_{ij}}) = \sum_{j=1}^{n}\Big(\sum_{k=1}^{N} R_S(t_{ij},k)/\mathrm{K_S}\Big) = \frac{1}{K_S}\sum_{j=1}^{n}\sum_{k=1}^{N} R_S(t_{ij},k) \qquad i \in [1,m] \tag{5}$$

Similarly, if $H(f_{T_i})$ denotes the conditional probability of the set of small files $f_{T_i}$ (representing the probability that if any file in $f_{T_i}$ is requested, the other files in $f_{T_i}$ will be requested within a short period of time), then on the basis of the 4th property of $R_S$ stated in section 3.2, we can define the following:

$$\begin{aligned} H(f_{T_i}) &= \sum_{j=1}^{n}\Big[P(f_{t_{ij}}|f_{T_i})\sum_{p=1}^{n} P(f_{t_{ip}}|f_{t_{ij}})\Big] \\ &= \sum_{j=1}^{n}\Big[\frac{\sum_{k=1}^{N} R_S(t_{ij},k)}{\sum_{j=1}^{n}\sum_{k=1}^{N} R_S(t_{ij},k)} \frac{\sum_{p=1}^{n} R_S(t_{ij},t_{ip})}{\sum_{k=1}^{N} R_S(t_{ij},k)}\Big] = \frac{\sum_{j=1}^{n}\sum_{p=1}^{n} R_S(t_{ij},t_{ip})}{\sum_{j=1}^{n}\sum_{k=1}^{N} R_S(t_{ij},k)} \qquad i \in [1,m] \end{aligned} \tag{6}$$

Eqs (5) and (6) define the relationship among $R_S$, the access patterns and the concurrent access probabilities of the small files. For simplicity, let $P(R_S, T_i) = \sum_{j=1}^{n}\sum_{k=1}^{N} R_S(t_{ij},k)$ and

$\mathrm{H}(R_S, T_i) = \sum_{j=1}^{n}\sum_{p=1}^{n} R_S(t_{ij}, t_{ip})$; then, Eqs (5) and (6) can be rewritten as follows:

$$P(f_{T_i}) = P(R_S, T_i)/K_S \tag{7}$$

$$H(f_{T_i}) = H(R_S, T_i)/P(R_S, T_i) \tag{8}$$

Furthermore, according to the definition of the map $T : F \Rightarrow \tilde{F}$, the **TCAP** $H(\tilde{F})$ can be defined as in Eq (9):

$$H(\tilde{F}) = \sum_{i=1}^{m} P(\tilde{F}_i)H(\tilde{F}_i) = \sum_{i=1}^{m} P(f_{T_i})H(f_{T_i}) \tag{9}$$

By combining (7), (8) and (9), we obtain the following:

$$H(\tilde{F}) = \sum_{i=1}^{m} \frac{P(R_S, T_i)}{K_S}\frac{H(R_S, T_i)}{P(R_S, T_i)} = \frac{1}{K_S}\sum_{i=1}^{m} H(R_S, T_i) = \frac{1}{K_S}H(R_S, T\ ) \tag{10}$$

where

$$H(R_S, T) = \sum_{i=1}^{m} H(R_S, T_i) = \sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{p=1}^{n} R_S(t_{ij}, t_{ip}) \tag{11}$$

Eq (10) describes the explicit relationship between the objective function $H(\tilde{F})$ and both $R_S$ and $T$. As mentioned above, the goal of **APSA** is to achieve parallel access to geospatial data files, which requires a **TPAP** that is as high as possible. Therefore, the goal of **APSA** can be restated as the attempt to obtain the lowest possible **TCAP**, and we can conclude that $H(\tilde{F})$ is proportional to $H(R_S, T)$. Therefore, the mathematical model of **APSA** can be defined as follows:

$$T^* = \arg\min_{T}(H(R_S, T)) \tag{12}$$

If we obtain an optimal $T$ according to Eq (12), then we also obtain an optimal $B$. However, this is a typical NP-hard problem, and the traversal search method is impractical because of the extremely large amount of calculation time required. Therefore, the goal of the algorithm must be modified to obtain a reasonable solution. This modification is discussed in the next section.

## Practical Heuristic Algorithm for APSA

According to the APSA description given in section 3, the orders in and among the groups of $\tilde{F}$ are meaningless, and thus, the underlying problem is typical of unordered average combinations. Therefore, we can develop a heuristic algorithm to obtain an optimal $T$ using Eq (12). This process includes 3 main steps: 1) obtain $F$ from the historical access log information, 2) generate $R_S$ using the algorithm proposed in section 3.2 and reorder $R_S$ to reduce the scale of the search; and 3) employ a locally approaching search method to find the optimal $T$.

### 4.1 Preprocessing of *F*

Large numbers of small files are stored in the Digital Earth system, and therefore, we must process a large collection of natural files. For example, for the 90-meter-resolution global terrain data files from the Shuttle Radar Topography Mission (SRTM90), the length of $I_o$ will be 3,538,890 and the size of $R_S$ will be 3538890×3538890. However, as indicated by the geospatial

DAP, only approximately 20% of these files will be requested [21–23]. Therefore, we only need to process a subset of the data, which allows us to reduce the size of $R_S$.

To satisfy the requirements for storing all $N$ small files in $m$ storage nodes on average, we can copy certain geospatial data files that have higher request rates and assign new labels to the copies (to expand the scale of $F$ and $I_o$).

## 4.2 Preprocessing and reordering of $R_S$

The historical access log information is produced by the Digital Earth system after a long period of operation, and from this information, we can obtain $A$. Then, we can generate $R_S$ using the algorithm described in section 3.2.

To reduce the search scale, we must concentrate non-related elements together to allow the storage distribution rule $T$ to be rapidly sought and obtained. Thus, we can reorder $R_S$ using the RCM ordering algorithm, which was developed based on the CM ordering algorithm [25–26]. We can then obtain a new $P$, in which most of the nonzero elements are concentrated along the diagonal.

The objective of **APSA** is to generate grouped geospatial data files and to ensure that these groups have the smallest possible storage conflict, meaning that the value of $R_S$ is zero or near zero. To employ the RCM ordering algorithm to concentrate the non-related elements along the diagonal, we must first preprocess $R_S$ in two steps: 1) denote the largest value of $R_S$ by $R_{\max}$ and search for and obtain $R_{\max}$ from $R_S$, and 2) $\forall R_S(i, j)$, set $R_S(i, j) = R_{\max} - R_S(i, j)$ $(i < j \leq N)$.

Afterward, we can use the RCM algorithm to reorder $R_S$, export the resulting permutation $P$ in reverse order and then export the corresponding matrix $P_S$ (the standard RCM algorithm is used in this paper; therefore, a description of this process is omitted).

## 4.3 Determination of the optimal $T$ using a locally approaching search algorithm

Several steps are required to obtain the optimal $T$:

1. Initialize $T = (0)_{m\times n}$ and set $k = 1$.
2. Let $i$ be the label of the first row in $P_S$; then, a non-zero length can be obtained: *non_zero_len* $= \max\{|i-j|, A_{ij} \neq 0\}$.
3. If *non_zero_len* $\leq n$, then for every $j \in [1, n]$, let $T[k][j] = P(j)$. Then, delete the $n$ top rows of $P_S$ and delete the first $n$ elements of $P$. Go to 7).
4. If *non_zero_len* $> n$, then take the upper triangular matrix $UTM_m = \{A_{ij}, 1 \leq i \leq j \leq$ *non_zero_len*$\}$. Let $X = (x_1, x_2, \cdots\cdots, x_n)$ denote an $n$-dimensional temporary vector and initialize $x_1 = 1$, $i_1 = 1$, and $j = 2$. Set the basis vector of the local search to $B = UTM_m(i_1)$.
5. While $j \leq n$, search for the largest element $B(i_j)$ in $B$. Then, the label of the $j$th file is $i_j$; set $x_j = i_j$. Update the basis vector of the local search as follows: $B = B + UTM_m(i_j)$. Set $B(i_j) = -K_S$ and $j = j + 1$.
6. For $1 \leq j \leq n$, set $T[k][j] = P(x_j)$. Then, delete the $n$ rows of $P_S$ that are defined in the temporary vector $X$ and delete the corresponding $n$ elements of $P$.
7. Set $k = k + 1$. If $k \leq m$, then return to 2); otherwise, stop.

For the optimal $T$, $n$ small files are included in each group, i.e., the $i$th group includes small files labeled as $T[i][1]$, $T[i][2]$, $#x2026;$#x2026;, $T[i][n]$, and the relationship among the files in a given group is as weak as possible. Therefore, the *i*th group of files can be stored in the *i*th storage node $c_i$.

## Experimental Results and Analysis

To evaluate the performance of the algorithm, several tasks were experimentally investigated: 1) selecting the geospatial image dataset to be stored in distributed storage nodes; 2) finding the optimal $T$ based on the historical access log information recorded by the Digital Earth server [20] using the heuristic algorithm proposed in section 4; 3) requesting the same dataset simultaneously based on other historical access log information; and 4) computing the **TPAP** performance and comparing it with those of LSA and RSA.

