A Web Cache Replacement Strategy for Safety-Critical Systems

Jianhai DU, Shiwei GAO, Jianghua LV, Qianqian LI, Shilong MA

Abstract: A Safety-Critical System (SCS), such as a spacecraft, is usually a complex system. It produces a large amount of test data during a comprehensive testing process. The large amount of data is often managed by a comprehensive test data query system. The primary factor affecting the management experience of a comprehensive test data query system is the performance of querying the test data. It is a big challenge to manage and maintain the huge and complex testing data. To address this challenge, a web cache replacement algorithm which can effectively improve the query performance and reduce the network latency is needed. However, a general-purpose web cache replacement algorithm usually cannot be directly applied to this type of system due to the low hit rate and low byte hit rate. In order to improve the hit rate and byte hit rate, a data stream mining technology is introduced, and a new web cache algorithm GDSF-DST (Greedy Dual-Size Frequency with Data Stream Technology) for the Safety-Critical System (SCS) is proposed based on the original GDSF algorithm. The experimental results show that compared with state of the art traditional algorithms, GDSF-DST achieves competitive performance and improves the hit rate and byte hit rate by about 20%.

Keywords: data mining; data query; Safety-Critical System; spacecraft; Web Cache Replacement Strategy

1 INTRODUCTION

Information technology significantly improves the efficiency of automated testing of the equipment, which leads to an explosive growth of the test data [1, 2]. In particular, more complex and extensive test data are generated in the comprehensive testing of the complex SCS (safety-critical systems), such as the spacecraft [3]. For each model of SCS, the total amount of comprehensive test data per year can reach the T order of magnitude, and the test data are various and multidimensional. These data need to be stored in multiple data tables. In each data sheet of a table there are tens of millions of data items [4]. Thus, managing and maintaining the huge and complex test data poses challenges. The primary factor that affects the system performance and the management experience is efficiently querying test data, which is one of the most frequent applications.

In a traditional B/S (Brower/Server) data querying system, data is retrieved from the database server for every query. Meanwhile, each query has to retrieve data from the entire database; therefore, a large number of queries from the large database may cause a significant time delay. This approach frequently transmits a large number of the same data between the database server and the application server, resulting in unnecessary network burden. In addition, for the equipment being tested, the database server needs to dynamically load a large amount of test data. The database server is accessed frequently, and thus the performance of the whole system is seriously affected. To ensure the stability of the server and the integrity of the data entry, data query should consume as few resources of the server as possible.

Web caching presents an effective way to solve the aforementioned problems. It plays an important role in improving Web performance by placing probably re-accessible objects closer to users, which can thus reduce the response time, the consumption of network bandwidth, and the workload of the original server [3]. An efficient web cache replacement algorithm for a particular application environment can also significantly affect the system performance and the user experience. The main criterion for the cache replacement algorithm is to minimize the metrics of average delay, error rate, and byte error rate. In a cache replacement algorithm, the main factors that need to be considered in the value function design include Recency, Frequency, Cost, Size, Expiration Time and Modification Time [6, 7]. Particularly, Recency refers to the last time when the object is referred. Frequency is the total number of times of object reference. Cost refers to the resources’ cost of acquiring the object from the original server. Size refers to the size of the object. Expiration Time refers to the expiration time of the object, and Modification Time refers to the time when the object is recently modified.

Considering the above factors, the cache replacement algorithm can be divided into the following five categories [6]. The first type is based on LRU (Least Recently Used), including LRU, LRU-Threshold (Least Recently Used-Threshold), Pitkow/Recker strategy, EXP1 (Exponential 1), Value-Aging, HLRU (History LRU), PSS (Pyramidal Selection Schema) [7], and Partitioned LRU. These algorithms are relatively easy to implement with low time complexity. However, the LRU variants are simple, and unable to effectively consider the effect of the object size or the access frequency. The second type of algorithms is based on LFU (Least Frequently Used) algorithms, including DSLFU (Delay sensitive LFU), LFU-Aging (Least Frequently Used-Aging), LFU-DA(LFU with Dynamic Aging), σ-Aging, swLFU (Server-Weighted LFU), DSLV (Dynamic Semantic LFU Policy with Victim tracer) [8], and ALFUR (Average Least Frequency Used Removal). These algorithms require complex cache management, and can cause cache pollution. There may exist multiple objects with the same frequency count at the same time, in which case additional factors are required to determine which object to remove. The third type of algorithms takes both recency and frequency into account. Some algorithms add other factors as well. These algorithms include LRFU (Least Recently/Frequently Used), SLRU (Segmented LRU), SF-LRU (Second ChanceFrequency—Least Recently Used), Generational Replacement, LRU* (Least Recently Used**), LRU-Hot (Least Recently Used-Hot), LRU-SP (LRU-size-adjusted and popularity-aware), and
learning based replacement algorithm (LBR) is proposed to build an efficient replacement model for web caching by incorporating a machine learning technique (naive Bayes) into the LRU replacement method to improve the prediction of possibility [14]. Fuzzy Similarity is integrated with traditional caching algorithms for caching replacement strategies [15]. A semi-intelligent approach is developed for web cache replacement using a multinomial web object classifier [16]. A Neural Network Proxy Cache Replacement (NNPCR) method is proposed to use neural networks to decide which objects are worth caching or being replaced by other objects [17]. Three replacement policies are introduced by following different strategies, including a dynamic programming approach, a branch and bound approach, and a heuristic approach, to find the best objects to be kept and the worst objects to be replaced, as well as to efficiently select objects to be evicted, respectively [18]. Specific cache replacement strategies have been improved and designed for specific fields in recent studies. Particularly, an enhanced method is proposed to actively cache the data for data-intensive computations in distributed GIS by considering both data relationships and their timeliness [19]. A novel cache scheme, named Optimized Cache Replacement, is proposed for Information Centric Networks (OCICN) [20]. For Network-Coding-Assisted Data Broadcast, a decoding-oriented cache management scheme, named decoding-oriented least recently used (DLRU), is proposed to incorporate decoding information in making cache replacement decisions [21].

However, these existing intelligent algorithms above have some common shortcomings. For instance, to achieve desirable performance for the intelligent algorithms, they usually need more system resources, a large number of data for training in advance and higher computational complexity. Thus, the systems based on these algorithms have low cost performance and mainly aim at ordinary users access web pages, multimedia, etc. in the World Wide Web. In addition, each type of algorithms has its own advantages and disadvantages. Even if some are the ones so-called "good enough" through which better results can be obtained on the basic performance indicators [6, 22], none can perform well in any application environment. The network environment is dynamic and uncertain. For instance, workload varies with different applications; a variety of Web access features exist and design goals for different cache systems and the design factors to be considered are not the same. No replacement strategies can well adapt to any network situation. Therefore, it has become a hot spot in the field of cache replacement which algorithm is the most appropriate replacement strategy according to varied network conditions and access characteristics [22].

