Classification and Processing of Big Data in Sensor Network based on Suffix Tree Clustering

https://doi.org/10.3991/ijoe.v15i01.9785

Jun Tian(✉), Lirong Huang
Foshan Polytechnic, Foshan City, Guangdong, China
juntianeu18293@163.com

Abstract—Aiming at the perception data acquired by the widely used, fast-developing but still not perfect wireless sensor network system, a relatively complete and universal system for the collection, transmission, storage and cluster analysis of perception data is designed. Perception data is spliced and compressed at the node and reconstructed at the base station, the problem of the acquisition of perception data and energy consumption of transmission is optimized, the distributed storage system is established, and the data reading mechanism and data storage architecture are designed accordingly. The data acquisition protocol and the traditional protocol, the storage system itself and the Oracle database system, and Standard Deviation and Eigensystem Realization Algorithm are respectively adopted for comparison test. Based on Standard Deviation algorithm, the operation of suffix tree clustering is carried out, and the general steps of suffix tree clustering are studied and the structure of perception data and the characteristics of storage are adapted, and the data classification operation based on suffix tree clustering is completed. The results show that proposed Standard Deviationalgorithm algorithm not only inherits the efficiency of the classical algorithm for processing big data, but also has obvious effect on large-scale discrete data processing, and the efficiency is obviously improved compared with the traditional method.

Keywords—Sensor network, big data, storage system, suffix tree, clustering

1 Introduction

With the development of the Internet, big data technology has gradually replaced traditional network technology and has become the trend and mainstream of the current Internet development. Its variety and large storage space make the storage and utilization of big data bring great challenges to existing storage technologies. The sensory data of wireless sensor networks is an important part of current big data. It plays an important role in people's daily life, scientific research, business data analysis, service industry and even military field. However, the processing level of this part of big data is still unable to meet the requirements of relevant industries and fields, and academic and industrial circles have conducted extensive and in-depth research on it.

Sensor network's perceived data has become an important part of big data family due to its large number and considerable increment. However, unlike most existing types of big data, the amount of single acquisition is small and frequently acquired during the process of acquiring data. The characteristics of large-area distribution of sensing nodes make the sensing data itself have its unique characteristics of high dispersion, frequent reading, high storage frequency and complex data. Frequent transmissions will inevitably result in enormous precious energy costs, and delayed transmission will lose the timeliness necessary in areas such as military. Therefore, the problem of real-time data transmission due to the limited energy carrying capacity of the node itself has always been the research core of sensory data acquisition [1]. Generally, a single machine for big data design (such as a database server assumed by a traditional database management system such as Oracle, MySQL, DB2, etc.) or a distributed storage system (such as a distributed database server group such as HBase or Hadoop) only consider the optimization of large data storage and massive data read and write aging, and do not pay attention to the unique data category of perceived data. Storing sensory data on these systems imposes significant limitations on the use, read, write, and categorization of data. Therefore, academics and industry are studying the energy-saving issues of real-time data transmission. It is also working on a big data storage system for storing discrete content such as perceptual data with the above characteristics.

2 Literature Review

At present, the sensory data acquired by the sensor network is first concentrated on the nodes and transmitted to the base station, and then the data is transmitted to the storage system through the network and utilized. However, most of the current research focuses on the assurance of overall architectural design and privacy protection. Glöckner et al. (2017) introduced the concept of the latest scale-free network into the traditional architecture of mature wireless sensor networks in the research and architecture research of wireless sensor network overall protocol, which brings good scalability to the network and great flexibility to the network scale [2].

In another direction, research on privacy protection has received increasing attention. For example, a number of latest achievements in data encryption and privacy protection for sensor network data transmission continue to emerge. However, for an important aspect of data acquisition, the data transmission from the sensing node to the base station and the data compression in the sensing node have been rarely achieved in recent years, so that in a very good network architecture, the initial processing and transmission overhead of the data is large, resulting in limited node life, low data transmission efficiency, and can't effectively adapt to the need of high-speed wireless sensor networks for frequent data transmission. The energy-saving methods proposed by Qin et al. (2016) sacrifice timeliness. When the amount of data reaches a certain scale (the value of the disk size occupied by a certain data), it is compressed according to the traditional file system data compression method, the data is then transmitted. This method of data compression is obviously not applicable in areas with high timeliness requirements such as military and meteorological [3].

