Research Article

DSMC: A Novel Distributed Store-Retrieve Approach of Internet Data Using MapReduce Model and Community Detection in Big Data

Xu Xu,1 Jia Zhao,2 Gaochao Xu,1 Yan Ding,1 and Yunmeng Dong1

1 College of Computer Science and Technology, Jilin University, Changchun, Jilin 130000, China
2 College of Computer Science and Engineering, ChangChun University of Technology, Changchun, Jilin 130000, China

Correspondence should be addressed to Gaochao Xu; xugc@jlu.edu.cn

Received 21 July 2014; Accepted 21 August 2014; Published 16 November 2014

Academic Editor: Gongfa Li

Copyright © 2014 Xu Xu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The processing of big data is a hotspot in the scientific research. Data on the Internet is very large and also very important for the scientific researchers, so the capture and store of Internet data is a priority among priorities. The traditional single-host web spider and data store approaches have some problems such as low efficiency and large memory requirement, so this paper proposes a big data store-retrieve approach DSMC (distributed store-retrieve approach using MapReduce model and community detection) based on distributed processing. Firstly, the distributed capture method using MapReduce to deduplicate big data is presented. Secondly, the storage optimization method is put forward; it uses the hash functions with light-weight characteristics and the community detection to address the storage structure and solve the data retrieval problems. DSMC has achieved the high performance of large web data comparison and storage and gets the efficient data retrieval at the same time. The experimental results show that, in the Cloudsim platform, comparing with the traditional web spider, the proposed DSMC approach shows better efficiency and performance.

1. Introduction

The amount of the global Internet data has an explosive growth currently. At the same time, millions of websites in each region update their data in real time every day. The old data is enormous and new data grows rapidly. This shows the significance of Internet data backup. It is a major challenge about how to capture, store, compare, and retrieve data efficiently and rapidly on the Internet. Traditional web spider is weak to the large number of websites. Using data backup to store the real time data on the Internet is an effective method, but it needs very large storage and computing resource. This situation makes the single-host web spider and traditional data backup methods no longer applicable.

For some functions like data comparison, we only need to know whether the website data changed or not in different time. So it is not necessary to back up the whole information and data comparison can be achieved with very small resource. In this case, we will consider optimizing it from the point of distributed data capture, web data compression, and storage structure division to optimize the existing problems.

This paper presents a distributed data capture strategy in a cloud environment. On the basis of hash functions’ characteristic of the compressing texts corresponding with original data, we use hash functions [1] to compress the web data and ensure the reliability of data. According to the thought of data deduplication with MapReduce model [2–4], we use MapReduce model to cut the repeating data produced by different distributed hosts and to integrate the rest and finally we can get the no-repeat data. We still integrate community detection thought into storage structure division strategy. Community detection puts different data into different communities [5–7] according to certain conditions and stores the data in terms of the existing communities. The thought of community detection orders large amount of disordered data efficiently and solves the classified storage
issues well as well as making it convenient to inquire and compare the data, providing a great convenience to the deeper research of big data on the basis of this structure.

The distributed structure combining the web spider achieves a large amount of network data’s fast fetching. It changes from traditional single-host web spider data capture to multihost distributed processing in cloud data centers. Using the method which makes multihost work together with the same algorithm at the same time, compared with traditional web spider, it can avoid the problems of the low capture efficiency, long task cycle, and less fetching in a specific period of time. Due to lightweight hash functions which can convert a large number of texts into short hash values, we can achieve the compression of large amount of data to address the problem of the heavy storage burden caused by a full backup. And also due to the security property of hash functions, we can make one to one correspondence between texts and their hash values. Compared with the traditional way of storing the texts directly, the proposed approach can solve the problems of insufficient storage space, storage pressure of rapid growth, and high storage cost and the low ratio of the data needs to all the data. Using the MapReduce model, we collect and process the results from different distributed virtual machines with tasks [8]. And then the collected data is transformed to each network data entry. Each entry contains all the hash values of its website. Taking the entries as parameters and cutting the repeating entries until only left one can ensure that there is no repeat in the final data. It can save the storage space and avoid the repeating data while querying.

This paper adopts the local module method of community detection to realize the storage structure of large amount of integrated data. The network diagram is built on the basis of the data entries after processing and then the local modularity and the comprehensive characteristic value are calculated according to the network diagram. Then we use the local module method to put data in the same community and cycle to the same storage area. It not only makes it convenient for users to retrieve the data, but also decreases the time complexity in query, the redundant operations, and the heavy system burden.

The rest of this paper is as follows. In the second section, we give the related works and the reasonable assumptions. The third section contains the formalization of the problems, the algorithm, and the implementation process. In the fourth section, the experimental results and analysis are presented. We will summarize the whole paper and put forward the future work in the fifth part.

2. Related Work

Data partitioning and storage is always a popular research topic. Its targets are to ensure that all data could be classified and integrated with certain rules, to search and sort the stored data fast, and to do other researches which are relevant to the data division. The question raised is how to find the best divided communities under one goal according to the content of data and the tightness of link under the condition of consolidated data. The collections of these Internet data form the network communities. Researchers have put forward a series of community detection strategies [9–12] which can meet various needs. In addition, there are still some researches using the mathematic models [13, 14] to satisfy specific features and varieties of the communities. As a matter of fact, most of the community detection strategies only sort the data according to the requirements.