Spacecraft integration test data are time-related, time-intensive, of large volume and complex structures. The users' query behaviors on the data are of strong temporal locality, strong spatial locality, high frequency (a small part of objects are accessed many more times than the others) and multiple visits (in order to ensure the completeness and the accuracy of the test, each data is accessed multiple times) [23]. Nevertheless, the existing web cache replacement algorithms are not suitable for integrated test data query systems for safety critical systems like Spacecraft. For this kind of system, a new
This study proposes a Web cache replacement algorithm based on GDSF algorithm, GDSF-DST (Greedy Dual-Size Frequency with Data Stream technology). The algorithm is proposed by analyzing the characteristics of test data of the safety-critical system and users' query behaviors. Considering the characteristics of practical application requirements and the current classical Web cache replacement algorithms, this algorithm also introduces technology related to data stream mining and adopts the idea of sliding time window. With comprehensive experiments and analysis, the results show that the algorithm presents better performance, with less consumption of system resources, higher efficiency, stability and certain universality.

2 BACKGROUND
2.1 The characteristics of Spacecraft Integrated Test Data and the Query Behaviours of Users

In the spacecraft integrated test database system, a variety of test data are stored in various types of database tables. The typical spacecraft integrated test data table structure includes the following characteristics. Taking time as the primary key, the comprehensive test data store the value [24] of each parameter at this time point. Specifically, the integrated test data of spacecraft have the following four characteristics.

(1) Large volume of data: About 20 G of data are produced per hour, and the amount of data produced per day can be up to 480 G. For a single spacecraft model, the total amount of its data can reach T orders of magnitude.

(2) Complicated data structures: The data types of spacecraft integrated test data are numerous. The subsystems each spacecraft model belongs to are various, and each subsystem has different parameters. So the data structures are very complicated. These complex data are stored in multiple database tables.

(3) The data are associated with time: In the database, the business data take time as the key. Logically, the data are time-correlated, that is, the time information is used as the retrieve condition when data are queried and analyzed. It emphasizes on the situation of the data at each time point in a certain period and the changing trend of the data throughout the whole time period.

(4) Time intensive: One piece of datum or more are produced per second. Even if the data within a relatively short period of time are queried, there is still a large number of query results. For example, query data within an hour are at least 3600 pieces.

From a macro point of view, users have some behavioral characteristics in querying the data of spacecraft integrated test system, which mainly include the following four aspects.

(1) The data queried by users are of strong temporal locality: For the case of the data query with a device being tested, the data queried in a particular day are usually limited to the data generated on that day, and thus the queried data extent evolves as the day goes by. For the case of the summary query, the data queried daily are generally limited to a period of time (1 week to 10 days) in the previous test phase, and the data are usually summarized and analyzed according to a certain time period.

(2) Users repeatedly visit the data: In order to ensure the integrity and the accuracy of the test, each datum will be queried many times.

(3) The data queried by users are of spatial locality: All the integrated test data of the current models are stored in the database server. However, the data queried by users are limited to a short test time. Therefore, being a small part of the data in the database, the accessed data are of spatial locality.

(4) Some objects are accessed frequently: the frequency of some objects accessed over a certain period of time is significantly larger than that of the others. For example, the part with a fault will be repeatedly checked by some staff.

2.2 Problem analysis

A query system is a Web application of B/S architecture in which data access is incentive; that is, each front-end operation will have a corresponding response. Due to the confidentiality of the spacecraft system, data backups are forbidden on an ordinary PC. And the large result set makes it unfeasible to cache the results on the browsers. Therefore, for each query operation, the results must be returned from the background. Based on the previous analysis of the spacecraft integration test data and the query characteristics of users, with the method of the Web server caching data, this algorithm successfully solves the problem that the same data have to be repeatedly transmitted between the browsers and the database, reducing the access load of the database server and the network traffic user query response time. Those methods referred to above are not perfectly suitable for spacecraft integrated test data query in a real-time environment. The method proposed in this paper is that all the data accessed by users within a business day are considered as an accessing scale unit. The time interval for accessing data within a business day is represented as timeInterval = [begin, end]. In order to ensure the continuity of the data in the cache, the data in the timeInterval in each original database table are logically divided into different small data tables by the hour. Then data can be obtained from the database server on a small data table scale. And each small data table is viewed as an object to be cached on the Web application server. Consequently, the core problem is that excellent cache replacement algorithms need to be designed to improve the performance of the system.

Depending on different locations in the Web, the Web cache can be categorized as a browser cache, a Web proxy cache, and an original server cache. The browser cache is on the client side, the original server cache is the cache of the server itself, and the Web proxy cache is on the Web proxy server. The Web proxy server lies between the user receiver and the original server, receiving requests from hundreds of users and replying the results to users. The Web proxy cache is widely used in Web
proxy servers, the performance of which has a direct influence on that of Web access. It is exactly the focus in this paper. Spacecraft system is a typical safety critical complex system. The common methods given in previous literature are not completely applicable to the spacecraft integrated test data query in a real-time environment. Therefore, it is of great significance to design an excellent cache replacement algorithm to improve the performance of the system.

3 GDSF ALGORITHM

GreedyDual algorithm was first proposed in the paper [25], and then based on GreedyDual algorithm, Pei Cao and Sandy Irani proposed GreedyDual-Size (GDS) algorithm [26]. The literature [27] proposed GDSF algorithm on the basis of GDS algorithm, which introduced the influence of the times of object visit on the performance of cache replacement algorithms. At present, GDSF algorithm, one of the most widely used cache replacement algorithms, is universally recognized as a "good enough" cache replacement algorithm with a better cache hit rate. It takes into account the access frequency of Web objects, the cost of acquiring objects and the influence of object sizes. By setting up the aging factor, the influence of the Web access temporal locality on cache replacement is perfectly fused into the algorithm. Whereas unable to adapt to the rapid changes of data stream, GDSF algorithm fails to get enough good cache hit rate. Web application is a typical data stream application [28], and Spacecraft integrated test data query system is a type of Web application. On the basis of GDSF algorithm, GDSF-DST algorithm, GDSF with Data Stream Mining, is put forward, with some methods used in GDSF algorithm, GDSF-DST algorithm will be presented in the next section.