We define the **TPAP** performance as follows:

$$TPAP = \frac{\sum_{i=1}^{L}\sum_{j=1}^{m} x_{ij}}{L \times m} \tag{13}$$

where $L{\times}m$ denotes the total number of requests for small files over a long period and $x_{ij}$ denotes whether the *j*th storage node is accessed during a short period. Specifically, $\forall i \in [1, L]$, if $c_j$ is accessed during this short period, then $x_{ij} = 1$; otherwise, $x_{ij} = 0$. Therefore, the value of *TPAP* cannot exceed 1.

The simulation algorithm was implemented using Microsoft Visual C++6.0, and the sequences were accessed and processed using MATLAB R2009a (Version 7.8.0.347) in accordance with the rules specified in section 4. All datasets are summarized in Table 2. Two types of datasets were included: geospatial image datasets produced by our own simulation system [27] and an INS (Instructional Workload) dataset obtained from the University of California at Berkeley [28].

### 5.1 Contrasting experiments on different algorithms

From the 30-m-resolution global terrain data files from the Shuttle Radar Topography Mission (SRTM30), we selected **10,000** geospatial image data files from a given spatial region for use as the experimental dataset. All available sequences of access log information for the considered SRTM30 datasets are summarized in Table 2 (category 1 and category 2). We determined the optimal $T$ using the first access log information file in category 1, and we then assessed the performance of the various algorithms using the second log file in category 1. Fig 1 presents the results of the comparison at different scales of $m$.

**Table 2. The datasets used in this analysis.**

| Category | Dataset | Number of access sequences | Dataset size[a] | Access sequence length[b] |
|---|---|---|---|---|
| 1 | SRTM30 | 5 | 10,000 | 180,000~204,000* |
| 2 | SRTM30 | 10 | 2,000~10,000 | 180,000~204,000* |
| 3 | SRTM90 | 2 | 10,000 | 200,000* |
| 4 | Landsat7 | 2 | 10,000 | 200,000* |
| 5 | INS | 3 | 20,000 | 244,339~712,605** |

[a]All data were relabeled with natural numbers ranging from zero to the length of the dataset.

[b]Each access sequence recorded only the labels of the data in chronological order.

* As stated in section 4.1, only of the 20% files will be requested.

** All files will be requested.

doi:10.1371/journal.pone.0133029.t002

As shown in Fig 1, APSA and LSA exhibit comparable performance in a small-scale environment, especially when *m* is less than 8. In this case, only a small number of storage nodes are available, and most of the small files must be stored in the same storage node. Moreover, the majority of clients will request small files according to their navigation paths [22–24], and therefore, LSA, which stores small files in different storage nodes depending on their spatial locations, can satisfy the requirements of parallel I/O.

For a larger number of storage nodes, however, **APSA** performs better than **RSA** or **LSA**, especially when *m* is larger than 22. The performance of **APSA** is higher than that of **LSA** by approximately 10% and higher than that of **RSA** by approximately 15%, especially for a large number of servers.

Furthermore, we tested the performance of APSA using all log files in category 2, which represented various scales of small files. Fig 2 displays the results of the comparison for various values of *N*.

As shown in Fig 2, as the scale of the geospatial data increases, the performance of LSA exhibits almost no change, and the performance of RSA becomes considerably more unstable and fragmented and even decreases to some extent. By contrast, the performance of **APSA** improves, exhibiting a sustained increase. The experimental results show that the proposed