The existing basic method of network community detection strategies is clustering, which is a procedure dividing the collections of abstract objects to lots of classes made of similar objects. Divisive method and agglomerative method [15–18] are classical algorithms of community detection. The commonly used algorithms are GN (Girvan and Newman) algorithm and Newman fast algorithm. GN algorithm is accurate, but it has a high time complexity and does not suit the condition of a network with a large number of network nodes and data. Small-scale network data is suitable for GN algorithm. In the meantime, GN algorithm still has a problem that you may not locate the community boundaries accurately when you partition the communities. There are still many researchers doing the study of optimizing GN at the moment. Yang et al. [19] adopt an optimized algorithm based on GN which has the concept of integration and the concept of closed changes of clustering coefficients. The edge with the lowest weight in the network diagram is always transferring iteratively until this network diagram degenerates into an isolated node. The advantages of GN algorithm are its simplicity, feasibility, high quality, and high accuracy, but when faced to millions of network nodes and large-scale data with complex network structure, the efficiency of community detection is not ideal.

Newman fast algorithm is the first classical agglomerative method of community detection. It is mainly used in the community detection of large-scale network with complex structure. It can divide the communities of large-scale network rapidly. It is a kind of agglomerative method based on the idea of greedy algorithm. The total time complexity of this algorithm is $O(m(m + n))$. A tree diagram of the detected communities can be obtained after finishing the whole algorithm and then choose to disconnect in different place to get different network community structures. There is a problem that the right community structure is not got when the number of the communities is known in some specific applications. And the agglomerative method tends to find out the core of the communities and ignore the periphery. The core of the community is in closed touch with its surrounding nodes in most cases, so it is easy for us to find out. And the periphery is hard to divide because of the less connection with other nodes. In literature [20], Li and Song present ECGA (Extended Compact Genetic Algorithm) and it uses the coupled variables of probability distribution to optimize the algorithm. In [21], Sun and Yang put forward a divisive method based on the basic principle of agglomerative method and iterative edges. Although the agglomerative method can solve the community detection problem of large-scale network, the results tend to be one-sided optimization because the above algorithms could not satisfy the fast community division, the most suitable community scale, and...
the best community structure at the same time. And in some situations like that if one node has an edge with just one community, it is hard to distribute the node correctly.

Local module method [22–24] is an efficient strategy of community detection, and also a way to divide the communities available and to fix the above disadvantages. It can correct the scale of the network communities according to the maximum value and minimum value. It can detect the clear structure of a given network. In the process of the detection, it will use the two values to calculate the degree of differentiation of the community level to judge if the detection has arrived at the border. It will clip the network and do multiple detections to find out the whole network level and structure. The distributed data capture strategy put forward by this paper is more efficient to traditional methods meanwhile and makes up for the little deficiency on the cycle length. In [25], Gong et al. present a way to detect the communities based on density. The calculation of cluster density is similar to the local module method [26–29]. It modifies the parameters of the communities upon one or several values of dynamic change. The parameters have a certain degree of affect to the structures, scales, borders, and the contact of the content of the communities. Considering these factors, we will relatively complete community structure which is in accordance with our need.

2.1. The Raise of the Problem. While using the web spider to capture data, we want to get lots of data from different areas in the world. Generally, we need to capture them through different source URLs of the areas. When we just have one host, we can only capture data from the different areas in sequence. This undoubtedly makes the capture time of all the areas extremely long. At the same time, we need to put a large amount of network data into the storage space and the overhead of the storage is too much. When a lot of users want to search the same entry, redundant operations are more and the time complexity is very high. Apparently, it is necessary to achieve a data-storing and data-retrieving approach which have capabilities in improving the efficiency of data capture, saving the storage space, and dividing the storage structure.

2.2. The Formalization of the Problem. The proposed DSMC approach using distributed thought on the problem of single-host web data capture is applying for many virtual machines in cloud data center and doing the distributed data capture [30, 31]. Supposing in the same network environment there are \( n \) virtual machines in cloud data center, these virtual machines are parallel and equal in priority. Distribute different source URLs of different areas to each virtual machine and let the hosts run the web spider method, respectively, according to their own URLs of different areas at the same time. Define a triple to express the problem scenario of the distributed capture strategy: \( Y = (\text{VM}, U, R) \). VM represents that in cloud data center there is a group of virtual machines with capture mission \( \text{VM} = \{\text{VM}_1, \text{VM}_2, \text{VM}_3, \ldots, \text{VM}_n\} \). \( U \) represents a group of source URLs of different areas \( U = \{U_1, U_2, U_3, \ldots, U_n\} \). \( R \) represents a gather of the processed results of the virtual machines after capturing \( R = \{R_1, R_2, R_3, \ldots, R_n\} \). The condition that the number of the elements in VM, \( U \), and \( R \) is the same is met. Our strategy distributes a nonredundant element in \( U \) to each of the elements in VM. Then the system processes the obtained results to generate the elements in \( R \). Each element in \( R \) is \( R_k = \{R_{k1}, R_{k2}, R_{k3}, \ldots, R_{km}\} \). The number of the elements in \( R_k \) is the number of the web pages captured by virtual machine \( \text{VM}_k \).