The cache replacement algorithm GDSF uses the following function to calculate the eigenvalue of each Web object:

\[
V(I) = L + f_{freq}(I) \cdot \frac{\text{Cost}(I)}{\text{Size}(I)}
\]

where:
- \( I \) is a Web object.
- \( V(I) \) is the eigenvalue of the Web object \( I \).
- \( L \) is an aging factor whose initial value is 0. When cache replacement occurs, the value of \( L \) equals the eigenvalue of the latest replaced object.
- \( f_{freq}(I) \) represents the frequency count of the object \( I \) that is, the times the object has been requested before.
- When object \( I \) is hit by cache, its corresponding value is incremented by 1, otherwise \( f_{freq}(I) \) is equal to 1.
- \( \text{Size}(I) \) refers to the size of the object \( I \).
- \( \text{Cost}(I) \) stands for the cost of retrieving the object \( I \).

The pseudocode of the algorithm GDSF is shown in Tab. 1. When a cache replacement is needed, GDSF replaces the object with the smallest eigenvalue. It is obvious in Eq. (1) that the GDSF algorithm takes into consideration the influence of the object access frequency \( f_{freq}(I) \), the cost of retrieving the object \( \text{Cost}(I) \) and the object size \( \text{Size}(I) \). In addition, GDSF also takes into account the temporal locality of object access. So each time the object in the cache is hit, the eigenvalue \( V \) of the object will be updated. And when the cache replacement occurs, the value of the aging factor \( L \) is updated to the maximum \( V \) value in the eigenvalues of those evicted objects. Consequently, the value of the aging factor \( L \) is increasing gradually. By this way, the aging factor part in the eigenvalue function of the recently hit object is larger than that in the eigenvalue function of the object not having been accessed for a long time. And the influence of the temporal locality feature of the object access is integrated into the algorithm accordingly. Yet, it is beyond GDSF that users are more concerned about recent transactions and those older transactions have little effect on the transactions currently queried. However, it is rather costly to deal with those older ones. So the transaction data in a certain period of time may be of more significance. The application of sliding window model can just cope with the problem of data expiration. Hence, the algorithm GDSF-DST will be presented in the next section.

Table 1 GDSF algorithm

| cache replacement algorithm GDSF: GDSF(\( I_1, I_2, \ldots, I_m \)) |
|---------------------------------------------------------------|
| Input: The sequence of objects requested: \( I_1, I_2, \ldots, I_m \) |
| Output: The processing of each object by the cache replacement algorithm, and then return the requested object sequence. |
| 1. Initialize \( L \leftarrow 0 \); |
| 2. For each \( I \) in \( I_1, I_2, \ldots, I_m \) |
| 3. If \( I \) is already in cache then |
| 4. \( V(I) \leftarrow L + f_{freq}(I) \cdot \frac{\text{Cost}(I)}{\text{Size}(I)} \); |
| 5. End If; |
| 6. If \( I \) is not in memory then |
| 7. While there is not enough room in cache for \( In \) ; |
| 8. Evict \( I^z \); //\( I^z \) is the minimum \( V \) value of all the object in cache |
| 9. End While; |
| 10. \( L \leftarrow V(I^z) \); //\( I^z \) is the maximum \( V \) value of all the object evicted recently |
| 11. End If; |
| 12. bring \( I \) into cache ; |
| 13. \( V(I) \leftarrow L + f_{freq}(I) \cdot \frac{\text{Cost}(I)}{\text{Size}(I)} \); |
| 14. End For; |
| 15. return the requested object sequence; |

4 GDSF-DST

4.1 Related definitions

Before the explanation of the algorithm, the concepts to be used in it are introduced firstly.

Let \( I = \{I_1, I_2, \ldots, I_m\} \) be a set of data items.

Definition 1: transaction \( T_j = \{T_{j_1}, T_{j_2}, \ldots, T_{j_k}\} \) is a subset of \( I \), namely, \( T_j \subseteq I \). And each transaction has its own identifier called TID. The data items in the transaction are out of order and can be arranged in any order.

Definition 2: data stream \( DS = \{T_{j_1}, T_{j_2}, \ldots, T_{j_m}\} \) consists of transactions sequentially arranged in time order. The number of transactions may be infinite. The
TID of each transaction is assigned in terms of the arrival
time and it is unique.

Definition 3: The sliding window SW [29] is the
sequence of the most recent \( N \) \((N \geq 0)\) transactions in the
data stream \(DS\). The order of the transactions in the
sliding window is consistent with that in the data stream
\(DS\). Depending on whether \( N \) is fixed or variable, the
sliding window can be classified into fixed-sized sliding
window and variable-sized sliding window. The
commonly used variable-sized sliding window is mainly
based on the time sliding window, which means that
transactions for the most recent time period (e.g. 1 hour)
are saved in the window. The operation in sliding
windows involves deleting the oldest transactions from a
sliding window and adding new ones to it. When the
number of transactions in the window reaches \( N \), removal
and insertion in a fixed-sized sliding window must be
simultaneous, while it is un restrained in a variable-size
one [30].

In this paper, a transaction represents a set of the
objects requested by a user with a Web click. Each data
item table in the set denotes a Web object. The data
objects requested by a user with a Web click. Each data

4.2 GDSF-DST Algorithm Description

The eigenvalue function of the algorithm GDSF-DST

\[
\begin{align*}
V(I, T_j) &= L + f_j(I, T_j) \cdot \frac{Cost(I_n)}{Size(I_n)}, \quad (T_{oldest} \geq T_j \leq T_{last}) \\
V(I, T_j) &= 0, \quad (T_{oldest} < T_j, T_j \leq T_{last})
\end{align*}
\]

Where:

\[
f_j(I, T_j) = \begin{cases} 
 r & \text{if } j = 1 \\
 f_j(I, T_j) + f + r & \text{if } j \geq 2
\end{cases}
\]

\[
r = \begin{cases} 
 1 & \text{if } I_n \in T_j \\
 0 & \text{if } I_n \not\in T_j
\end{cases}
\]

In Eq. (2), the meanings of \( L \), \( Cost(I_n) \) and \( Size(I_n) \)
are the same as those in formula (1). \( V(I_n, T_j) \) represents the
eigenvalue of object \( I_n \) when transaction \( T_j \) arrives. \( f(I_n, T_j) \)
denotes the decay support amount of object \( I_n \) when
transaction \( T_j \) arrives. \( T_{oldest} \geq T_j \leq T_{last} \) and \( T_{oldest} < T_j \) or
\( T_j \leq T_{last} \) represent the sliding time window, which
introduces the impact of expired transactions on the
algorithm.