Abdullah et al. (2018) believe that in terms of data storage, the sensory data should be updated at any time and stored comprehensively. Large-scale data sets using traditional relational database systems such as Oracle MySQL or other stand-alone storage systems obviously cannot meet the storage requirements of large-scale data such as perceptual data. Even the latest architectures such as distributed storage structures and disk array systems recently proposed by traditional database vendors, due to historical reasons, for the data read/write or update operation, a process is established for each transaction, in high concurrency, it is easy to cause system resources to run out, causing the system to crash or even crash, which will seriously affect other service programs deployed on the same physical machine, resulting database systems incapable of being well-suited for reading and writing data such as perceptual data that may have extremely high concurrent read and write operations [4].

Makhoul et al. (2015) believe that due to the huge amount of data and rapid growth, the traditional IOE combination widely used by the industry (ie IBM minicomputer, Oracle database and EMC high-end storage device combination) will no longer be suitable for a wide range of production environments because of its high cost. With the continuous release of distributed storage systems such as GFS and HDFS, full storage and cost savings of big data are realized. Its distributed architecture has good scalability and can effectively adapt to large-scale data storage. However, perceptual data is a special type of large-scale data with high dispersion, high processing difficulty, frequent reading and writing, and complex categories. It has high value, high effective information content, needs to be stored in a relational database and has requirements for backup synchronization consistency, so it can’t be stored in the NoSQL type database used by traditional big data [5].

The distributed system accurate repair method based on boundary part representation designed by Wang (2018) et al. is aimed at the obstacles that the storage system itself may encounter, and it has certain effects on server failure caused by frequent storage of discrete data but not ideal. They proposed a caching strategy based on end node freezing, but this strategy has a large hardware requirement and is prone to data delay in large-scale concurrent architecture [6].

Maa et al. (2015) redefined and optimized the encoding of distributed storage systems and strengthened the management of system storage, but it still cannot adapt to a large number of small-scale frequent storage. Therefore, in the current distributed storage system, there is no special research on the data such as the perceptual data, which cannot meet the requirements of data consistency and transactional reading and writing when the data is read and written [7].

Based on the above analysis, it can be concluded that for the sensing data from the acquisition and storage links, most of the current research on wireless sensor networks is aimed at the overall grasp and design protocols to improve the overall sensor network, while taking into account the data transmission, or focus on data encryption and privacy protection, there is less special research on data transmission, and the implementation is more complicated or affects the timeliness of data transmission, so it is not universal. However, when the current wireless sensor network has been applied on a large scale, changing the overall architecture and organizational form of the network is too costly, resulting in the inability of these results in practice; traditional relational database systems and stand-alone storage systems cannot meet the needs of big data storage. The existing distributed storage system is mainly for the storage of big data objects and relies too much on the file system itself; it can’t meet the requirements of frequent additions and deletions, large number of classified queries, and conditional queries. Distributed database systems evolved by traditional transactional databases cannot effectively cope with high concurrent read and write requirements.

3 Perceptual data classification processing based on discrete dataset suffix clustering

3.1 Clustering based on suffix trees

In the current field of Internet data analysis, the use of suffix trees for clustering operations is a well-recognized clustering method. The suffix tree clustering performs clustering operations based on the index established by the suffix tree, and the efficiency is based on the efficiency of the suffix tree construction algorithm and the indexing efficiency of the suffix tree.

The clustering operation based on the suffix tree is divided into the following basic steps:

Construct a query condition to prepare to retrieve the data to be read from the storage system; perform data read operations on the corresponding fields by the storage system, preprocess the text and extract features, store the feature values in the form of discrete files or discrete strings on the host hard disk or hard disk array for clustering operations; construct a suffix tree model of the collection with a suffix tree; use the suffix tree model to select the base class, and finally merge the base classes and use the base class connectivity to determine the clustering results.

Among them, the base class refers to the label of the edge of the suffix tree. The similarity is described in the data classification and suffix tree and cluster related operations, reflecting the overlap and similarity of the texts carried by the two base classes, which are expressed as percentage values. When this percentage is greater than a value, combine them. This value is called the merge factor. The merged base class is grouped into a cluster, and this operation is repeated to determine whether each of the two base classes can be classified into one cluster. The so-called base class connectivity assumes that every two base classes belonging to the same cluster are connected. According to the connection relationship, all the base classes that can be connected are classified into one cluster. Then the clustering operation is completed.

After the clustering is completed, the category information is provided to the user who needs further operations and the current result is stored in the corresponding field of the storage system.