It is found that it is not necessary to back up the whole data on the web page completely for monitoring the websites, obtaining the evidence, or comparing the contents. So DSMC brings in hash functions to save the storage space. Hash functions are used to compress data from the web pages. For all the virtual machines, they capture data in the same way. We take one, for example. Supposing that this host will get some pages in a cycle, when it is capturing the current page, we adopt an appropriate hash function to compress its URL and texts into hash values and then store them. The reliability of the mapping can be ensured between hash values and their corresponding texts because of the security of the hash functions. When the texts change, their corresponding hash values are different from the previous values. The nature of brief hash values also helps us save the space. Define a triple to express this situation: \( X = \{H, D, HD\} \). \( H \) represents an appropriate hash function: \( H = \text{hash}() \). \( D \) represents all the URLs and data that one host has captured: \( D = \{d_1, d_2, d_3, \ldots, d_k\} \), where \( d_k = \{d_k·\text{-url}, d_k·\text{-data}\} \). \( HD \) represents the gather of the hash values of the URLs and data: \( HD = \{hd_1, hd_2, hd_3, \ldots, hd_i, \ldots, hd_j\} \), where \( hd_k = \{hd_k·\text{-url}, hd_k·\text{-data}\} \). The number of the elements in \( D \) is the same as \( HD \). Using the formula \( H = \text{hash}(D) \) to transform the elements in \( D, HD \) can be obtained. Each host with a capture task can get a HD set and all the HD sets can constitute the set \( R \) mentioned above.

MapReduce model is utilized to pick out the repeating data between the data sets of different hosts and the different data are integrated together to get the final data set. In the final data set, each entry of data can be denoted as \([\text{Num}, \text{hd-url}, \text{hd-data}, \text{connectNum}]\), \( \text{Num} \) is the mark number of a web page, \( \text{hash-url} \) is the hash value of a URL, \( \text{hash-data} \) is the hash value of a piece of content, and \( \text{connectNum} \) is the set of the mark numbers of the web pages the current page is linking to. If we get URL2 through URL1 while running the capture strategy, there is a relationship between URL1 and URL2, or URL1 links to URL2.

As for the storage and the users’ retrieval, the local module method is employed to detect and divide the network communities and the data is stored according to the communities and the cycle of capture. The advantages of using the local module method are that it not only puts the relevant pages into the same community to get a clear structure, but also reduces the searching time for users’ convenience.

For the obtained data of all the areas in one cycle, we build a network diagram \( G = (V, E) \). The pages represent the nodes and the links between the pages represent the edges.

Explanation of parameters:

- \( V \): the set of nodes,
- \( E \): the set of edges,
$D_i$: the degree of node $V_i$,
$p_i$: the number of the neighbor nodes of $V_i$,
$e_i$: the number of the edges between $p_i$ nodes,
$C_i$: clustering coefficient:

$$C_i = 2 \cdot \frac{e_i}{p_i} \cdot (p_i - 1),$$  \hspace{1cm} (1)

$CF_i$: the comprehensive eigenvalue of nodes:

$$CF_i = \alpha \cdot C_i + (1 - \alpha) \cdot \frac{D_i}{N},$$  \hspace{1cm} (2)

$\alpha$: an adjustable parameter, $0 < \alpha < 1$,
$N$: the number of the nodes in the entire network,
$Q$: community local modularity $Q = \frac{L_{in}}{L_{in} + L_{out}}$,
$L_{in}$: the number of the edges we got inside the community when the current node to be distributed belongs to this community,
$L_{out}$: the number of the edges we got between the nodes inside the community and the nodes outside the community when the current node to be distributed belongs to this community.

The basic thought of the local module method is regarding the node with the biggest comprehensive eigenvalue as the initial node and then looking for the candidate node which can maximize the local modularity $Q$ from the candidate set. Then put the node above into the community until the candidate set does not grow and the community finally determined. After that, look for the node with the biggest comprehensive eigenvalue as the initial node of the next community from the nodes which never belong to any community. Repeat the above steps until every node has a community for it. Now, we consider the community structure as the real network community structure.

2.3. Framework Design of the System. Figure 1 describes the distributed framework in cloud data center. It shows the whole distributed capture module, data processing module, and storage module clearly. First of all, every distributed host gets the source URLs of different areas and executes the web spider method with them. A large amount of data is transformed to the hash values through the same hash function. Next, the system puts all the hash values into the resource pool [32] and does the community detection through the local module method. Finally, the data is stored in different region according to the time periods and the divided communities.

2.4. The Algorithm of DSMC. The proposed DSMC algorithm is a heuristic data capture and storage strategy which is used to classify the big data collected by distributed host and stores them in the cloud data center. At first, DSMC makes up a distributed host group by $n$ hosts which can capture data and capture the data in a distributed way through the given source URLs. In the second place, it compresses the texts captured by the hosts with a specific hash function in each host. The large amount of data is transformed into brief hash values. Then the hosts send the hash values to the resource pool in cloud center. In the resource pool, DSMC utilizes MapReduce model to do the data deduplication. Thirdly, the network diagram is built according to the links between the network nodes. Each node’s local modularity and other threshold values are calculated from the diagram and the network communities are divided on the basis of the calculated results. When the division is stable and does not change anymore, it will be our foundation of the coming storage. Finally, the data is stored in different regions according to the divided communities and the cycle in which the data was captured. The procedure of detecting the network communities is the procedure of looking for the best storage structure.

2.5. The Implementation Process of the Algorithm. The specific implementation process of DSMC is as follows.

Step 1. To achieve the best effect and avoid that a single-host captures a large amount of Internet data too slow and collects a period of data for too long, multiple hosts are needed to run the web spider and capture data at the same time. So we need to apply the distributed and parallel processing in cloud data center. Assume that there are $n$ hosts waiting for tasks in cloud data center, to distribute a source URL to each host.