Web object access is of temporal locality, namely, the
closer from now the object is accessed, the more likely it
is to be accessed in the future. In other words, the Web
click stream is bound to change over time, and commonly
the information contained in recently generated
transactions is more valuable than it in historical
transactions [31]. Therefore, the time decay model support
amount algorithm [32, 33] is proposed to
gradually lower the impact of historical transactions. The
decay ratio of data items in unit time is decay factor \( f(0 < \]

\[
f \leq 1).\ When\ transaction\ \( T_j \) \ arrives,\ the\ decay\ support
amount of\ data\ item\ \( I_n \) is \( f(I_n, T_j) \). Finally, the decay
support amount of data item \( I_n \) can be calculated by Eq.

4.3 Algorithm Flow of GDSF-DST

Before introducing the algorithm flow, the data
structure to be used in the algorithm is given first:

A sliding time window is a time-based one able to
store the ordered transactions arriving in the most recent
variable period (time). The members of the sliding time
window are represented by a data structure called \( SwItem\).
Fields include a submission time \( subTime \) and a
transaction TID. When a new transaction \( T_j \) arrives, the
sliding time window requires updating. Encapsulate its
fields into \( SwItem \), insert the \( SwItem \) into the back of the
queue, check from the beginning of the queue and then
delete all the expired transactions. \( oldestTid \) and
\( newestTid \) respectively represent the transaction TID in
the head of the sliding time window queue and the one in
the end of the queue (\( newestTid \) minus \( oldestTid \) equals
the length of the sliding window (time)).

\( fdHash \) refers to object information Hash table, which
is used to store the decay support amount of Web objects
and other related information and expressed in the form of
key value pairs. Given the Web object name is the
primary key, the value is a data structure called
\( ItemNode \), \( <ItemName, ItemNode> \). \( ItemNode \) contains five field: \( tid \) (the TID of the latest
transaction consisting of object \( ItemName \)), count (the
decay support amount), size (the object size), value (the
object eigenvalue) and cost (the cost of gaining the object
from the original server). Recording \( tid \) is to judge
whether a transaction is an outdated one, and to calculate
the decay support amount of the object as well.

When a new transaction \( T_j \) arrives, the GDFS-DSM
algorithm flow for each object \( I_n \) in the transaction
is depicted as follows:

If object \( I_n \) exists in the cache and is hit, the decay
support amount of the object \( I_n \) is updated:

\[
count = count \ast f^\text{tid} + 1
\]

(4)

(The decay support amount \( count \) has the same
meaning as \( f(I_n, T_j) \) in Eq. (2), \( f \) is the decay factor, which
equals with that in Eq. (3). The range of \( f \) is \((0,1)\). When
the value of \( f \) is 1, no decay happens.)

Update the TID recently contained by the object \( I_n \):
\( tid=j \); update the eigenvalue \( V(I_n, T_j) \) of the object \( I_n \) by
Eq. (2); and the value of \( Used \) remains unchanged. Then
the processing of the object ends in the algorithm.

If the object \( I_n \) does not exist in the cache, the decay
support amount of the object \( I_n \) equals 1: \( count=1 \). The
TID of the transaction recently containing the object \( I_n \) is \( j \):
\( tid=j \).

Compute the eigenvalue \( V(I_n, T_j) \) of the object \( I_n \) by
Eq. (2) and save the relevant information of the object \( I_n \)
to the object information hash table \( fdHash \):

\[
Used = Used + Size(I_n)
\]

(5)
If \( \text{Used} \leq \text{TotalSize} \), there is adequate space in the cache to save the object \( I_n \). In this case, store the object \( I_n \) into the cache directly. The process of the object is over.

If \( \text{Used} > \text{TotalSize} \), the cache space is inadequate. In this case, a cache replacement is required. So expired objects need picking out one by one in \( \text{fdHash} \). (As to an \( \text{Item Node} \) in \( \text{fdHash} \), if \( \text{Item Node}.\text{tid} < \text{oldestTid} \), then the object corresponding to that node is an expired object). Once an expired object \( I_e \) is found, perform as the following steps:

Eliminate the object \( I_e \) from the cache and its relevant information from \( \text{fdHash} \).

\[
\text{Used} = \text{Used} - \text{Size}(I_e) \tag{6}
\]

Continue to search for the next expired object; and repeat the above steps until \( \text{Used} \leq \text{TotalSize} \) or no expired objects exist in \( \text{fdHash} \).

If \( \text{Used} \leq \text{TotalSize} \), it means that there is enough cache space to store the object \( I_n \) after removing the expired object \( I_e \). Store the object \( I_n \) in the cache. The processing of the object ends in the algorithm.

If \( \text{Used} > \text{TotalSize} \), it means there is still not enough cache space to store the object \( I_n \), after removing all expired objects. In this case, traverse \( \text{fdHash} \), select the least \( k (k \geq 1) \) eigenvalues (when the eigenvalues are the same, select the objects containing the least \( \text{tid} \) preferentially) and find their corresponding objects \( I_1, I_2, \ldots, I_k \) which meet the following three conditions:

\[
V(I_1, T_j) \leq V(I_2, T_j) \leq \cdots \leq V(I_k, T_j) \tag{7}
\]

\[
\text{Used} - \sum_{x=1}^{k} \text{Size}(I_x) \leq \text{TotalSize} \tag{8}
\]

\[
\text{Used} - \sum_{x=1}^{k} \text{Size}(I_x) > \text{TotalSize} \tag{9}
\]

The aging factor \( L \) is assigned to the maximum value \( V(I_i, T_j) \), that is,

\[
L = \max_{1 \leq x \leq k} V(I_x, T_j) = V(I_k, T_j) \tag{10}
\]

Remove objects \( I_1, I_2, \ldots, I_k \) from the cache. And update the value of \( \text{Used} \):

\[
\text{Used} = \text{Used} - \sum_{x=1}^{k} \text{Size}(I_x) \tag{11}
\]

Clear away the relevant information of the objects \( I_1, I_2, \ldots, I_k \) from \( \text{fdHash} \). Add the object \( I_n \) to the cache. The process of the object is over. The above steps are looped until all objects in transaction \( T_j \) are processed. Return to the set of objects for the transaction request, and the processing ends. The pseudocode of the algorithm GDSF-DST is shown in Tab. 2.