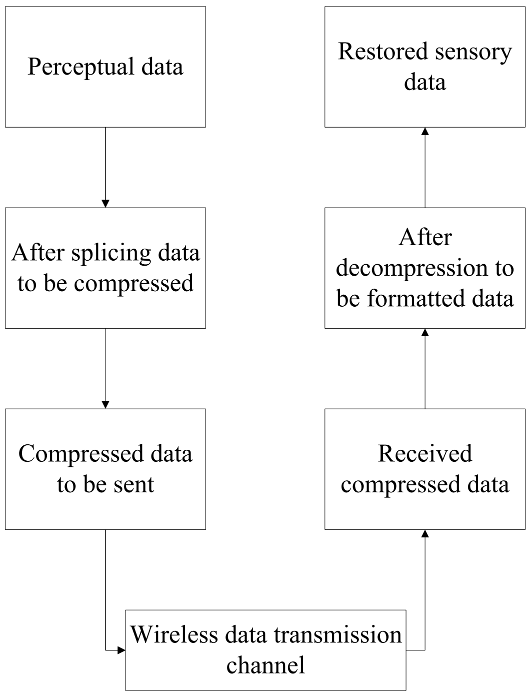
Perceptual data
After splicing data to be compressed
Compressed data to be sent
Wireless data transmission channel
Received compressed data
After decompression to be formatted data
Restored sensory data

Fig. 1. Perceptual node data compression and base station data analysis flow chart

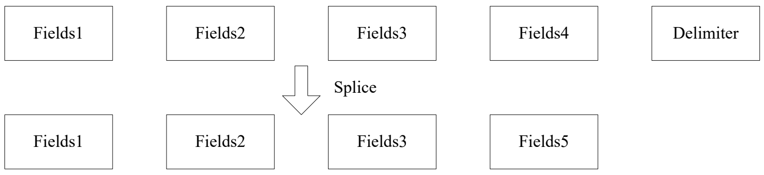

Fig. 2. Field data splice schematic diagram

3.2 Introduction of ERA algorithm

So far, no suffix tree construction method has been found for optimizing discrete data sets. Existing methods are not efficient for dealing with discrete data sets, especially for highly discrete data sets.

Chandaka et al. (2017) proposed the latest and most important suffix tree construction algorithm, namely ERA (Elastic Range) algorithm [8]. The algorithm has both serial and parallel versions, and its serial version is more efficient than all other existing methods, even faster when dealing with very long input strings with large character tables. The algorithm has no merge phase, so its parallel version is easy to implement. And its parallel version can run on shared memory and shared disk architecture systems including common multi-core desktop systems. There is no final merge phase that makes the algorithm easy to implement parallel processing. As shown in Figure 1, the algorithm partitions the problem horizontally and vertically. The vertical partition divides the suffix tree into subtrees that can be loaded into main memory. In order to maximize memory usage and reduce I/O overhead, the algorithm groups the suffix subtrees and forms a virtual suffix subtree as a separate unit for processing. The horizontal partition will process each suffix subtree or virtual subtree independently in a top-to-bottom manner. This method is very novel and very effective for suffix tree construction of data sets such as the human genome. However, this method is much less efficient for discrete data sets such as log files of wireless sensor networks or acquired perceptual data sets than continuous data sets, and the time it takes is unacceptable in practical applications.

STD is improved based on ERA. The main purpose is to optimize the processing of discrete data sets so that it can satisfy the perceptual data processing of highly discrete wireless sensor networks, and then this algorithm is used to perform clustering operations and complete perception data processing. The main work is:

An algorithm is proposed, which is optimized for the processing of discrete data sets, and it uses the latest decomposition suffix tree without the idea of merging steps. Converting data sets with content in different formats into strings for processing enhances the versatility of the algorithm. It retains innovations in I/O access and other aspects for most of the latest algorithms, ensuring efficiency in many aspects such as I/O access, memory usage.