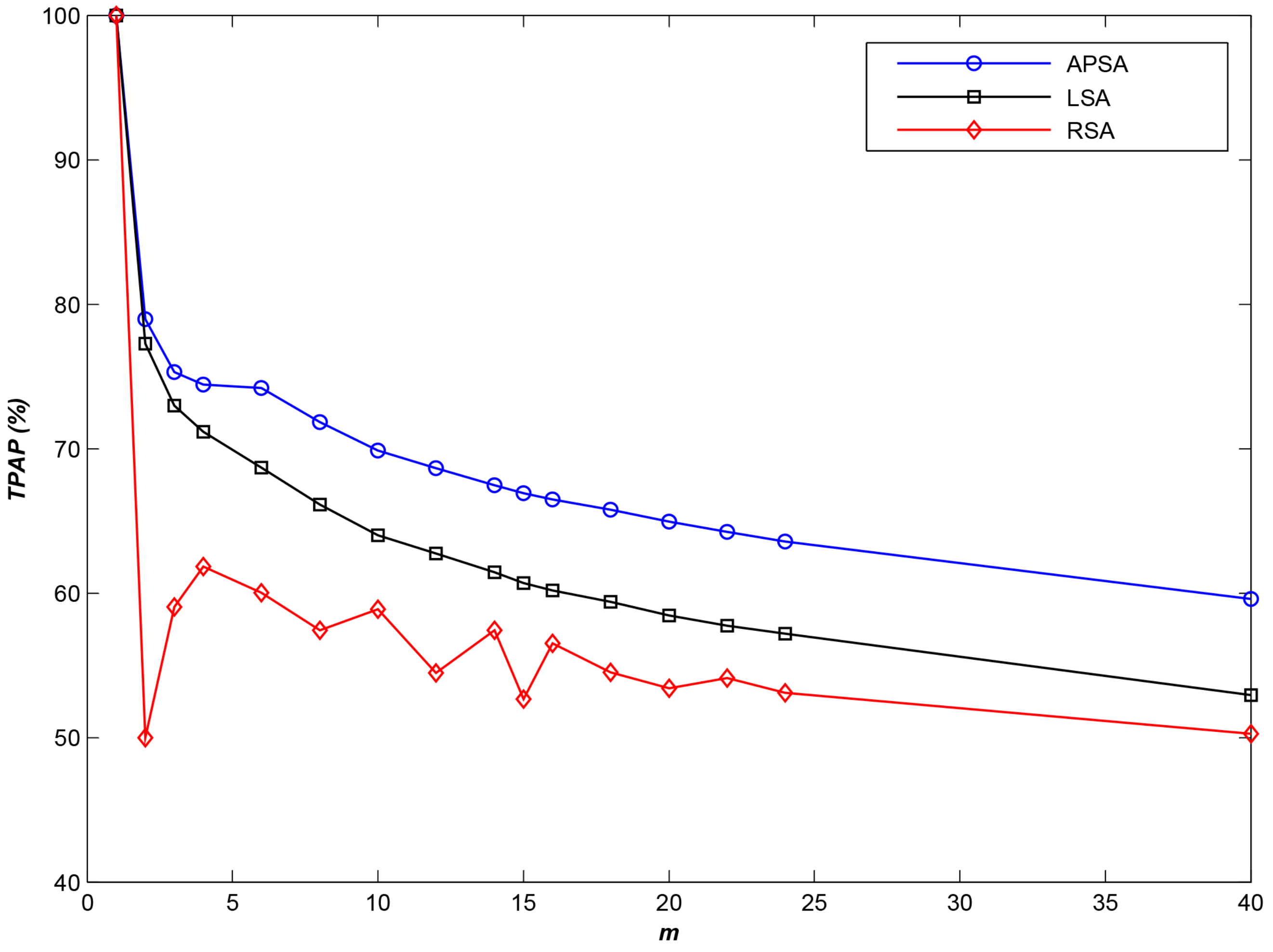

**Fig 1. Comparison of *TPAP* results at various scales of *m* (*N* = 10,000).**

doi:10.1371/journal.pone.0133029.g001

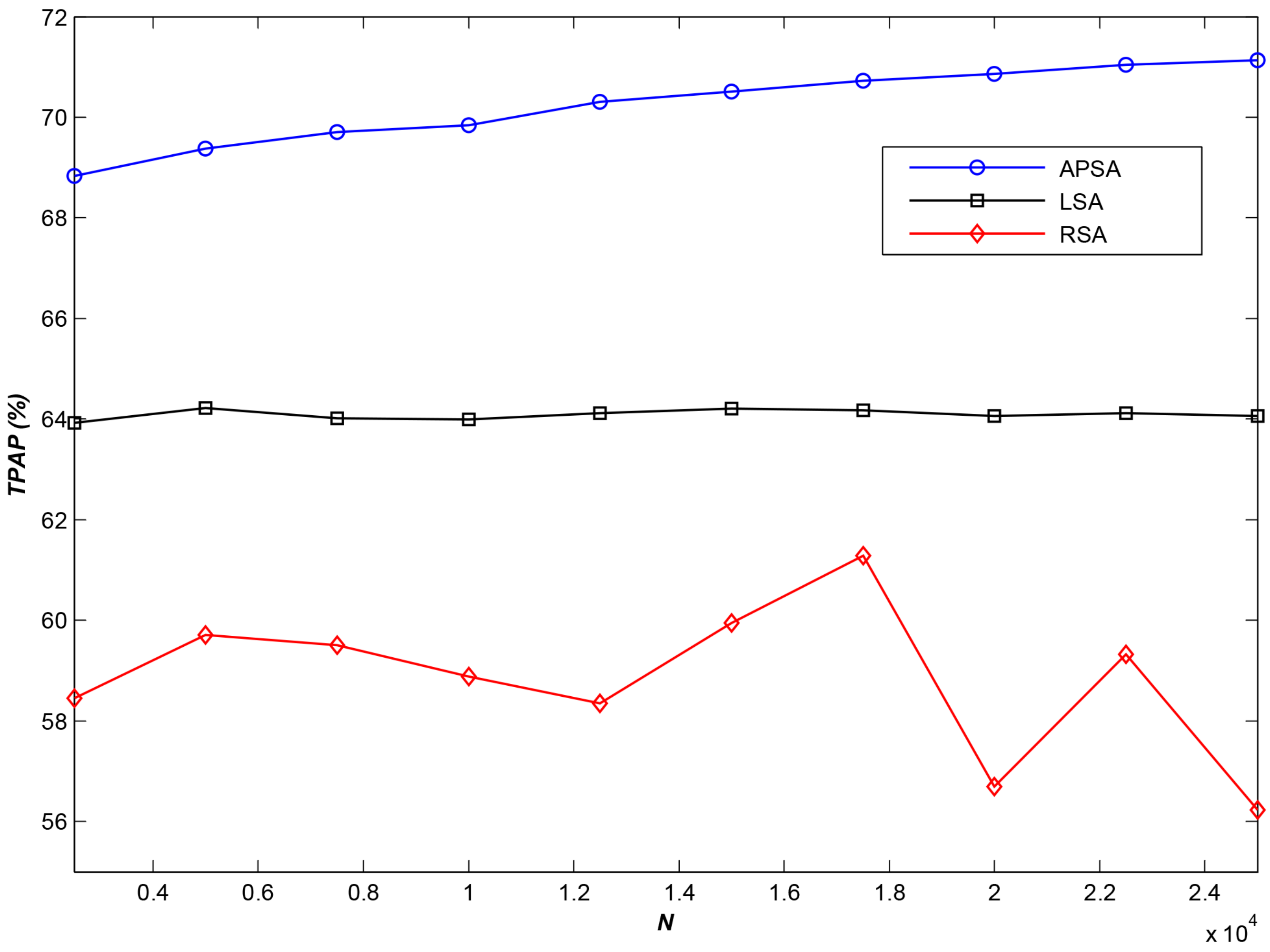

**Fig 2. Comparison of *TPAP* results for various values of *N* (*m* = 10).**

doi:10.1371/journal.pone.0133029.g002

algorithm offers greater advantages in a large-scale environment (i.e., for a large number of storage nodes).

## 5.2 Experiments using different types of access log information

It is important to determine whether an algorithm can always provide high performance. To assess the adaptability of the algorithm, we selected four typical access log information files representing the access behavior of end users (clients) at different times. As in the first experiment, we used the first log file from category 1 to determine the optimal *T* and then used the 2nd through 5th log files of category 1 to test the performance of APSA. The experimental results are shown in Fig 3A–3D).

As shown in Fig 3, comparable performances were obtained when different access log files were used to obtain a feasible solution. In addition, the rate of change in ***TPAP*** did not exceed **6%**. Thus, this experiment demonstrated that **APSA** exhibits stable parallel access performance under different conditions.

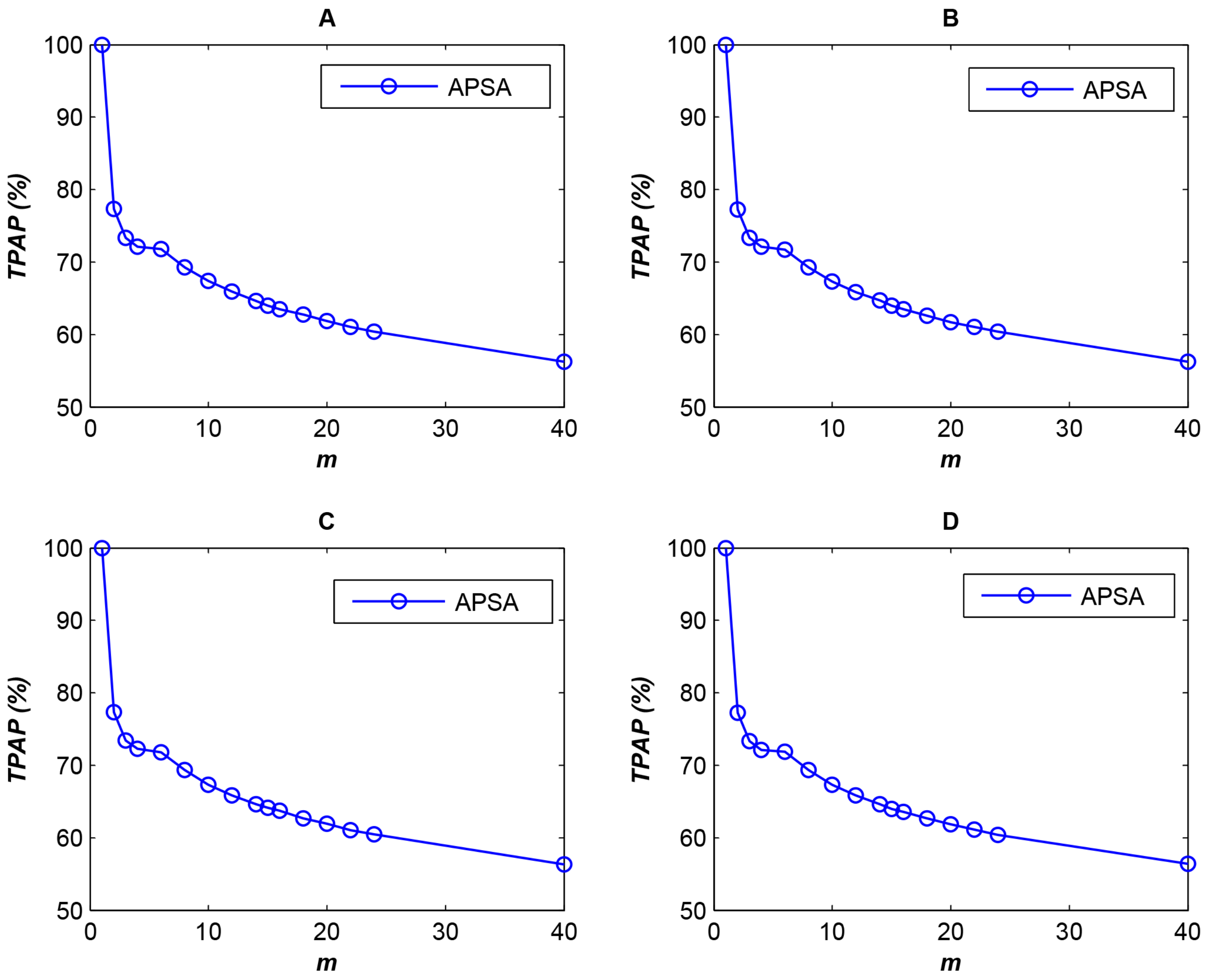

**Fig 3. Comparison of *TPAP* results for different access log files (*N* = 10,000) (each plot, A-D, represents a different access log file).**

doi:10.1371/journal.pone.0133029.g003

## 5.3 Experiments using different geospatial image datasets

To assess the adaptability of the algorithm to different geospatial image datasets, we selected three typical geospatial datasets: SRTM30, SRTM90 and Landsat7 ETM+[29]. In this experiment, the first two log files from categories 1, 3 and 4 were used.

Let $TPAP_{SRTM\ 30}$ be the performance indicator for SRTM30, let $TPAP_{SRTM\ 90}$ be the performance indicator for SRTM90, and let $TPAP_{\mathrm{Landsat7}}$ be the performance indicator for Landsat7. Then, the ***TPAP*** change rate (*CCAR*) can be calculated using Eq (14). The experimental results are shown in Fig 4.

$$\begin{cases} CCAR_{SRTM\ 90} = (TPAP_{SRTM\ 90} - TPAP_{SRTM\ 30})/TPAP_{SRTM\ 30} \\ CCAR_{\mathrm{Landsat7}} = (TPAP_{\mathrm{Landsat7}} - TPAP_{SRTM\ 30})/TPAP_{SRTM\ 30} \end{cases} \tag{14}$$

As shown in Fig 4, ***CCAR*** exhibits essentially no change, and the highest rate of change does not exceed **0.4%** and thus is generally negligible. Therefore, the experiment shows that **APSA** demonstrates broad adaptability to different geospatial datasets.