In the algorithm GDSF-DST, the concept of sliding window model in data stream mining is employed. The idea of the sliding window model is originated from that it is impractical to store all the transactions in the data stream with limited memory size. Besides, the large time overhead for processing all historical transactions is unacceptable. Most importantly, users are more concerned about the most recent transactions. And the application of the sliding window model can exactly address the problem of data expiration.

| Table 2 | GDSF-DST algorithm |
|---|---|
| cache replacement algorithm GDSF-DST: GDSF-DST\( (T_j) \) |
| Input: Transaction \( T_j \) |
| Output: Perform a cache replacement algorithm on each Web object in the transaction, and return the requested object set for the transaction |
| 1. Initialize parameters; |
| 2. For each \( I_n \) in \( T_j \); |
| 3. If \( I_n \) is already in cache then |
| 4. \( \text{count}(I_n) \leftarrow \text{count}(I_n) + 1; \) |
| 5. \( \text{tid}(I_n) \leftarrow j; \) |
| 6. \( V(I_n, T_j) \leftarrow L + \text{count}(I_n) \times \text{Cost}(I_n) / \text{Size}(I_n); \) |
| 7. Add the object \( I_n \) to result list; |
| 8. continue; |
| 9. Else/Create information of the object \( I_n \) |
| 10. \( \text{count}(I_n) \leftarrow 1; \) |
| 11. \( \text{tid}(I_n) \leftarrow j; \) |
| 12. \( V(I_n, T_j) \leftarrow L + \text{count}(I_n) \times \text{Cost}(I_n) / \text{Size}(I_n); \) |
| 13. End If; |
| 14. \( \text{Used} \leftarrow \text{Used} + \text{Size}(I_n); \) |
| 15. If \( \text{Used} \leq \text{TotalSize} \) then |
| 16. Save the object \( I_n \) to the cache; |
| 17. Add the object \( I_n \) to result list; |
| 18. continue; |
| 19. Else |
| 20. While there is expired object \( I_e \) in the cache? |
| 21. Evict \( I_e \); \( //I_e \) is the expired object in cache; |
| 22. End while; |
| 23. End If; |
| 24. If \( \text{Used} \leq \text{TotalSize} \) then |
| 25. Save the object \( I_n \) to the cache; |
| 26. Add the object \( I_n \) to result list; |
| 27. continue; |
| 28. Else |
| 29. While \( \text{Used} > \text{TotalSize} \) |
| 30. Evict \( I_{\text{max}}; \) \( //I_{\text{max}} \) is the minimum \( V \) value |
| 31. \( \text{Used} \leftarrow \text{Used} - \text{Size}(I_{\text{max}}); \) |
| 32. End while; |
| 33. \( L \leftarrow V(I_{\text{max}}); \) \( //I_{\text{max}} \) is with the maximum \( V \) value of all the object evicted recently |
| 34. Save the object \( I_n \) to the cache; |
| 35. Add the object \( I_n \) to result list; |
| 36. End If; |
| 37. End For; |
| 38. return the result list; |

5 EXPERIMENTS
5.1 Experimental Design
5.1.1 Experimental Environment

In this experiment, the algorithm is implemented in Java language on the DELL PC with Microsoft Windows 7(CPU model is Intel i5-7500; the CPU frequency 3.4 GHz; the capacity of the physical memory 4 GB; the hard drive Seagate 500G SATA3 and the hard drive speed 7200 rpm).

5.1.2 Experimental Data

In this paper, the Trace Driven Simulation \([5, 34]\) is adopted to simulate the algorithm. The experimental data
are derived from the query log information recorded by the server of the spacecraft integrated test data query system. Not all the original query log information can be published, some of which is indeed unnecessary. As a result, several processed Web query log files are employed in this paper. The file naming specification is WebLog + log number, twenty-one query records are stored in each log file. Spacecraft integrated test data are made up of quite a few data streams, data types with huge volume of data. For ease of operation in this experiment, four representative data streams are selected. In the above log files, each record represents a query and is treated as a transaction. And a transaction usually accesses multiple Web objects. To facilitate operation in the cache replacement algorithm, each of the mentioned log files is parsed into a transaction file respectively. The transaction file naming format is TransactionFlow + the number of the corresponding log file. Each log is parsed into a transaction record. In the experiment, log data during 15 days, from September 1th, 2017 to September 15th, 2017 are employed. Since the log data statistics tables will occupy too much space, they are omitted here.

5.1.3 Performance Indicators

The commonly used indicators to measure the performance of the cache replacement algorithm [22] mainly include Hit Ratio (HR), Byte Hit Ratio (BHR) and Delay Saving Radio (DSR). In this paper, Hit Radio (HR) and Byte Hit Ratio (BHR), the most commonly used ones, are adopted. Hit rate is an indication of user perceived lag, while byte hit rate is an indication of the amount of network traffic [35]. The total number of the requested Web objects is represented by \( N \). \( S_i \) represents the size (the number of bytes) of the \( i \)th requested Web object. \( \sigma_i \) indicates whether the \( i \)th object is hit in the cache. If it is hit, \( \sigma_i = 1 \); otherwise \( \sigma_i = 0 \). \( L \) represents the download time required to obtain the object \( i \) from the original server. Definitions are as follows:

Hit Ratio (HR): The percentage for the number of the objects hit in the cache and the total number of the objects requested.

\[
HR = \frac{\sum_{i=1}^{N} \sigma_i}{N}
\]

The calculation formula is: \( HR = \frac{\sum_{i=1}^{N} \sigma_i}{N} \)

Byte Hit Ratio (BHR): The percentage for the number of bytes of the objects hit in the cache and the total number of bytes of the objects requested.

\[
BHR = \frac{\sum_{i=1}^{N} \sigma_i S_i}{\sum_{i=1}^{N} S_i}
\]

The calculation formula is: \( BHR = \frac{\sum_{i=1}^{N} \sigma_i S_i}{\sum_{i=1}^{N} S_i} \)

5.2 Experimental Analysis

From the previous algorithm description, it can be seen that the parameters used in both GDSF and GDSF-DST -- the aging factor \( L \), the frequency count \( f_{red}(i) \) of the object, the decay support amount \( f_d(L, r, T) \) of the object and the size \( \text{Size}(i) \) of the object -- are defined clearly. However, measuring the cost \( \text{Cost}(i) \) of retrieving the object is complicated. The cost refers to anything required to obtain the object from the original server, such as the time delay, the expenditure (paid objects or paid periodic lines), the number of hops, etc., or a combination of multiple ones. In practice, the value of the \( \text{Cost}(i) \) is often defined according to the target of the caching system. According to the reference [26], to maximize \( HR \), let \( \text{Cost}(i) = 1 \). And to maximize \( \text{BHR} \), let \( \text{Cost}(i) = 2 + \text{Size}(i)/536 \). In the spacecraft integrated test data query system, the original data are stored on the same server, and the data server and the Web server are in the intranet environment. Therefore, the time cost of retrieving different objects from the original server is not very different, and there is no need to consider the economic cost. In this case, the value of \( \text{Cost}(i) \) is defined as 1 so as to achieve a better hit ratio.