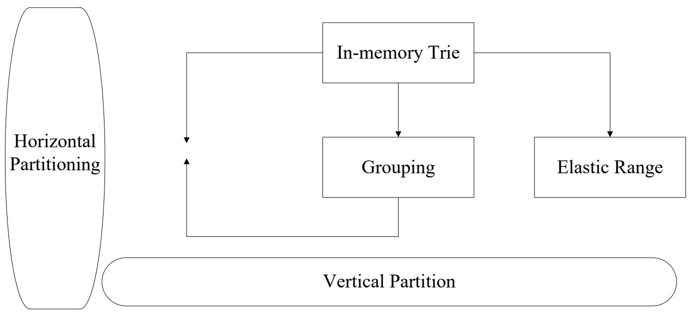

Fig. 3. Partitioning method

3.3 STD algorithm and efficiency analysis

In order to adapt to the actual situation of perceptual data processing, improve the efficiency of the algorithm in processing discrete data sets, it is optimized primarily for the vertical partition phase and adds the preparation phase before all partition operations. The preparation phase of the latest definition will preprocess the input data and prepare the input string and the alphabet containing all the characters in the string. The alphabet will be redefined instead of continuing to use the previous alphabet to ensure that this data structure can accommodate the processing of discrete data sets.

For the algorithm itself, the input data set can be a collection of data files that are either logs or other data that can be read and stored as strings, or it can be a collection of strings. Each file in the input collection creates a string and indexes each string to ensure a meaningful correspondence between the string and the file. In this subject, the input data may store discrete strings read from the storage system, strings of corresponding fields obtained directly from the storage system or the corresponding fields in discrete files. If the input dataset itself is already a collection of strings, then all strings should be indexed to ensure that the next steps are effective.

A data structure named C-Member (character member) is defined that stores a character of the input string and an array of indexes of all the strings in the input data set that contain this character. The definition A is the alphabet, which is a collection of C-Members corresponding to all the characters contained in the input data set. A is defined as an index-based data structure, so all elements in A can be represented by indexes or subscripts, which can minimize the storage space occupied by temporary results.

Defines the data structure S-prefix, which contains the prefix of the input string suffix and the array of the index of the string in the input data set corresponding to the character in the prefix. Let p be an S-prefix, and T_p be the suffix subtree corresponding to the prefix p in the suffix tree T. In T_p, each suffix corresponds to a leaf node, and the number of non-leaf nodes is the same as the number of leaf nodes. Then the size of the subtree T_p is $2f_p$*sizeof(node), where f_p is the number of suffixes prefixed by p. Therefore, T_p can be loaded into memory only when $f_p \leqslant FM$, where MS is the size of the main memory space reserved by the system for the suffix subtree, and

$$F_M = \frac{MS}{2 * sizeof(node)} \tag{1}$$

In order to split the suffix tree into suffix subtrees that can be loaded into the MS, the idea of variable length S-prefix is used, and the grouping operation is performed after the preliminary division. This algorithm is designed for discrete data sets. When generating S-prefix, the index of the string corresponding to each character will be checked, so that the string index of S-prefix is the common index of all the characters contained therein, otherwise the S-prefix will lose its meaning. First, the algorithm creates a working set that contains all the S-prefixes corresponding to C-Member, that is, creates an S-prefix for all C-Members and adds all S-prefixes to the working set. Then the occurrence frequency of each S-prefix is calculated based on the entire input data set, and all S-prefix whose appearance frequency is not greater than F_M is removed. Add a character to the end of the S-prefix that still exists in the working set. It is required to have a public string index with S-prefix and replace the original index of S-prefix with the public string index. Repeat this process until the working set is empty.

The S-prefix obtained in the previous step is added to a linked list in descending order of appearance frequency. First, the first node is added to a group, and then the linked list is traversed sequentially. When the sum of the frequencies of all S-prefixes in the group does not exceed FM, the nodes are sequentially added to the group, the S-prefix with the most common string index in the process will be prioritized into a group to maximize the memory usage and improve the processing efficiency to some extent.

The above improvements make the algorithm very effective for the processing of discrete data sets. The overall description of the preparation phase and the vertical partitioning phase is shown in Figure 4.