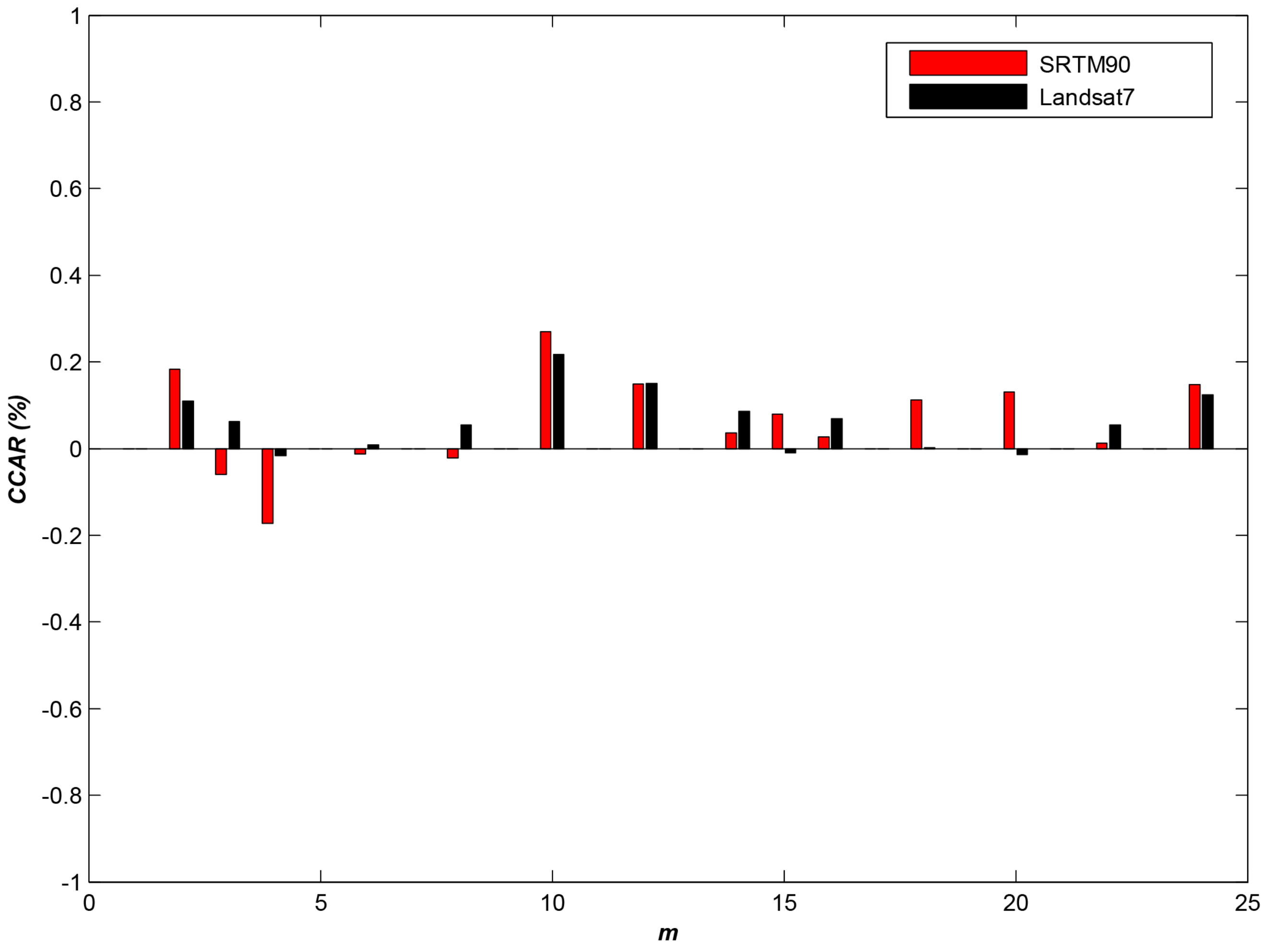

**Fig 4. The adaptability of APSA to different datasets ($N$ = 10,000).**

doi:10.1371/journal.pone.0133029.g004

## 5.4 Experiments based on redundant data storage

To simplify data processing and enhance the efficiency of the algorithm, we assume (as noted in the **Appendix** (**Property 2**)) that any given small file will not be repeatedly requested within a short period of time. Nevertheless, in actuality, certain small files do exist that are repeatedly requested within short periods of time.

However, most storage solutions adopt a copy storage strategy for data security; examples of such systems include RAID, which copies and stores each datum to backup disks, and RADOS (Reliable, Autonomic Distributed Object Store), which manages a number of copies on demand [30].

Inspired by this approach, we can copy select geospatial data files that have higher request rates, and then we can store them in different storage nodes with new labels. Let *CR* be the ratio of the number of copies to the total number of geospatial data files; next, a larger *CR* obviously implies the generation of more copies. Based on the results of the first experiment, Fig 5 displays a comparison between the original APSA and $\mathbf{APSA_b}$, for which *CR* is 6.3% (the geospatial data files with the highest access rates are each copied only once). Furthermore, comparisons between the original APSA and variants of $\mathbf{APSA_b}$ with different *CR*s are shown in Fig 6.

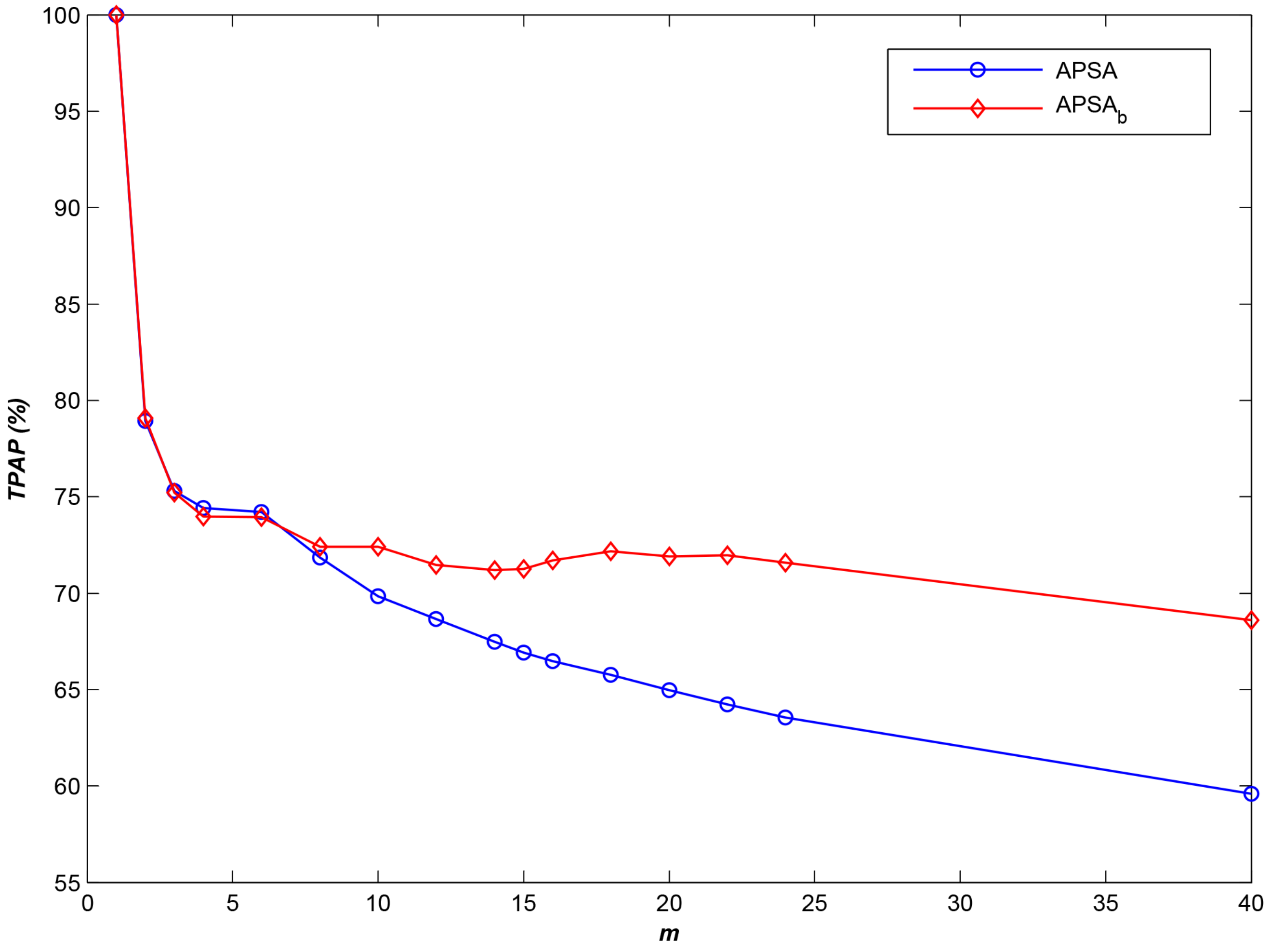

**Fig 5. The performance improvement achieved using the copy storage strategy (*N* = 10,000, *CR* = 6.3%).**

doi:10.1371/journal.pone.0133029.g005

As shown in Fig 5, the performance of $\mathbf{APSA_b}$ is essentially identical to that of APSA for small numbers of storage nodes, especially when *m* is less than 8. In this case, only a small number of clients can access the system simultaneously (for simplicity, we assume that one storage node can provide service to only one client at a time; in fact, all clients share the resources of the storage node, including CPUs and bandwidth, when they simultaneously request service from the same storage node, but the server may require the same amount of time to serve all end users simultaneously as it does to serve each client sequentially). Therefore, there is only a very small probability that more than one client will attempt to access the same geospatial data file simultaneously, and therefore, the copy storage strategy will not function effectively. However, as the scale of the system increases, the performance improves considerably, especially for an *m* greater than 22; indeed, the performance can be improved by more than **20%**.