5.2.1 The Influence of Decay Factor \( f \) on Hit Ratio and Byte Hit Ratio

Before considering the influence of decay factor \( f \) on the performance of the algorithm, the influence of the sliding time window size \( \text{STW SIZE} \) should be shielded first. The actual query feature of Spacecraft integrated test data query system is that users usually query data between 8 am and 8:30 pm. When \( \text{STW SIZE} \) is set to 13 hours, there are no expired transactions within a day, that is, the sliding time window does not work. Therefore, in the experiment in this section, \( \text{STW SIZE} \) is set to 13 hours. The literature [22] demonstrates that a better hit ratio can be achieved in the cache replacement algorithm with the relative capacity of the cache reaching 20%. Therefore, in this section, the relative size of the cache capacity \( \text{TotalSize} \) is set to 20% of the total amount of access. The algorithm GDSF-DST is experimented with daily log data from September 1th, 2017 to September 10th, 2017. When the decay factor \( f \) is respectively set to 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 0.991, 0.992, 0.993, 0.994, 0.995, 0.996, 0.997, 0.998, 0.999 and 1, the values of \( HR \) and \( BHR \) are shown in Fig. 1 and Fig. 2 respectively. In Fig. 1 and Fig. 2, the abscissa represents the values of the decay factor \( f \) while the ordinate represents the values of the hit ratio and those of the byte hit ratio respectively. Obviously, in Fig. 1, the best hit ratio can be obtained with values of the decay factor \( f \) in the range \([0.996, 0.997]\). When the values of \( f \) are in the range \([0.94, 0.996]\), the hit ratio increases with the increase of the values of \( f \). When the values of \( f \) are in the range \([0.997, 1]\), the hit ratio decreases with the increase of the values of \( f \). The relationship between the average hit ratio and the values of \( f \) also presents the same rule. In Fig. 2, when \( f = 0.996 \) or \( f = 0.997 \), the byte hit ratio is the maximum. When the values of \( f \) are in the range \([0.94, 0.996]\), the byte hit ratio increases with the increase of the values of \( f \). When the values of \( f \) are in the range \([0.997, 1]\), the byte hit ratio decreases with the increase of the values of \( f \). The relationship between the average byte hit ratio and the values of \( f \) also presents the same rule. When the value of \( f \) changes, hit ratio and byte hit ratio have the same trend. The main reason why the decay factor \( f \) has a significant impact on both the hit ratio and the byte hit ratio of the algorithm is that the data queried by the users of the spacecraft integrated test query system are of strong
temporal locality. And the decay factor $f$ can speed up the reduction of the eigenvalues of long-history data, so it can adapt to the user query behavior characteristics of the spacecraft integrated test query system better.

On the basis of the query temporal locality. And the decay factor $f$ can speed up the reduction of the eigenvalues of long-history data, so it can adapt to the user query behavior characteristics of the spacecraft integrated test query system better.

Because a query usually involves a wide range of objects, transactions in a short time can contain most objects, which makes a large number of the objects non-expired. The smaller sliding windows are, the fewer transactions exist. Therefore, in this experiment, $STW\_SIZE$ is set to 13 h, 12 h, 11 h, 10 h, 9 h, 8 h, 7 h, 6 h, 5 h, 4 h, 3 h, 2 h, 1 h, 30 m, 20 m, 10 m, 9 m, 8 m, 7 m, 6 m, 5 m, 4 m, 3 m, and 2 m. (h stands for hours, and m stands for minutes).

Tab. 3 presents the experimental data of average hit ratio and $SW\_SIZE$ and Tab. 4 presents the experimental data of average byte hit ratio and $SW\_SIZE$. Fig. 3, drawn from the data in Tab. 3, presents the relationship between the hit ratio and the size of $STW\_SIZE$, in which the ordinate indicates the hit ratio and the abscissa indicates the sliding window size $STW\_SIZE$. The units of $STW\_SIZE$ are minutes. In order to show the changes more clearly, this plots the logarithm to base 10 of the values on the abscissa. Fig. 4, drawn from the data in Tab. 4, presents the relationship between the byte hit ratio and the size of $STW\_SIZE$, in which the ordinate indicates the byte hit ratio and the abscissa indicates the sliding window size $STW\_SIZE$. The units of $STW\_SIZE$ are minutes. Evidently, in Fig. 3 and Fig. 4, with the sliding window size declining from 10 hours to 8 minutes, GDSF-DST can achieve great hit ratio and byte hit ratio, and the performance index remains stable. When the size of the sliding window is greater than or equal to 13 hours, no expired transactions exist. Therefore, in this experiment, $STW\_SIZE$ is set to 13 h, 12 h, 11 h, 10 h, 9 h, 8 h, 7 h, 6 h, 5 h, 4 h, 3 h, 2 h, 1 h, 30 m, 20 m, 10 m, 9 m, 8 m, 7 m, 6 m, 5 m, 4 m, 3 m, and 2 m. (h stands for hours, and m stands for minutes).

5.2.2 The Influence of the Sliding Window Size $STW\_SIZE$ on Hit Ratio and Byte Hit Ratio

Through the previous experimental analysis, it can be seen that the best hit ratio can be achieved in GDSF-DST when the values of $f$ are in the range [0.996, 0.997]. In this section, let $f = 0.996$. And the relative size of the cache capacity $Total\_Size$ is still set to 20% of the total amount of access. The algorithm GDSF-DST is experimented with daily log data from September 1st, 2017 to September 10th, 2017. On the basis of the query

As the size of the sliding window is reduced from 10 hours to 8 minutes, the hit ratio and the byte hit ratio remain stable. Because a query usually involves a wide

![Figure 1 The relationship between HR and $f$](image1)

![Figure 2 The relationship between BHR and $f$](image2)

| $STW\_SIZE$ / min | 780 | 720 | 660 | 600 | 540 | 480 | 420 | 360 | 300 | 240 | 180 | 120 |
|-------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| The average hit ratio | 0.565 | 0.566 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 |
| The average hit ratio | 0.523 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 | 0.567 |

| $STW\_SIZE$ / min | 780 | 720 | 660 | 600 | 540 | 480 | 420 | 360 | 300 | 240 | 180 | 120 |
|-------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| The average byte hit ratio | 0.539 | 0.539 | 0.540 | 0.540 | 0.540 | 0.540 | 0.540 | 0.540 | 0.540 | 0.540 | 0.540 | 0.540 |
| The average byte hit ratio | 0.581 | 0.693 | 0.668 | 0.668 | 0.668 | 0.668 | 0.668 | 0.668 | 0.668 | 0.668 | 0.668 | 0.668 |

![Table 3 The relationship between HR and $STW\_SIZE$](image3)

![Table 4 The relationship between BHR and $STW\_SIZE$](image4)
the stored transactions are, thereby reducing the spatial complexity of the algorithm. Therefore, under the premise of ensuring the performance of the algorithm, the smaller time window should be selected. In Fig. 3 and Fig. 4, the red data point is the one of the sliding window size as 10 minutes, and the performance of the algorithm is stable before and after the point. Therefore, for the workload of the spacecraft integrated test data query system, the sliding window size can be selected as 10 minutes.