To get more comprehensive and wider data, the principle to choose source URLs is choosing several URLs of each area and determining one of them in common use. The other
Step 2. Begin to do the data capture with the chosen source URLs. The hosts in cloud data center start to capture data with the same capture strategy through their own URLs at the same moment. Predetermine the data scale we need or other conditions. If a host satisfies the conditions, it can stop running. Transform the data and URL in each page to hash values with the same hash function in the capture procedure. Each page will have two hash values. Each host’s hash-data will be sent to the resource pool in cloud data center. If every host has finished the above process, the hosts will be ready for the data capture of next period. The data of this period will be processed in the resource pool.

Step 3. After putting the data in the resource pool in Step 2, two hash values of each web page make up an entry. The two values are the text value and the URL value. Although different areas have a low intersection of the grabbed data, there are still some pages captured more than once in different regions. Delete the repeating data in the resource pool using MapReduce model before storing. We make the entries be the parameters of MapReduce model. When two entries were found with the same two values, one of them will be deleted. And finally, we will get the data set of one period with no duplicated data.

Step 4. Build the network diagram $G(V, E)$ according to the links between the entries. Use local module method to do the community detection referring to the diagram. Calculate the comprehensive eigenvalues of every node using formulas (1) and (2). Make the node with the largest comprehensive eigenvalue to be the initial node. Then find the candidate node which could get the maximum local modularity $Q$ from the candidate set. Put the node into the community and refresh the candidate set until $Q$ does not increase. Now one community shapes up. Find the node with the largest comprehensive eigenvalue and without any communities to be the next initial node. Repeat the above acts until every node belongs to a community. The community structure at this time will be the real network community structure.

The steps of community detection procedure (named Local Module Method($G$)) are shown below.

1. Initialize: read in the network diagram. Keep a linear dynamic chain table for each node in the network. Each node $V_j$ can be denoted as $V_j = \{N_j, D_j, C_j, CF_j, CM_j\}$; here $N_j$ is the node number, $D_j$ is the node degree, $C_j$ is the node clustering coefficient, $CF_j$ is the node comprehensive eigenvalue, and $CM_j$ is the mark number of the community the node belongs to. Calculate the comprehensive eigenvalues of the nodes using formulas (1) and (2). Map the information of all the nodes to the chain table. Set the mark number of community of each node to 0. Set the community set $CS = \{\}$. (2) Choose the initial node: set a community $W$ with no node, $W = \{\}$. Sort the nodes according to the comprehensive eigenvalues and form a queue in descending order. Choose a node $V_j$ with the biggest eigenvalue and whose mark number is 0 in sequence to be the initial node, and add the node into $W$. If there is no node whose mark number is 0, execute (8). Set the mark number of the initial node equal to the community number of $W$. It represents that the node belongs to community $W$. Set the initial local modularity $Q_c$ of community $W$ to 0.

3. Determine the candidate set: put the nodes which are linked to the nodes in community $W$ and whose mark number is 0 into set $B$.

4. Form the structure of the communities: for each node $V_k$ in set $B$, suppose that node $V_k$ belongs to community $W$. Use the formula $Q = L_{in}/(L_{in} + L_{out})$ to calculate the modularity $Q_k$. We can get the maximum modularity $Q_k$ by calculating. If $Q_k$ is greater than $Q_c$, update the value of $Q_c$ and put the node $V_k$ that $Q_k$ belongs to into community $W$. In the meantime, change the mark number of node $V_k$ to the community number of $W$. If $Q_c$ is greater than $Q_k$, execute (6). To reduce the complexity of the algorithm, we rule that if $Q_c$ is greater than $Q_k$ for two consecutive times, node $V_k$ will be erased from set $B$.

5. Repeat (3) and (4).

6. Get the community $W$, and put $W$ into set $CS$.

7. Repeat (2), (3), (4), (5), and (6).

8. Get the final community set $CS$.

In this way, we will divide the set of the random data into several data sets successfully.

Step 5. Distribute the entries to different storage areas according to the communities and the periods. The number of the areas is “the number of the communities multiplied by the number of the periods.” And the number will grow up with the increase of the periods. The network nodes in the same community and the same period will be stored in the same area. Define a gather of the storage areas $S(m, t)$. $m$ is the community that the data belong to and $t$ is the period in which the data was captured. The storage area of the data of one web page is controlled by variables $m$ and $t$. There is a matrix below which we can express gather $S(m, t)$:

$$
\begin{pmatrix}
S(t_1, m_1) & S(t_1, m_2) & \cdots & S(t_1, m_n) \\
S(t_2, m_1) & \vdots & & \vdots \\
\vdots & \vdots & \ddots & \vdots \\
S(t_n, m_1) & \cdots & \cdots & S(t_n, m_n)
\end{pmatrix}
$$

We do not need too much storage space because of the hash strategy.