Obviously, larger *CR*s mean more copies, and more copies result in a higher probability that different clients can simultaneously access the same geospatial data files in parallel. Fig 6 shows the corresponding improvement in performance as *CR* increases. However, achieving continuous improvement in the system performance is difficult once *CR* reaches a certain size, because some of the small files will be simultaneously accessed at certain times and accessed in a

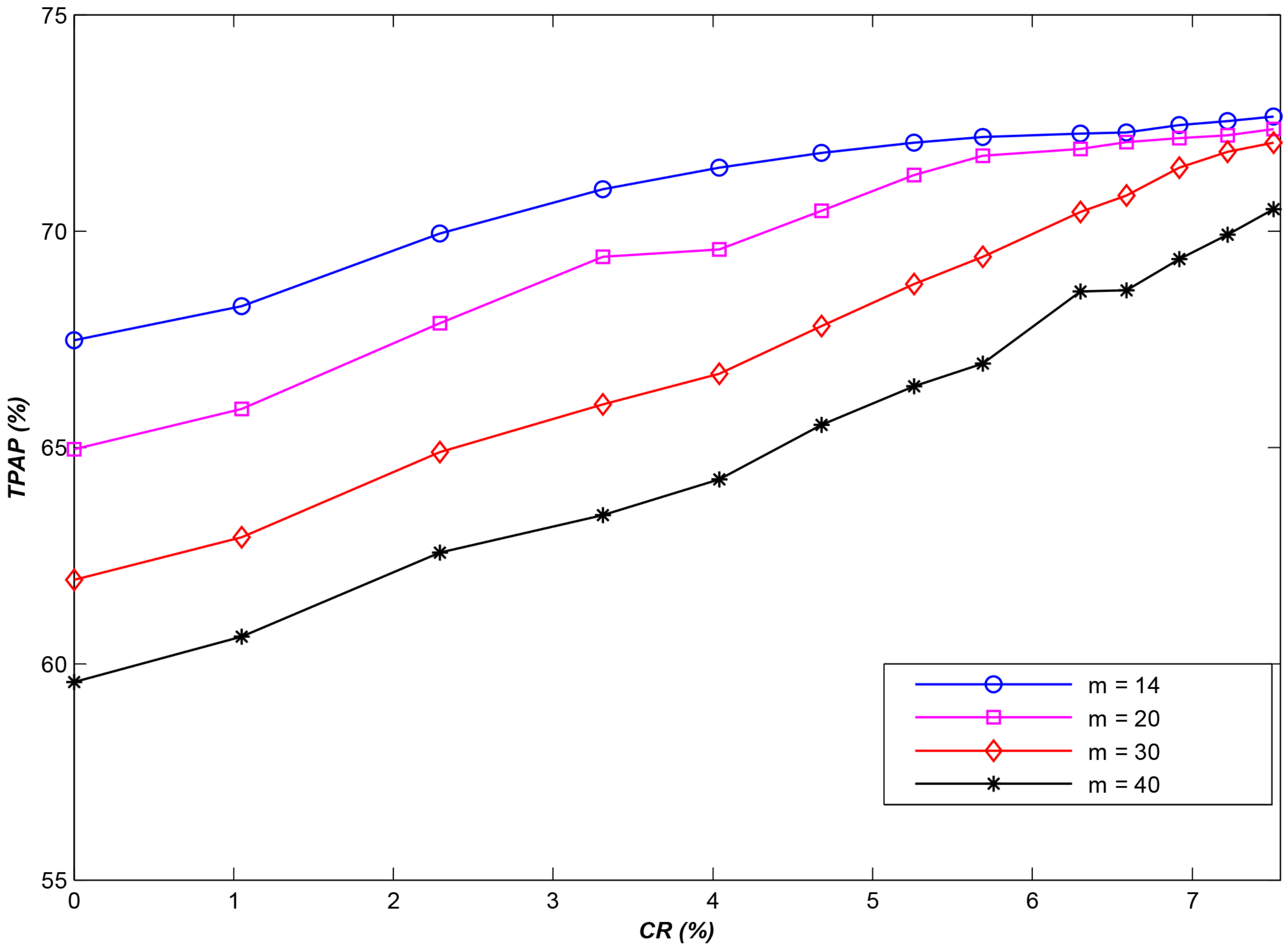

**Fig 6. The performance improvement achieved using the copy storage strategy with various *CR*s (*N* = 10,000).**

doi:10.1371/journal.pone.0133029.g006

staggered manner at others. This situation cannot be fully resolved using a copy storage strategy alone, and it can only be partially resolved through algorithm optimization to obtain a better solution than that identified by the method introduced in section 4. Such algorithm optimization will be a focus of our future studies.

## 5.5 Adaptability experiments based on another type of dataset

In this section, we consider another typical type of log file, namely, INS (Instructional Workload) files, to test the adaptability and flexibility of our solution. The INS files used in this experiment were obtained from separate distributed file system application environments at the University of California at Berkeley. Roselli [28] traced two groups of Hewlett-Packard series 700 workstations running HP-UX 9.05. INS data were collected from twenty machines in these groups, which were located in laboratories for undergraduate classes.

As in the experiments presented in section 5.1, we selected **20,000** files for use as the experimental datasets and employed different access log information files to obtain the optimal *T* and to simulate file access. Thus, we were able to compute *TPAP* at various storage node scales.

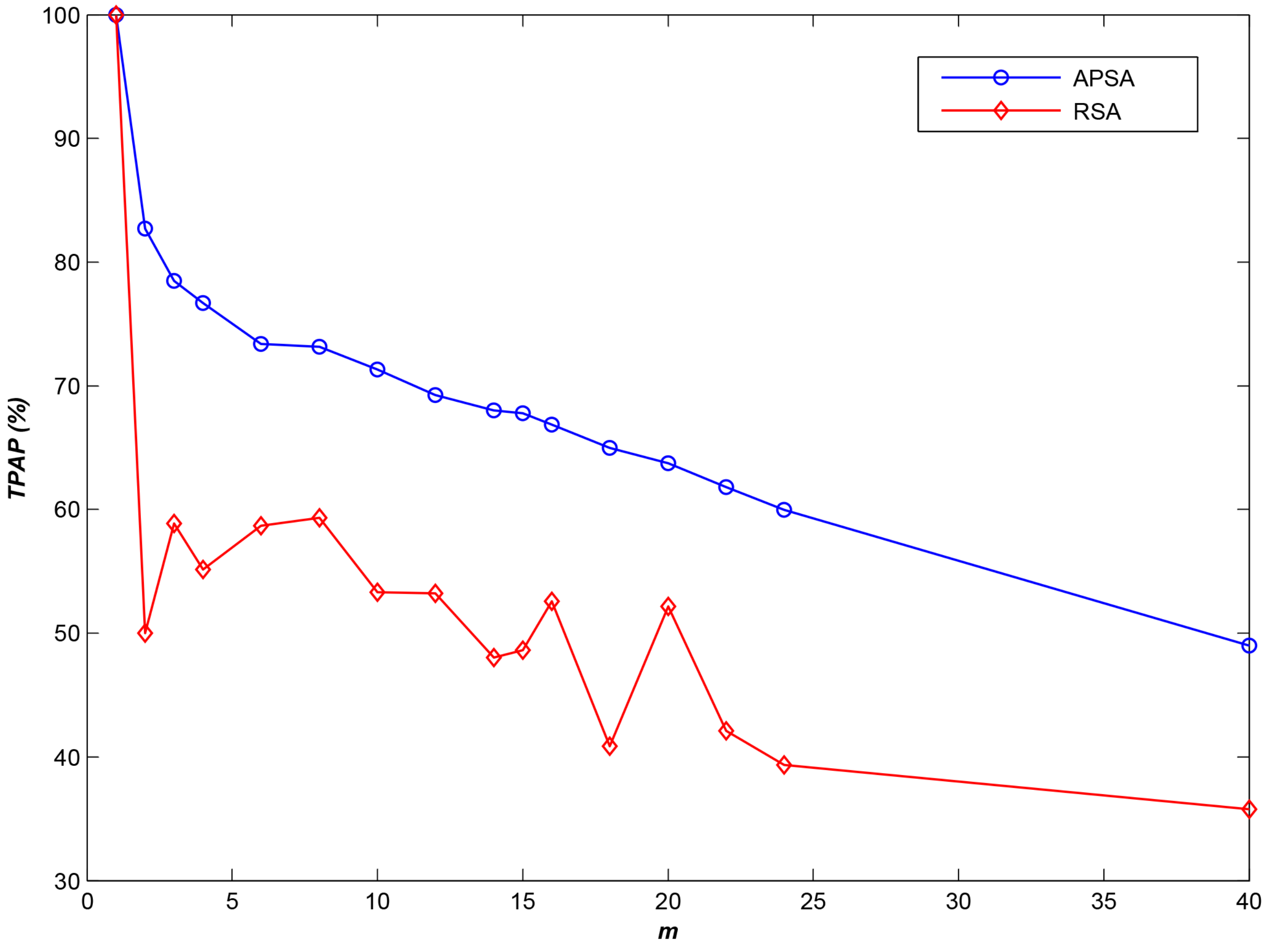

**Fig 7. Adaptability experiment based on the INS dataset (*N* = 20,000).**

doi:10.1371/journal.pone.0133029.g007

All log files for the INS dataset are summarized in Table 2 (category 5). Fig 7 presents the results of a comparison between APSA and RSA on these files (LSA was not used because there was no location relationship among the INS files).