5.2.3 The Influence of the Cache Capacity \( \text{TotalSize} \) on Hit Ratio and Byte Hit Ratio

Based on the previous analysis, in this section, the sliding window size \( \text{STW\_SIZE} \) is set to 10 minutes, and the decay factor \( f \) is set to 0.996. The cache capacity relative size \( \text{TotalSize} \) is set to 1%, 2%, 3%, 4%, 5%, 10%, 15%, 20%, 25%, and 30%, respectively, and then experiments are carried out. The log data used for the experiments are the log records during 5 days, from September 11\(^{th}\), 2017 to September 15\(^{th}\), 2017. In Fig. 5, the ordinate represents the value of the hit ratio and the abscissa indicates the relative capacity of the cache. In Fig. 6, the ordinate represents the value of the byte hit ratio and the abscissa indicates the relative capacity of the cache. It can be seen from the figures that the hit ratio and the byte hit ratio increase rapidly with the increase of the cache capacity \( \text{TotalSize} \). When the relative capacity of the cache reaches 30%, the hit ratio and the byte hit ratio can exceed 70%.

5.2.4 Comparison with the Classical Algorithms

In this section, the sliding window size \( \text{STW\_SIZE} \) in the algorithm GDSF-DST is set to 10 minutes and the decay factor \( f \) is set to 0.996. The log data used for the experiments are the log records during 5 days, from September 11\(^{th}\), 2017 to September 15\(^{th}\), 2017. The relative cache capacity \( \text{TotalSize} \) is set to 1%, 2%, 3%, 4%, 5%, 10%, 15%, 20%, 25%, and 30%, respectively, and experiments are carried out respectively on algorithms GDSF-DST, GDSF, LFU, LFU-DA and LRU.

Fig. 7 shows the curves of hit ratio of each algorithm varying with the cache capacity. Fig. 8 shows the curves of the byte hit ratio of each algorithm varying with the
cache capacity. Because the decay factor \( f \) and the sliding window mechanism are introduced for the users’ query behavior characteristics of the spacecraft integrated test query system, both the hit ratio and the byte hit ratio in GDSF-DST algorithm are better than those in other algorithms (Fig. 7 and Fig. 8).

6 CONCLUSIONS

(1) In this paper, the characteristics of data in spacecraft integration test data query system and users’ query behaviors are analyzed. Under the premise of full investigation of the cache technology research progress, combined with the relevant technology of Web data stream mining, a cache replacement algorithm, GDSF-DST, is designed for spacecraft integrated testing data query system.

(2) The specific process of the algorithm implementation is presented and the Trace Driven Simulation experiment is performed with the query log data in the actual runtime environment.

(3) The newly-designed algorithm is fully analyzed and compared with the existing typical cache replacement algorithms, and it is proved that the algorithm has better performance.

The deficiency in this paper and further work is to evaluate the validity of the algorithm with Data Envelopment Analysis (DEA). The method proposed in this paper is only for the spacecraft integration test data query. In future, we will collect data in other fields except spacecraft comprehensive test, such as stock trading, video websites, news websites, and some other data applications with large volume of data, rapid changes and the latest data changes which public users are more concerned about. And then we will discuss the application and expansion of GDSF-DST algorithm proposed in this paper in these fields. The new algorithm proposed in this paper is integrated with the algorithms of machine learning to further improve the performance index of cache replacement strategy and to verify the applicability of the new intelligent algorithms in other environments.

ACKNOWLEDGEMENTS

This work was supported by the National Natural Science Foundation of China under Grant Nos. U1736116 and 61300007, the Fundamental Research Funds for the Central Universities of China under Grant Nos. YWF-15-GJSYS-106 and YWF-14-JSJXY-007, and the Project of the State Key Laboratory of Software Development Environment of China under Grant Nos. SKLSDE 2015ZX-09 and SKLSDE-2014ZX-06.

7 REFERENCES

[1] Djurovic, D., Bulatovic, M., Sokovic, M., & Stoic, A. (2015). Measurement of maintenance excellence/Mjerenje izvrsnosti odrzavanja. Tehnicki vjesnik-Technical Gazette, 22(5), 1263-1269. https://doi.org/10.17559/TV-20140922094945

[2] Blagovjević, M., Rakić, D., Topalović, M., & Živković, M. (2016). Optical coordinate measurements of parts and assemblies in automotive industry. Tehnicki vjesnik/Technical Gazette, 23(5), 1541-1546. https://doi.org/10.17559/TV-20130918160442

[3] Lv, J., Ma, S., Li, X., et al. (2015). A high order collaboration and real time formal model for automatic testing of safety critical systems. Frontiers of Computer Science, 9(4), 495-510. https://doi.org/10.1007/s11704-015-2254-y

[4] Gao, S. W., Lv, J. H., Du, B. L., et al. (2015). Balancing Frequencies and Fault Detection in the In-Parameter-Order Algorithm. Journal of Computer Science and Technology, 30(5), 957-968. https://doi.org/10.1007/s11390-015-1574-6

[5] Kastaniotis, G., Maragos, E., Douligeris, C., et al. (2012). Using data envelopment analysis to evaluate the efficiency of Web caching object replacement strategies. Journal of Network and Computer Applications, 35(2), 803-817. https://doi.org/10.1016/j.jnca.2011.11.013

[6] Podlipnig, S. & Bösztörményi, L. (2003). A survey of web cache replacement strategies. ACM Computing Surveys (CSUR), 35(4), 374-398. https://doi.org/10.1145/954339.954341

[7] Aggarwal, C., Wolf, J. L., & Yu, P. S. (1999). Caching on the world wide web. IEEE Transactions on Knowledge and Data Engineering, 11(1), 94-107. https://doi.org/10.1109/69.755618

[8] Geetha, K. & Gounden, N. A. (2017). Dynamic Semantic LFU Policy with Victim tracer (DSLV): A Customizing Technique for Client Cache. Arabian Journal for Science and Engineering, 42(2), 725-737. https://doi.org/10.1007/s13369-016-2287-z

[9] Wang, S., Li, Z., Chao, W., et al. (2012). Applying adaptive over-sampling technique based on data density and cost-sensitive SVM to imbalanced learning. International Joint Conference on Neural Networks. IEEE, 2, 1-8. https://doi.org/10.1109/IJCNN.2012.6252896

[10] Wang, S., Li, Z., Zhang, X. (2013). Bootstrap Sampling Based Data Cleaning and Maximum Entropy SVMs for Large Datasets. IEEE, International Conference on TOOLS with Artificial Intelligence. IEEE, 1(11), 1151-1156. https://doi.org/10.1109/ICTAI.2012.164.