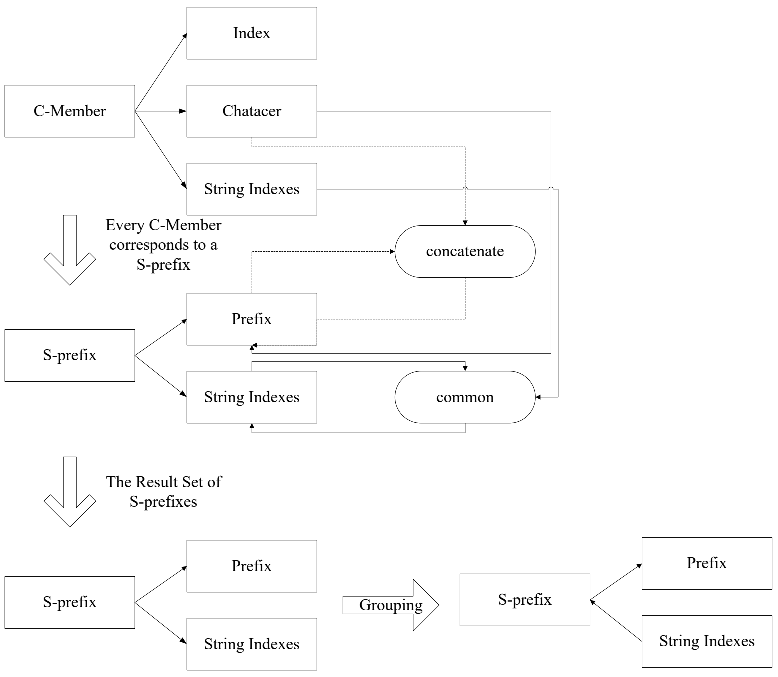

Fig. 4. Diagram of description of the preparation phase and the vertical partition phase

The subtree construction process and the resulting suffix tree are introduced. This algorithm does not have a final merge operation. Instead, it uses a dictionary tree structure of resident memory to connect the subtrees.

First, two solutions for constructing subtrees are proposed: (a) string indexing is utilized to improve the intermediate data structures used by all ERA algorithms to accommodate discrete content of the data set; (b) the string corresponding to all indexes included in the StringIndexes attribute of S-prefix is spliced. Scenario (a) seems to work, but the cost of each query string and the memory overhead required to boost the data structure will greatly reduce the efficiency of the algorithm. Therefore, option (b) is selected, and the string SC is generated by splicing and stored on the disk, and the ending character corresponding to each discrete string is marked to avoid generating an invalid path in the suffix tree.

The following propositions are established for the suffix tree:

Define SC as a string, and e is an edge in the suffix tree corresponding to the SC. Label(e) is the label of e, and parent(e) is the unique parent of e and corresponds to the string of the root node to the label corresponding to all edges on e.

Edge e connects to a leaf node if and only if pathlabel(e) appears only once in the SC.

If label(e) = $s_1 \ldots s_k$, then the neutron string pathlabel (parent(e)) $s_1 \ldots s_{i-1}$ is always followed by s_i.

The edge e can be followed by the branch $e_1 \ldots e_j$ if and only if pathlabel(e)si($1 \leqslant i \leqslant j$) appears at least once in the SC, where $s_1 \ldots s_j$ are the first characters corresponding to label(e_1) ... label(e_j), and they are different.

This proposition is used to optimize string access and form a general method of constructing suffix subtrees. Since the non-sequential and non-local access process to the string during the construction process will bring a lot of overhead, which leads to a large amount of time overhead caused by upgrading the suffix tree, an improved two-step processing method is adopted. The stitched string SC is used as an input string, but it doesn't make sense to splicing the end character of one string with the start character of another string to form an edge label, and this is an error in most cases. Therefore, the end character is used to mark the end of each discrete string. And the selection of the ending character depends on the specific situation.

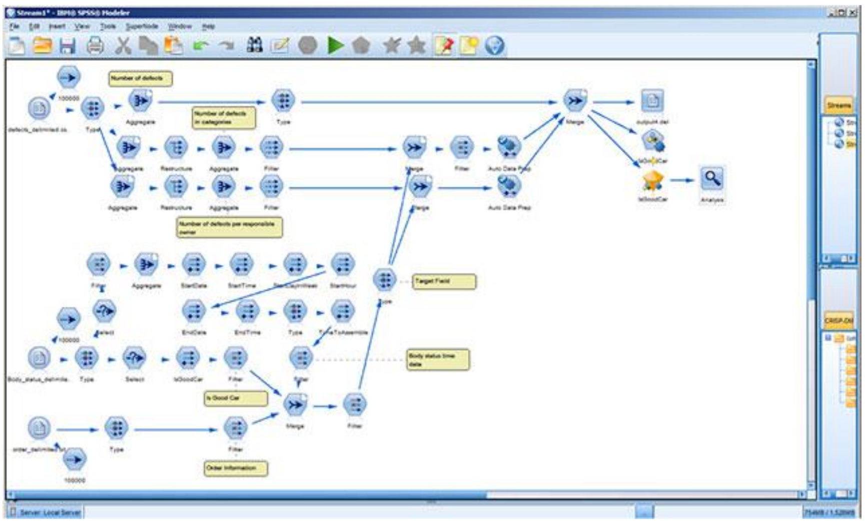

Fig. 5. Algorithm implementation diagram

The horizontal partitioning operation of the ERA algorithm uses a preliminary algorithm to generate two intermediate data structures, namely L and B, where L is an array of leaf nodes and B stores branch information. The suffix subtree is then constructed on the basis of L and B. This algorithm uses excellent ideas such as buffers and intermediate data structures, which can improve the efficiency of the algorithm.