As shown in Fig 7, APSA demonstrates higher performance than does RSA (by approximately 18%) when applied to this type of general dataset. However, INS datasets include not only small files but also several large files. Therefore, when end users (clients) access these large files, they will occupy storage node resources for a longer time, and therefore, the performance of **APSA** will decline sharply (as shown in Fig 8). Nevertheless, the performance of **APSA** is still higher than that of **RSA**, by approximately 10%. Furthermore, if we ignore accesses to large files, where *LR* is the ratio between the minimum length of the ignored sequences and the total sequence length, then the results are as shown in Fig 9 for various *LR*s. From this figure, it is evident that the performance of **APSA** can be effectively improved by increasing *LR*. Recall that, as mentioned in section 1, declustering technologies can be used to satisfy the requirements of parallel I/O in distributed environments; therefore, hybrid storage strategies are required to satisfy the requirements for both small and large files. The development of such a hybrid strategy will be a focus of our future investigations.

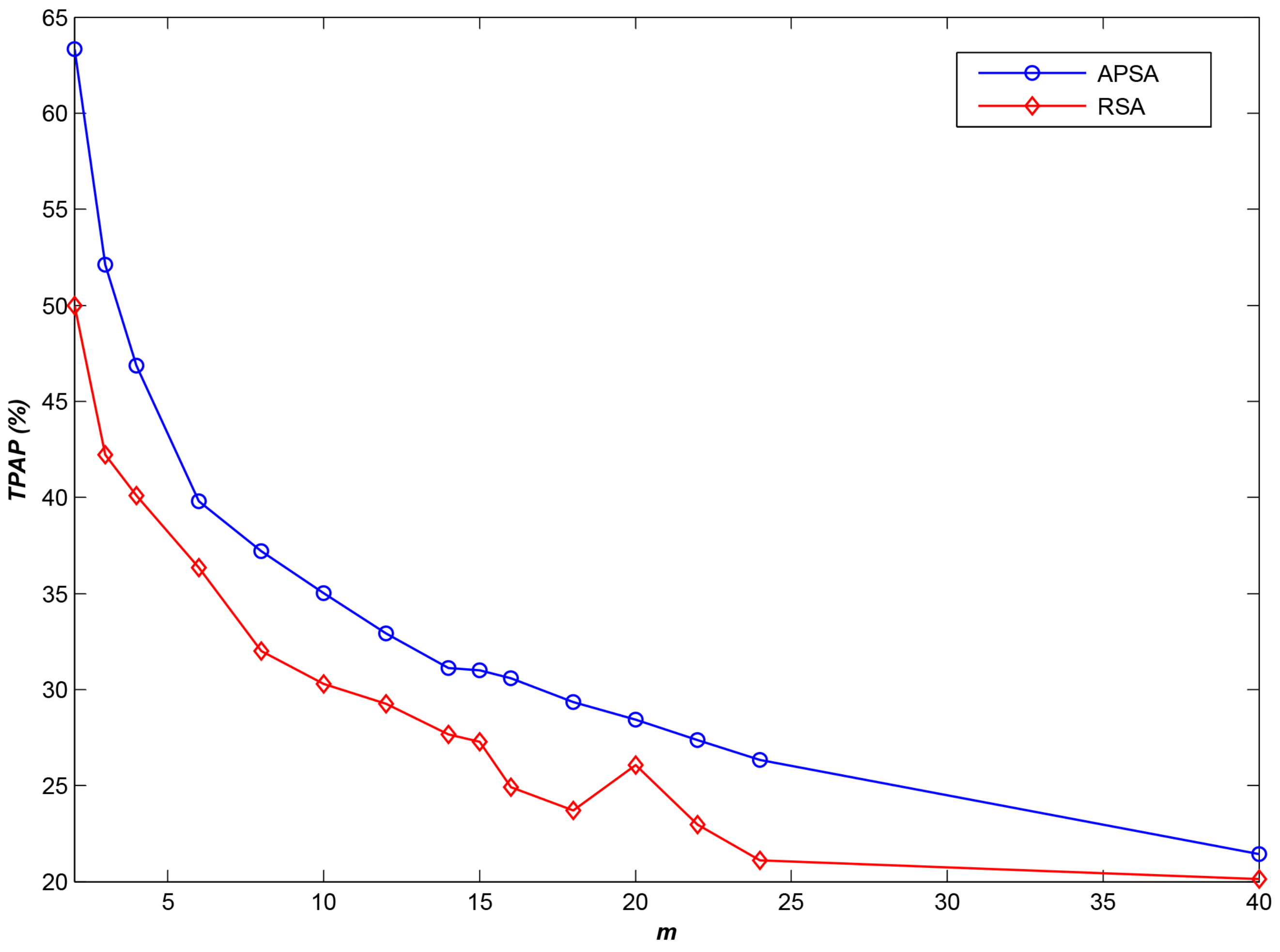

**Fig 8. Experimental results for another access log file (*N* = 20,000).**

doi:10.1371/journal.pone.0133029.g008

## Conclusions

In this paper, we investigated the challenges associated with the distributed storage of small files and proposed a complete theoretical system for distributing small geospatial data files for storage. First, we discussed the patterns of access to geospatial data files and created a theoretical mathematical model to express the relationship among geospatial data files. Then, we developed a practical heuristic algorithm to find an acceptable optimization solution.

To verify the developed mathematical model, a series of comparative experiments was performed, as described in section 5. All of these comparative experiments demonstrate that our method can achieve a higher ***TPAP*** than other algorithms can (by approximately **10–15%**) and that its performance can be further improved by more than **20%** using a copy storage strategy. Most importantly, all experiments show that APSA can satisfy the requirements for the storage of a large amount of small files in a distributed environment.

Although our algorithm exhibits higher performance than any of the other tested strategies, it is based solely on historical access log information; there is no provision for dynamically updating the strategies, even when the hotspots change (i.e., when the relationship among the small files changes) [31]. Thus, the establishment of a rapid and efficient updating mechanism

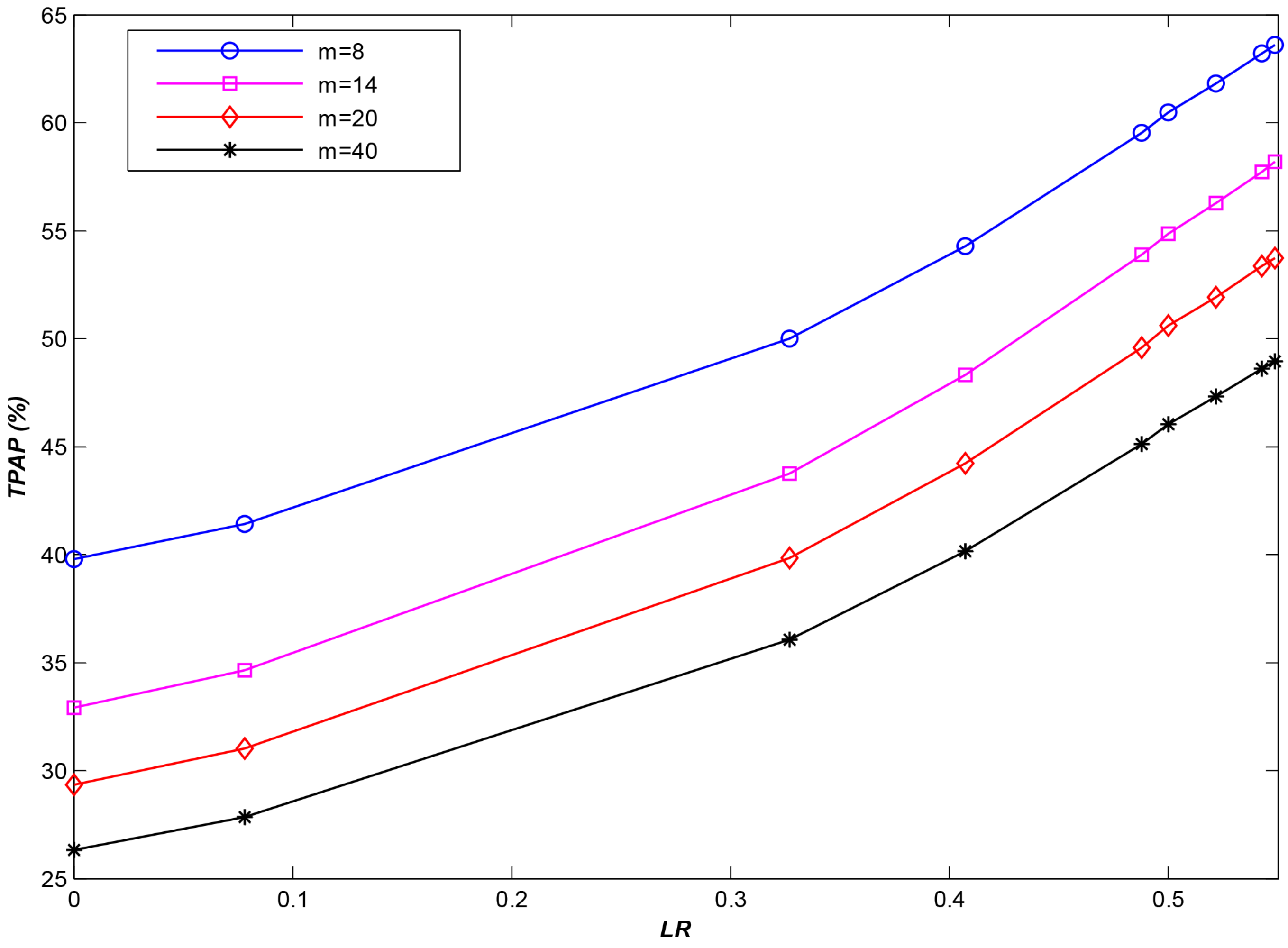

**Fig 9. The performance improvement at various *LR*s (*N* = 20,000).**

doi:10.1371/journal.pone.0133029.g009

that considers and tracks such changes in hotspots is worth further investigation. Moreover, hybrid storage strategies that satisfy the requirements for the storage of both large and small files should also be further investigated.