Step 6. Repeat the above steps.
Procedure DSMC(n, Hash)
Input: the number n of virtual machines, hash function Hash()
Output: each period’s data structure FS
(1) Integer t = 1; //the number of current period
(2) Initiate virtual machines set VM = [VM_1, VM_2, VM_3,..., VM_n];
(3) Initiate URLs set U = [U_1, U_2, U_3,..., U_n];
(4) For (TRUE) do
(5) For i = 0 to n - 1 do
(6) Distribute U_i to VM_i;
(7) End for
(8) For each virtual machine VM_i in VM
(9) Run WebSpider;
(10) Get R_i = [R_{i1}, R_{i2}, R_{i3},..., R_{im}]; //R_i is the set of captured data for VM_i,
(11) //m: the number of captured data entries.
(12) For j = 0 to m - 1 do
(13) R_{ij} = Hash(R_i);
(14) End for
(15) Send R_i to Resource Pool;
(16) End for
(17) R = MapReduce(R_1, R_2, R_3,..., R_n);
(18) Build G(V, E); //network diagram for R, V: the set of nodes, E: the set of edges
(19) CS = Local Module Method(G); //
(20) Store R to cloud data center on the basis of CS and t;
(21) Get FS;
(22) t++; //to the next period
(23) End for
(24) End DSMC

Pseudocode 1

The pseudocode of DSMC’s algorithm has been given as in Pseudocode 1.

In microcosm, distributed data capture is also an application process of the web spider. Web spider algorithm is a method for single-host to grab network data. The efficiency and the improvement of web spider are relevant to the data scale we need and the performance of the hosts. Web spider gets the link between the existing web pages and other pages by analyzing the contents of the original pages. Get other contents of the pages through the existing links and repeat the steps until the condition was satisfied.

It is hard to get significant results by optimizing the web spider itself. When the effect of optimization is low and the performance of the machines could not be lifted, it is the best way to consider increasing the number of the active hosts. That is usually called the distributed process. We need to classify the web sites according to the regions they belong to. The web sites in the same region have a tight connection with each other. The web sites in different regions connect loosely and the degree of the captured pages is low. Arrange each host of a source URL and let them run the algorithm at the same time. If the time that one host finishes the entire job in a period is T, assuming that the workload in each region is almost the same, the time for n hosts finishing the entire job in a period is T/n. Thus it can be seen that the efficiency is in direct proportion to the number of the hosts in cloud data center.

Hash functions can be used to save much storage space because hash functions can compress the texts and each section of texts will have only one hash value. The storage space that the hash values occupied is much less than the texts. Supposing that there is a web page text A, so the hash value of A is hash(A). The latter usually uses 8, 16, or 32 bytes just like MD5 and SHA1. In other words, only when A is less than 32 bytes, the hash storage strategy may not save the space. As everyone knows, the data of one page is usually much greater than 32 bytes. That means A > hash(A). Similarly, A_1 + A_2 + A_3 + ⋯ + A_n > hash(A_1) + hash(A_2) + hash(A_3) + ⋯ + hash(A_n).

From macroscopic perspective, it is a procedure of multiple parallel hosts’ working process and reducing the time at the cost of the increase of working space. In one period, we transform the data captured by the hosts to hash values and send a large number of hash values to the resource pool together. We use MapReduce model to do the data deduplication to ensure the uniqueness of the data. The distributed hosts in cloud data center capture the data of next period meanwhile. Comparing with nondistributed methods, it shortens the cycle and increases more valid data in a period of time. And the local module method while storing the data is a procedure of looking for the clusters actually. The clusters have the following characteristics. The clusters themselves are tight and the clusters separate each other as far as possible.

The reason that our local module strategy conforms to the thought of clustering algorithm is using the tightness between the links of the nodes as evaluation parameters. The obtained data could be divided using the local module method because of the links between the web sites. The links between the web
sites are the only direct connections and the connections cannot be segmented. We consider the abstraction of all information from one page as an entry and then abstract each entry to a node. The links between the nodes can make up a network diagram $G = (V, E)$. Use the eigenvalue formula of the local module method to calculate the comprehensive eigenvalue of each node. Choose the node with the maximum eigenvalue to be the initial community and choose the node which can make the local modularity of the community the greatest to join the community from the rest until the modularity cannot grow. We can get one final community like that. Choose the node with the maximum modularity from the nodes without a community and repeat the steps above. There is a sign that no single node represents the end of clustering. Each entry will be in a community at this moment and then store the data on the basis of the communities and periods.

DSMC can get a complete database and a store-retrieve system after multiple task cycles. It is a long-term process and the proposed DSMC algorithm ensures that a good result can be obtained in a reasonable period of time in this process.

3. Evaluation

In this section, we have conducted several experiments to verify the distributed capture, classification, and storage method proposed in this paper. The experiments in this section mainly consist of four parts: (1) verify the efficiency of the data capture by comparing single-host strategy with distributed capture strategy; (2) verify the occupation of the storage space by comparing hash storage strategy with the original text storage strategy; (3) verify the efficiency of data deduplication by comparing the MapReduce model with others; (4) verify the efficiency of the data retrieval by comparing the community detection strategy with other partition strategies. In addition, an extra experiment is made to verify that storing the data according to the task cycle will increase the efficiency of retrieval. To imitate a distributed data capture center composed of 10 hosts, these 10 source URLs of different regions and distribute them to the hosts. Let the distributed hosts use web spider to capture the data periodically and then send them to the resource pool to be processed. After the process, the data will be stored in the database. Furthermore, we set the change rate of the web page contents to once per half an hour.

3.2. Comparison of the Data Capture Efficiency. This experiment is set to verify the high efficiency of the distributed capture strategy against the traditional web spider. In this experiment environment, we combine the distributed method and web spider to achieve the high efficiency data capture. The distributed data capture is compared with traditional web spider in time consumption by capturing the same amount of data. The time consumption expressed the efficiency. As shown in Figure 2, when capturing the same amount of data, traditional web spider will spend more time and the time cost of distributed strategy is relatively less. So the distributed strategy is more efficient than traditional web spider. The reason is that traditional way only considers the optimization on a single-host and the improvement of the single-host algorithm has a poor effect. And the distributed strategy considers multiple hosts working in parallel while optimizing the algorithm. In ideal conditions, each additional host means more speed than single-host. Therefore, distributed strategy will save more time than traditional way so it is more efficient.