[11] Li, Y., Zhou, G., Nie, B. (2017). Improving Web Performance in Home Broadband Access Networks. Wireless Personal Communications, 92(3), 925-940. https://doi.org/10.1007/s11227-016-3585-1

[12] Ali, W., Shamsuddin, S. M., & Ismail, A. S. (2012). Intelligent Web proxy caching approaches based on machine learning techniques. Decision Support Systems. 53(3), 565-579. https://doi.org/10.1016/j.dss.2012.04.011

[13] Ali, W., Shamsuddin, S. M., & Ismail, A. S. (2012). Intelligent Naive Bayes-based approaches for Web proxy caching. Knowledge-Based Systems, 31, 162-175. https://doi.org/10.1016/j.knosys.2012.02.015

[14] Songwattana, A., Theeramunkong, T., & Vinh, P. C. (2014). A learning-based approach for web cache management. Mobile Networks and Applications, 19(2), 258-271. https://doi.org/10.1007/s11263-014-0498-7

[15] Lee, C. (2013). Streaming media service based on fuzzy similarity in wireless mobile networks. The Journal of Supercomputing, 65(1), 86-105. https://doi.org/10.1007/s11227-012-0778-6

[16] Sajeev, G. P. & Sebastian, M. P. (2013). Building semi-intelligent web cache systems with lightweight machine learning techniques. Computers & Electrical Engineering, 39(4), 1174-1191. https://doi.org/10.1016/j.compeleceng.2013.02.005

[17] Romano, S. & ElAarag, H. (2011). A neural network proxy cache replacement strategy and its implementation in the Squid proxy server. Neural computing and Applications, 20(1), 59-78. https://doi.org/10.1007/s00521-010-0442-0

[18] Rashkovits, R. (2016). Preference-based content replacement using recency-latency tradeoff. World Wide Web, 19(3), 323-350. https://doi.org/10.1007/s11280-014-0313-1
[19] Pan, S., Chong, Y., Xu, Z., et al. (2017). An enhanced active caching strategy for data-intensive computations in distributed GIS. *The Journal of Supercomputing*, 73(10), 4324-4346. https://doi.org/10.1007/s11227-017-2012-2

[20] Sun, X. C., Wang, Z. J., Chu, H., et al. (2016). An Efficient Resource Management Algorithm for Information Centric Networks. *Journal of Internet Technology*, 17(5), 1007-1015. https://doi.org/10.6138/JIT.2016.17.5.20160111

[21] Chen, J., Lee, V. C. S., Liu, K., et al. (2017). Efficient Cache Management for Network-Coding-Assisted Data Broadcast. *IEEE Transactions on Vehicular Technology*. 66(4), 3361-3375. https://doi.org/10.1109/TVT.2016.2589460

[22] Kastaniotis, G., Maragos, E., Dimitras, V., et al. 2007. Web Proxy caching object replacement: Frontier analysis to discover the good-enough algorithms. *Proceedings of the 2007 15th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. IEEE Computer Society*, 132-137, https://doi.org/10.1109/MASCOTS.2007.88

[23] Gao, S. W., Lv, J. H., Wuniri, Q. Q. G., et al. (2015). Spacecraft Test Requirement Description and Automatic Generation Method. *Journal of Beijing University of Aeronautics and Astronautics*, 41(7), 1275-1286. https://doi.org/10.13700/j.bh.1995.2014.0762

[24] Chen, H. (2007). Design and Implementation of Integration Test Data Platform for Manned Spacecraft. *PhD. Thesis*, Beijing University of Aeronautics and Astronautics, Beijing.

[25] Young, N. (1994). Threk-server dual and loose competitiveness for paging. *Algorithmica*, 11(6), 525-541. https://doi.org/10.1007/BF01199992

[26] Cao, P. & Irani, S. (1997). Cost-aware www proxy caching algorithms. *Usenix Symposium on internet technologies and systems*, 12(97), 193-206.

[27] Cherkasova, L. (1998). Improving www proxies performance with greedy-dual-size-frequency caching policy. *Hp Technical Report*.

[28] Ferrara, E., De Meo, P., Fiumara, G., et al. (2014). Web data extraction, applications and techniques: A survey. *Knowledge-based systems*, 70, 301-323. https://doi.org/10.1016/j.knosys.2014.07.007

[29] Li, H. F. & Lee, S. Y. (2009). Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 36(2), 1466-1477. https://doi.org/10.1016/j.eswa.2007.11.061

[30] Thanh Lam, H. & Calders, T. (2010). Mining top-k frequent items in a data stream with flexible sliding windows. *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 283-292. https://doi.org/10.1145/1835804.1835842

[31] Mahanti, A., Eager, D., & Williamson, C. (2000). Temporal locality and its impact on Web proxy cache performance. *Performance Evaluation*, 42(2), 187-203. https://doi.org/10.1016/S0166-5316(00)00032-8

[32] Giannella, C., Han, J., Pei, J., et al. (2003). Mining frequent patterns in data streams at multiple time granularities. *Next generation data mining*, 212, 191-212.

[33] Chang, J. H. & Lee, W. S. (2003). Finding recent frequent itemsets adaptively over online data streams. *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 487-492. https://doi.org/10.1145/880530.880598

[34] Patil, J. B. & Pawar, B. V. (2007). Trace driven simulation of GDSF# and existing caching algorithms for web proxy servers. *Proceedings of the 6th WSEAS International Conference on Data Networks, Communications and Computers (DCOCO 2007), Trinidad and Tobago*. 378-384.

[35] Romano, S. & ElAarag, H. (2012). A quantitative study of Web cache replacement strategies using simulation. *Simulation*, 88(5), 507-541. https://doi.org/10.1007/s11197-014-4081-x

Contact information:

Jianhai DU, PhD Student
School of Computer Science and Technology,
Beijing University of Aeronautics and Astronautics
Beijing, 100191, China
E-mail: dujianhai6@163.com

Shiwei GAO, Engineer
Beijing Institute of Aerospace Control Devices
Beijing, 100039, China
E-mail: ge89@163.com

Jianghua LV, Associate Professor
(Beijing University of Aeronautics and Astronautics,
Beijing, 100191, China
E-mail: jianghuavpaper@126.com

Qiangqian LI, PhD Student
Beijing University of Aeronautics and Astronautics,
Beijing, 100191, China
E-mail: paper135789@163.com

Shilong MA, Professor
School of Computer Science and Technology,
Beijing University of Aeronautics and Astronautics,
Beijing, 100191, China
E-mail: shilongma8@163.com