However, our paper only considers how small geospatial image data files can be separately stored into different storage nodes to improve system I/O performance. The nodes also include many very tiny files, some of which must be merged together to avoid storage fragments. One such example is sport activity files, which include a significant amount of information such as GPS locations, distance, speed, calories, and heart-rate; these files are widely used to optimize athletes' training [32]. Due to differences in duration and GPS distance, the sizes of sport activity files will vary substantially, as some may only require a few kilobytes (KB), while others may reach a size of 10 MB [33]. Thus, a type of integrated storage strategy that can satisfy the requirements of very tiny files, small files and large files should also be considered.

## Appendix: Proof of the Properties of $R_S$

### Property 1

From Eq (3), it is clear that $R_{S_k}(i,j) = R_{S_k}(j,i)$. Then, we can obtain the following:

$$R_S(i,j) = \sum_{k=1}^{l} R_{S_k}(i,j) = \sum_{k=1}^{l} R_{S_k}(j,i) = R_S(j,i) \qquad (A-1)$$

Therefore, $R_S(i, j) = R_S(j, i)$; that is, $R_S{}^T = R_S$, indicating that $R_S$ is a symmetric matrix.

### Property 2

Several studies have shown that a chronological access sequence of small files, $R$, follows a Markov process; such processes are widely used in predictive models [21, 34–35]. In accordance with this model, $A = (a_1, a_2,\cdots, a_M)$ can be treated as a Markov chain with a state space $I_0 = [1, N]$, where the number of states is $N$. Moreover, based on the G-DAPs, we can assume that the service system has been running for a sufficiently long time before we obtain its access log information $R$ that the distribution of the small files in $R$ is stationary. Therefore, $A$ can be regarded as a stationary Markov chain.

Let $\Pi = \{\pi_i: i \in I_0\}$ denote the stationary distribution of the pattern of access to $F = \{f_1, f_2,\ldots, f_N\}$, let $P = (p_{ij})_{N\times N}$ denote the transition matrix, and let $p_{ij} = P(a_{k+1} = j|a_k = i)$ $(i, j \in I_0, k \in [1, M])$ be the $(i,j)$th element of $P$. We then have $\Pi = \Pi \mathrm{P}$. Thus, $\forall j \in I_0$, we have $\pi_j = \sum_{i=1}^{N} \pi_i p_{ij}$, and $\pi_i$ exhibits no relationship with its location. Therefore, for any set $\tilde{S}$ of all sub-vectors $S_k$, $\forall i \in I_0$, the probability that the $i$th state appears in $\tilde{S}$ is as follows:

$$P(i \in \tilde{S}) = 1 - (1 - \pi_i)^n \qquad (A-2)$$

Because of the large number of small files stored in the system, $N >> 1$, and thus, $\pi_i << 1$. Therefore, we find that $(1 - \pi_i)^n \approx 1 - n\pi_i$, and thus, Eq (A-2) can be rewritten as follows:

$$P(i \in \tilde{S}) \cong n\pi_i \qquad (A-3)$$

Let $P^{(1)}(i \in \tilde{S})$ denote the probability that the $i$th state appears in $\tilde{S}$ only once; then, $P^{(1)}(i \in \tilde{S})$ can be defined as in Eq (A–4):

$$P^{(1)}(i \in \tilde{S}) = C_n^1 \pi_i (1 - \pi_i)^{n-1} \approx n\pi_i - n(n-1)\pi_i^2 \qquad (A-4)$$

If $P^{(>1)}(i \in \tilde{S})$ denotes the probability that the $i$th state occurs in $\tilde{S}$ more than once, then we obtain the following:

$$P^{(>1)}(i \in \tilde{S}) = P(i \in \tilde{S}) - P^{(1)}(i \in \tilde{S}) \qquad (A-5)$$

By combining (A-2), (A-3) and (A-4), we find that

$$P^{(>1)}(i \in \tilde{S}) = n(n-1)\pi_i^2 \qquad (A-6)$$

Based on the above analysis, $\pi_i << 1$, and thus, $P^{(>1)}(i \in \tilde{S})$ is negligible. Therefore, the one broad conclusion that can be drawn from this result is that all elements of $\tilde{S}$ are different; this conclusion is consistent with actual observations of G-DAPs, which indicate that a given small file will not be requested repeatedly within a short period of time. We can then write the following:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} R_{\tilde{S}}(i,j) = \sum_{i,j\in S_k} R_{\tilde{S}}(i,j) = 2C_n^2 = n(n-1) \qquad (A-7)$$

Therefore,

$$\begin{aligned}\sum_{i=1}^{N}\sum_{j=1}^{N} R_S(i,j) &= \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{l} R_{S_k}(i,j) = \sum_{k=1}^{l}\sum_{i=1}^{N}\sum_{j=1}^{N} R_{S_k}(i,j) \\ &= \sum_{k=1}^{l} n(n-1) = n(n-1)l = (n-1)M\end{aligned} \qquad (A-8)$$

Eq (A–8) shows that the sum of all elements in $R_S$ is a constant that is equal to $(n\text{-}1)M$. Therefore, if we let $K_S = (n-1)M$, then we can write the following:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} R_S(i,j) = (n-1)M \equiv K_S \qquad (A-9)$$

## Property 3

According to the analysis presented for **Property 2** and Eq (A–3), $\forall i \in [1, N]$ and $k \in [1, l]$, we can write the following:

$$P(f_i) = \pi_i = \frac{1}{n} \underset{S_k\in S}{P}(i \in S_k) \qquad (A-10)$$

Let $\tilde{S}_i = \{S_k : k \in [1, l],\ i \in S_k\}$ and let $C_i = card(\tilde{S}_i)$; then, the frequency of $S_k \in \tilde{S}_i$ in $S$ is as follows:

$$\underset{S_k\in S}{\bar{P}}(S_k \in \tilde{S}_i) = C_i/l \qquad (A-11)$$

Note that $\sum_{i\in S_k, j=1}^{N} R_{S_k}(i,j) = n-1$ and $\sum_{i\notin S_k,\, j=1}^{N} R_{S_k}(i,j) = 0$. Then, we can write the following:

$$\begin{aligned}\sum_{j=1}^{N} R_S(i,j) &= \sum_{j=1}^{N}\sum_{k=1}^{l} R_{S_k}(i,j) = \sum_{k=1}^{l}\sum_{j=1}^{N} R_{S_k}(i,j) \\ &= \sum_{i\in\tilde{S}_k, k=1}^{l} (n-1) = C_i(n-1)\end{aligned} \qquad (A-12)$$

Therefore,

$$C_i = \frac{1}{n-1}\sum_{j=1}^{N} R_S(i,j) \qquad (A-13)$$

According to the assumption described with regard to **Property 2**, namely, that the geospatial information service system has been running for a sufficiently long time, we know that $l$ is also sufficiently large and that $\underset{S_k\in S}{P}(i\in S_k)\cong\underset{S_k\in S}{\bar{P}}(S_k\in\tilde{S}_i))$. Therefore, we find that

$$\underset{S_k\in S}{P}(i\in S_k)\cong\underset{S_k\in S}{\bar{P}}(S_k\in\tilde{S}_i))=C_i/l=\frac{1}{(n-1)l}\sum_{j=1}^{N}R_S(i,j) \qquad (A-14)$$

Thus,

$$\begin{aligned}P(f_i)&=\frac{1}{n}\underset{S_k\in S}{P}(i\in S_k)\cong\frac{1}{n(n-1)l}\sum_{j=1}^{N}R_S(i,j)\\&=\frac{1}{(n-1)M}\sum_{j=1}^{N}R_S(i,j)=\frac{1}{K_S}\sum_{j=1}^{N}R_S(i,j)\end{aligned} \qquad (A-15)$$

Eq (A–15) shows that the access probability of one small file is proportional to the sum of the elements in the corresponding row (column) of $R_S$.