Moreover, Figure 3 tells us that traditional web spider has a longer data capture period than distributed data capture strategy. Distributed strategy not only considers the efficiency, but also thinks about shortening the capture period to avoid the loss of data because of the frequent changes of data in a short time that we cannot capture in time.

3.1. Experimental Scenarios. On Cloudsim platform, build a distributed data capture center composed of 10 hosts. These hosts have the same computation efficiency and configurations. Build a resource pool to temporarily store the data and build a database to store the data. The experiments will create 10 source URLs of different regions and distribute them to the hosts. Let the distributed hosts use web spider to capture the data periodically and then send them to the resource pool to be processed. After the process, the data will be stored in the database. Furthermore, we set the change rate of the web page contents to once per half an hour.

![Figure 2: Comparison of data capture time.](image)
strategy realizes the optimized strategies proposed by many works. DSMC can concentrate on their advantages and eliminate the contradictions which are brought by different strategies and make us only choose one of them. At the same time, it avoids the situation that the whole job cannot proceed because of the broken single-host or something else according to the independence of the hosts in the host group. Distributed strategy ensures that the other hosts will finish their work when one of the hosts stops working, so it can avoid the loss of the rest of the data. In conclusion, the proposed DSMC approach is highly efficient in data capture.

3.3. Comparison of the Occupation of Storage Space. This experiment is designed to verify that hash storage uses lower storage space than original text storage in a long period of time and it costs only a little storage space. The experiment designs that two storage centers run for a week. The size of used space in the two centers will be recorded every day. We achieve the comparison of the occupation of storage space between the hash transformation strategy and the original text strategy. In the meanwhile, we use two groups of distributed hosts with the same configuration, the same network situation, and different storage strategies to store data in each storage center periodically. The distributed hosts capture the network data and store it periodically without a break as everyone knows. Therefore, as time goes by, the difference between their occupied spaces will be more and more obvious. As shown in Figure 4, in the two storage centers with different storage strategies, hash strategy needs less storage space than the original text storage strategy and the difference will be clearer with the growth of data. The hash strategy has the characteristic of low cost. In the situation that the original text of one single page is extremely short, data from hash strategy will be a little more than the original texts. In Figure 5, hash strategy needs more storage space than original text storage strategy. It is because the length of the hash values is changeless but the length of the original texts is changeable. When the data length of the original text is shorter than its hash value, the hash value needs more storage space without doubt. However, the superiority of the hash strategy cannot be denied because the original text has larger data size in most cases. The space that hash strategy saved is further larger than the space which is used more in rare cases. And when the original text is less, the biggest difference between them is the length of the hash value. In other words, though there is a situation that the original text is less, hash
strategy saves more space than the original text strategy in general. To sum up, the hash storage strategy presented in this paper is an efficient and feasible strategy to save storage space.

### 3.4. Comparison of the Efficiency of Data Deduplication

In this experiment, we use the resource pool and the distributed hosts to do the data deduplication. These repeating data may cause the increase of the workload, the storage space, and the time complexity of retrieval and then it will encumber the whole strategy. It can be seen from Figure 6 that the MapReduce strategy is compared with the host-process strategy in the resource pool. In the simulative resource pool, we use MapReduce model to reduce the repeating data in one period and the host-process strategy processes the data before sending the data to the resource pool. Figure 6 tells us that the MapReduce strategy processes the data in a period faster and it can ensure that there will be no repeating data after the process. The host-process strategy needs more time while processing the same amount of data, so its efficiency is lower. And after detection, the host-process strategy remains a little repeating data. This is because each host's data area will have intersections with other hosts. The host itself can process the repeating data in its own data area, but it cannot process the data in the intersections if two hosts have repeating data there. That makes the repeating data still exist after sending the processed data into the resource pool. In Figure 7, all the data is sent to the resource pool first. Then we compare the MapReduce strategy with all file hash methods running in the resource pool. It can be seen in Figure 7 that MapReduce strategy spends less time than all file hash methods in the resource pool and the difference is more and more apparent while data is increasing. MapReduce strategy's main objective is to save time while doing data deduplication, so when faced with a large amount of data it can satisfy the requirement of high efficiency. Therefore, the MapReduce strategy is more reasonable and efficient. In this paper, we have always been focusing on the success of the data deduplication, so it is also an important target to ensure that there is no repeating data after process.

### 3.5. Comparison of the Retrieval Efficiency

In this experiment, the proposed DSMC approach is compared with the storage without strategies in the retrieval efficiency of stored data. The proposed DSMC strategy and no-strategy storage are executed. As shown in Figure 8, the retrieval efficiency of DSMC is higher than the other with the growth of the queries. This is because the queries of specific data can be
led to specific communities after dividing. The users have to traverse the database if they want to find the data without an appropriate community. This increases the retrieval time and makes the efficiency low. It is well known that when the data in database is larger, the burden of the retrieval time is heavier. Thus, aiming to reduce the retrieval burden, it is important to optimize the storage structure. The community detection storage strategy not only improves the efficiency of the retrieval, but also divides the network data according to the links and contents.

The distributed hosts store all the data they captured in storage center periodically, so we must know the period that the data you want belongs to if we want to get it. When you determine the period, you can find the storage area where the data of that period is in and then find the data you want. The length $T$ of a period is certain, so we can calculate the period numbers which a time point belongs to. The major point is the storage area of periods. Figure 9 reflects the comparison of the query time between periodic storage and nonperiodic storage with the same community detection strategy. While choosing data at random time points in each period, the periodic storage is more efficient because all data in one period is in the same storage area. While choosing many time points in one period, no matter how many time points, they all lead to the same storage area—the periodic storage is also more efficient. The community detection can only determine lengthways storage areas; only the periodic storage can determine the transverse areas to locate the data. We can make a completer method according to the above experiment. A mixed storage strategy of periodic storage and community detection storage is a good way. Store the data of each period periodically and detect the communities in each period at the same time. There is an accurate location in space and in time for each piece of data. It will guarantee that we can find the data we need accurately. It can be seen that this strategy concentrates on the advantages of community detection and periodic storage. It improves the efficiency greatly and strengthens the division of the data for researches in many respects. In a word, community detection storage strategy is a high-efficient storage and retrieval strategy and it is relatively visual.

4. Conclusion and Future Work

This paper puts forward a network data capture and storage approach DSMC based on distributed process and community detection and gives the main idea, implementation process, and evaluation of it after summarizing the related work. It adopts the heuristic idea based on distributed process and community detection. In DSMC, the data regions are divided and the number of the distributed hosts and the source URLs of the regions are determined at first. Then DSMC distributes different source URLs to the hosts to run the algorithm for higher efficiency and executive capacity in data capture. The idea of community detection is integrated into the network data storage. The community detection strategy is utilized to divide the communities that the network nodes belong to and the data is saved according to the divided communities. It will guarantee that the data of the same class will be stored in the same storage area and realize the function of increase of the data retrieval efficiency. Moreover, we have integrated the thought of local module method into community detection and formalized the network data and all the links between them. Using the calculated local modularity, the degree of the network nodes and the community detection strategy based on local module method will help us get the communities which are appropriate for the whole network accurately and rapidly. It makes the community detection restrain itself faster and shows the foundation of mathematics of the community detection method put forward.

To evaluate the performance and efficiency of DSMC, we have designed and conducted many experiments on Cloudsim platform. Firstly, in the experiments of data capture and data deduplication, distributed capture and MapReduce model spend less time under the same conditions. Secondly, in the experiment of storage space, hash strategy occupies less storage space than original text strategy. Thirdly, in the experiments of community detection and data retrieval, the result shows that community detection storage strategy is more efficient while retrieving data. At last, we compared periodic storage with nonperiodic storage. Its result shows that the combination of community detection and periodic storage will reduce the demand of retrieval time and it is a more appropriate method. The four groups of experiments demonstrate that the proposed DSMC approach is more efficient and can save more space while achieving the high-efficient data capture and data processing.

DSMC reduces the time cost of capturing a large amount of data and improves the frequency of data capture in a certain extent. In the distributed and community methods, there are several open problems. The number $n$ of the distributed hosts and the memory requested by the hosts are
experience problems. More experiments are needed to test the occupation of time and space to make the strategies more efficient and feasible. In this paper, all parameters are set to appropriate values.

To further improve the performance and efficiency of DSMC, we decide to research the superposition of the divided communities. Community detection method should have the ability to find the elements which can also belong to other communities. So we can locate the elements accurately. It is another break for DSMC to address the problem of MapReduce model’s bandwidth bottleneck.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This research is supported by the National Science-Technology Support Project (2014BAH02F02).