## Property 4

According to Eq (A–10), we can also write the following:

$$P_S(f_j|f_i)=\frac{\underset{S_k\in S}{P}(j\in S_k|i\in S_k)}{n-1}=\frac{\underset{S_k\in S}{P}(j\in S_k,i\in S_k)}{(n-1)\underset{S_k\in S}{P}(i\in S_k)} \qquad (A-16)$$

As in the proof of **Property 3**, because of the large value of $l$, we have $\underset{S_k\in S}{P}(i\in S_k,j\in S_k)\cong\underset{S_k\in S}{\bar{P}}(i\in S_k,j\in S_k)$. Using the same analysis presented for Eq (A–11), we can write the following:

$$\underset{S_k\in S}{\bar{P}}(i\in S_k,j\in S_k)=\frac{R_S(i,j)}{l} \qquad (A-17)$$

In combination with Eqs (A–14) and (A–17), Eq (A–16) can be rewritten as follows:

$$\begin{aligned}P_S(f_j|f_i)&=\frac{\underset{S_k\in S}{P}(j\in S_k,i\in S_k)}{(n-1)\underset{S_k\in S}{P}(i\in S_k)}=\frac{R_S(i,j)}{l}\frac{1}{(n-1)\underset{S_k\in S}{P}(i\in S_k)}\\&=\frac{R_S(i,j)}{(n-1)l}\frac{(n-1)l}{\sum_{j=1}^{N}R_S(i,j)}=\frac{R_S(i,j)}{\sum_{j=1}^{N}R_S(i,j)}\\&=K_SR_S(i,j)/P(f_i)\end{aligned} \qquad (A-18)$$

## Acknowledgments

We would like to thank the anonymous reviewers and the editor for providing insightful comments that have helped us to improve this paper. We thank Dr. Shihong Yao for useful suggestions, and we also thank the anonymous proofreaders of this paper.

## Author Contributions

Conceived and designed the experiments: SMP ZQX YKL. Performed the experiments: SMP. Analyzed the data: SMP ZQX. Wrote the paper: SMP YKL ZQX YWC.

## References

1. Ghemawat S, Gobioff H, Leung ST. The Google file system. ACM SIGOPS Operating Systems Review, 2003, 37(5): 29–43.
2. Zhu YF, Jiang H, Qin X, Fend D, Swanson D. Exploiting Redundancy to Boost Performance in a RAID-10 Style Cluster-Based File System. Cluster Computing, 2006, 9(4): 433–447.
3. Gregory G, Stefano N, Anthony L, Nicolas R. WPS mediation: An approach to process geospatial data on different computing backends. Computers & Geosciences, 2012, 47: 20–33.
4. Pablo T, Chiara P, Jennifer J. Fish Farms at Sea: The Ground Truth from Google Earth. PLOS ONE, 2012, doi: 10.1371/journal.pone.0030546
5. Carns P, Lang S, Ross R, Vilayannur M, Kunkel J, Ludwig T. Small-file access in parallel file systems. In: Proceedings of IEEE International Symposium on Parallel & Distributed Processing, 23–29 May 2009, Rome, 1–11.
6. Barclay T, Gray J, Ekblad S, Strand E, Richter J. Designing and building TerraService. IEEE Internet Computer, 2006, 10(5): 16–25.
7. Lü XF, Cheng CQ, Gong JY, Guan Li. Review of data storage and management technologies for massive remote sensing data. Science China Technological Sciences, 2011, 54(12): 3220–3232.
8. Boschetti L, Roy DP, Justice CO. Using NASA's World Wind virtual globe for interactive internet visualization of the global MODIS burned area product. International Journal of Remote Sensing, 2008, 29 (11): 3067–3072.
9. Bell D G, Kuehnel F, Maxwell C, Kim R, Kasraie K, Gaskins T, *et al*. NASA World Wind: open-source GIS for mission operations. In: Proceedings of IEEE Aerospace Conference, 3–10 March 2007, Big Sky, MT, 1–9.
10. Gibin M, Singleton A, Milton R, Mateos P, Longley P. An exploratory cartographic visualization of London through the Google Maps API. Appl Spat Anal Pol, 2008, 1(2): 85–97.
11. Sample J T, Loup E. Tile-base geospatial information system: principle and practices. New York: Springer, 2010. 23–200.
12. Dong B, Qiu J, Zheng Q, Zhong X, Li J, Li Y. A Novel Approach to Improving the Efficiency of Storing and Accessing Small Files on Hadoop: a Case Study by PowerPoint Files. In: 2010 IEEE International Conference on Services Computing, 5–10 July 2010, Miami, FL, 65–72.
13. Shafer J, Rixner S, Cox AL. The hadoop distributed file system: balancing portability and performance. In: Proceedings of IEEE international symposium on performance analysis of system & software, 28–30 March 2010, White Plains, NY, 122–133.
14. McCulloch CM. Why merge?-An examination of disk sorting strategy. IEEE Transactions on Computers. 1985, C-34(4): 387–391.
15. Liu XH, Han JZ, Zhong YQ, Han CD, He XB. Implementing webgis on hadoop: a case study of improving small file I/O performance on HDFS. In: Proceedings of the IEEE international conference on cluster computing and workshops, Aug. 31—Sept. 4 2009, New Orleans, LA, 1–8.
16. Dong B, Zheng QH, Tian F, Chao KM, Ma R, Anane R. An optimized approach for storing and accessing small files on cloude storage. Journal of Network and Computer Application, 2012, 35: 1847–1862.
17. Hendricks J, Sambasivan RR, Sinnamohideen S, Ganger GR. Improving small file performance in object-based storage. Technical Report 06–104, Parallel Data Laboratory, Carnegie Mellon University, 2006.
18. James G, Randy A. Reducing file system latency using a predictive approach. Summer USENIX Technical Conference (Boston, MA), pages 197–207, 6–10 June 1994.
19. Thomas MK, Darrell DEL. The case for efficient file access pattern modeling. Hot Topics in Operating Systems (Rio Rico, Arizona, 29–30 March 1999), pages 14–19, 1999.
20. Danyel F. Hotmap: Looking at Geographic Attention. In: Proceedings of IEEE Transactions on Visualization and Computer Graphics, 2007, 13(6):1184–1191.
21. Wang H, Pan SM, Peng M. Zipf-like Distribution and its Application Analysis for Image Data Tile Request in Digital Earth. Geomatics and Information Science of Wuhan University, 2010, 35(3): 356–359.
22. Pan SM, YU ZW, Hao W, Li R. Tile Scheduling Algorithm based on Active Cache in P2P Massive Terrain Navigation. Acta Geodaetica et Cartographica Sinica, 2009, 38(3): 236–241.
23. Li R, Guo R, Xu ZQ, Feng W. A prefetching model based on access popularity for geospatial data in a cluster-based caching system. International Journal of Geographical Information Science, 2012, 26 (10): 1831–1844.

24. Ricardo GM, Juan PF, Elena VP, María JV, Luisa MR. An OLS regression model for context-aware tile prefetching in a web map cache. International Journal of Geographical Information Science. 2013, 27 (3): 614–632.

25. Cuthill E, Mckee J. Reducing the bandwidth of sparse symmetric matrices. In: Proceedings of the 1969 24th National Conference of the ACM, NY: Brandon Systems Press, 1969: 157–172.

26. Gibbs NE, Poole WG, Stockmeyer PK. An algorithm for reducing the bandwidth and profile of a sparse matrix. SIAM Journal on Numerical Analysis, 1976, 13(2): 236–250.

27. Yu ZW, Li ZM, Zheng S. Network Geographic Information System Architecture Based on Object-Based Storage. Geomatics and Information Science of Wuhan University, 2008, 33(3): 285–288.

28. Roselli DS, Lorch JR, Anderson TE. A Comparison of File System Workloads. In USENIX Annual Technical Conference, General Track. 2000: 41–54.

29. Yan A, Jianf P, Wu H, Ma L. Design and Realization of Multi-source Remote Sensing Images Data base in Lop Nur “Grea t Ear” Area. Journal of Geo-Information Science. 2009, 11(2): 250–255.

30. Weil SA, Leung AW, Brandt SA, Maltzahn C. RADOS: A Fast, Scalable, and Reliable Storage Service for Petabyte-scale Storage Clusters. In: Proceedings of the 2007 ACM Petascale Data Storage Workshop, November 2007.

31. Pan SM, Xu ZQ, Liu XJ. A dynamic statistical model for geospatial data access laws based on cloud computing. In: Proceedings of the 8th International Conference on Computer Science & Education, 26–28 April 2013, Colombo, Sri Lanka, 1422–1426.

32. Iztok F J, Karin L, Ponnuthurai N S, Matiaž P, Iztok F. Computational Intelligence in Sports: Challenges and Opportunities Within a New Research Domain. Applied Mathematics and Computation, 2015, 262: 178–186. doi: 10.1016/j.amc.2015.04.004

33. Samo R, Iztok F J, Iztok F. A Collection of Sport Activity Files for Data Analysis and Data Mining, Technical report 0201, University of Ljubljana and University of Maribor, 2015.

34. Zuckerman I, Albrecht DW, Nichilson A E. Predicting user’s requests on the WWW. In: Proceedings of the 7th international conference on user modeling, 20–24 June 1999, Banff, Canada, Springer, 275–284.

35. Lee DH, Kim JS, Kim SD, Kim KC, Kim YS, Park J. Adaptation of a Neighbor Selection Markov China for Prefetching Tiled Web GIS Data. In: Proceedings of the Second International Conference on Advances in Information System, 23–25 October 2002, Izmir Turkey. Berlin: Springer, 213–222.