References

[1] C. Estébanez, Y. Saez, G. Recio, and P. Isasi, “Performance of the most common non-cryptographic hash functions,” Software: Practice and Experience, vol. 44, no. 6, pp. 681–698, 2014.
[2] I. Kitsos, K. Magoutis, and Y. Tzitzikas, “Scalable entity-based summarization of web search results using MapReduce,” Distributed and Parallel Databases, pp. 1–42, 2013.
[3] D. Lee, J.-S. Kim, and S. Maeng, “Large-scale incremental processing with MapReduce,” Future Generation Computer Systems, vol. 36, pp. 66–79, 2014.
[4] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” Communications of the ACM, vol. 51, no. 1, pp. 107–113, 2008.
[5] M. Girvan and M. E. Newman, “Community structure in social and biological networks,” Proceedings of the National Academy of Sciences of the United States of America, vol. 99, no. 12, pp. 7821–7826, 2002.
[6] D. Jin, B. Yang, C. Baquero, D. Liu, D. He, and J. Liu, “A Markov random walk under constraint for discovering overlapping communities in complex networks,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2011, Article ID P05031, 2011.
[7] D. Jin, D. He, Q. Hu, C. Baquero, and B. Yang, “Extending a configuration model to find communities in complex networks,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2013, no. 9, Article ID P09013, 2013.
[8] J. Zhao, Y. Ding, G. Xu, L. Hu, Y. Dong, and X. Fu, “A location selection policy of live virtual machine migration for power saving and load balancing,” The Scientific World Journal, vol. 2013, Article ID 492615, 16 pages, 2013.
[9] Y. Liu, J. Moser, and S. Aviyente, “Network community structure detection for directional neural networks inferred from multichannel multisubject EEG data,” IEEE Transactions on Biomedical Engineering, vol. 61, no. 7, pp. 1919–1930, 2014.
[10] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2008, no. 10, Article ID P10008, 2008.
[11] M. E. Newman, “The structure and function of complex networks,” SIAM Review, vol. 45, no. 2, pp. 167–256, 2003.
[12] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” Journal of Statistical Mechanics: Theory and Experiment, vol. 2008, Article ID P10008, 2008.
[13] Z. Wang and J. Lu, “A mathematical model for calculating urban heat island intensity using mobile survey data,” in Proceedings of the 8th International Symposium on Heating, Ventilation and Air Conditioning, pp. 563–572, Springer, Berlin, Germany, 2014.
[14] L. S. Bocci and V. Müller, “Mathematical model of three-phase induction machine connected to advanced inverter for traction system for electric trolley,” Anais da Academia Brasileira de Ciências, vol. 85, no. 2, pp. 849–858, 2013.
[15] C. Pizzuti, D. Talia, and G. Vona, “A divisive initialisation method for clustering algorithms,” in Principles of Data Mining and Knowledge Discovery, pp. 484–491, Springer, Berlin, Germany, 1999.
[16] J. Y. Pirçon and J. P. Rasson, “The last step of a new divisive monothetic clustering method: the gluing-back criterion,” in Classification, Clustering, and Data Mining Applications, Studies in Classification, Data Analysis, and Knowledge Organisation, pp. 43–51, Springer, Berlin, Germany, 2004.
[17] J. W. Liu, S. J. Yu, and J. J. Le, “Dynamic mining hierarchical topic from web news stream data using divisive-agglomerative clustering method,” in Advances in Knowledge Discovery and Data Mining, pp. 826–831, Springer, Berlin, Germany, 2005.
[18] M. Chavent, Y. Lechevallier, and O. Briant, “DHVCLUS-T: a monothetic divisive hierarchical clustering method,” Computational Statistics and Data Analysis, vol. 52, no. 2, pp. 687–701, 2007.
[19] Y. Yang, P. G. Sun, X. Hu, and Z. J. Li, “Closed walks for community detection,” Physica A: Statistical Mechanics and its Applications, vol. 397, pp. 129–143, 2014.
[20] J. Li and Y. Song, “Community detection in complex networks using extended compact genetic algorithm,” Soft Computing, vol. 17, no. 6, pp. 925–937, 2013.
[21] P. G. Sun and Y. Yang, “Methods to find community based on edge centrality,” Physica A: Statistical Mechanics and its Applications, vol. 392, no. 9, pp. 1977–1988, 2013.
[22] L. Chen, Q. Yu, and B. Chen, “Anti-modularity and anti-community detecting in complex networks,” Information Sciences, vol. 275, pp. 293–313, 2014.
[23] L. Waltman and N. J. Van Eck, “A smart local moving algorithm for large-scale modularity-based community detection,” The European Physical Journal B, vol. 86, no. 11, article 471, 2013.
[24] M. E. J. Newman, “Modularity and community structure in networks,” Proceedings of the National Academy of Sciences of the United States of America, vol. 103, no. 23, pp. 8577–8582, 2006.
[25] M. Gong, J. Liu, L. Ma, Q. Cai, and L. Jiao, “Novel heuristic density-based method for community detection in networks,” Physica A: Statistical Mechanics and Its Applications, vol. 403, pp. 71–84, 2014.
[26] D. Aloise, S. Caferi, G. Caporossi, P. Hansen, S. Perron, and L. Liberti, “Column generation algorithms for exact modularity maximization in networks,” Physical Review E: Statistical, Nonlinear, and Soft Matter Physics, vol. 82, no. 4, Article ID 046112, 2010.
[27] S. Cafieri, P. Hansen, and L. Liberti, “Locally optimal heuristic for modularity maximization of networks,” Physical Review E, vol. 83, no. 5, Article ID 056105, 2011.

[28] D. Aloise, G. Caporossi, P. Hansen et al., “Modularity maximization in networks by variable neighborhood search,” Graph Partitioning and Graph Clustering, vol. 588, p. 113, 2012.

[29] J. Lee and S. P. Gross, “Modularity optimization by conformational space annealing,” Physical Review E, vol. 85, no. 5, Article ID 056702, 2012.

[30] G. Xu, Y. Ding, J. Zhao, L. Hu, and X. Fu, “A novel artificial bee colony approach of live virtual machine migration policy using bayes theorem,” The Scientific World Journal, vol. 2013, Article ID 369209, 13 pages, 2013.

[31] L. Hu, J. Zhao, G. Xu, Y. Ding, and J. Chu, “HMDC: live virtual machine migration based on hybrid memory copy and delta compression,” Applied Mathematics and Information Sciences, vol. 7, no. 2, pp. 639–646, 2013.

[32] M. Xu, H. Yu, and J. Shen, “New approach to eliminate structural redundancy in case resource pools using α mutual information,” Journal of Systems Engineering and Electronics, vol. 24, no. 4, pp. 625–633, 2013.

[33] T. Goyal, A. Singh, and A. Agrawa, “Cloudsim: simulator for cloud computing infrastructure and modeling,” in Proceedings of the International Conference on Modelling Optimization and Computing, pp. 3566–3572, April 2012.

[34] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. de Rose, and R. Buyya, “CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms,” Software: Practice and Experience, vol. 41, no. 1, pp. 23–50, 2011.

[35] M. Armbrust, A. Fox, R. Griffith et al., “A view of cloud computing,” Communications of the ACM, vol. 53, no. 4, pp. 50–58, 2010.