The work is to improve the ERA algorithm and make it possible to construct a suffix tree for discrete data sets, and finally form the proposed algorithm - STD. To achieve this goal, the string index attribute is added to all intermediate data structures and the leaf nodes of the suffix tree, which makes the query of the original string easier, and the algorithm is therefore more efficient in practical applications.

4 Efficiency Analysis

The latest preparation phase runtime complexity is O(n), and the other two steps are very close to ERA. Only the improved data structure takes up more memory, but it won't be much more and can be ignored as the data dispersion increases. The specific storage space required for the algorithm to run depends on the actual situation, otherwise too many subtrees will be generated.

The log files and DNA sequences of the nginx server network are used for testing. The DNA sequence used is not large enough for a control trial, so the dataset is found again on the Internet. Based on experimental data, this change has no effect on the results of this experiment. The experimental results show that the time used by the algorithm to construct the suffix tree is almost the same as that used by the ERA to process continuous data set with the same size (not more than 5%). The used experimental environment demonstrates that the data structures used herein are valid. The experiment was also performed with a continuous data collection discrete data set to compare the efficiency of STD and ERA. The experimental data is shown in the table:

Table 1. Experimental data sheet

	ERA	STD
Continuous data set	210	213
Discrete data set	280	218

To test the efficiency of the STD algorithm, experiments are performed with discrete data sets and continuous data sets and compared to the ERA algorithm, the experimental data is shown in Table 1. On a computer platform with two quad-core 2.67GHz Intel CPUs with 1G (human limit ulimit-v) memory, the STD algorithm is used to process 4GB of DNA data for 213 minutes, almost the same as 210 minutes under the same conditions of ERA. However, when dealing with discrete data sets in this environment, ERA used 280 minutes for 4GB sensor network log files, while STD used 218 minutes, and ERA spent 28% time more than STD.

It can be concluded from the above experimental data and its analysis that the proposed STD algorithm not only inherits the efficiency of the classical algorithm for processing big data, but also has obvious effect on large-scale discrete data processing, and the efficiency is obviously improved compared with the traditional method. This feature indicates that the STD algorithm can effectively deal with the discrete characteristics of the perceptual data, and efficiently construct the suffix tree structure for the perceptual data to complete the clustering operation.

5 Conclusion

In terms of sensing data acquisition and storage, firstly, aiming at the real-time transmission requirements of data for most sensor networks and the special requirements of energy saving for sensor networks, a new data node is proposed to splicing and compressing the collected data. The impact of traditional data compression on the real-time performance of sensing data collection has been broken, and the energy-saving purpose of frequent transmission of small-scale data is realized, and the network structure management of the sensing nodes managed by the base station is also satisfied. The data collected by the base station from the node is transmitted to the storage system in real time via the wired network. The data structure is discrete, frequently read and written, large in scale, fast in growth, complex in content, and has a strong structural feature in a single piece of data. It requires targeted queries in the process of reading and writing and high concurrent reading and writing. The design proposes a distributed and consistent storage system. The system uses multi-machine group work, realizes fault-tolerant technology of replication redundancy, and has targeted design for scalability and concurrency. For the discrete content data sets such as perceptual data, an algorithm that can effectively construct the suffix tree of discrete content data sets is proposed, which lays a foundation for the clustering operation. Then, based on the algorithm, the suffix tree clustering operation is carried out, and the general steps of the current suffix tree clustering operation are studied. Based on this, the characteristics of the perceived data structure and storage are adapted. Then, the data classification operation based on suffix tree clustering is finally completed. Based on the above work, a set of perceptual data processing system that can complete the real-time collection, distributed storage, and clustering operation for the perceptual data is basically constructed, and the expected design goal at the beginning of the topic is realized. Experimental verification and theoretical analysis are arranged for each part of the work, which proves the feasibility of the scheme.

6 References

[1] Gani, A., Siddiqa, A., Shamshirband, S., & Hanum, F. (2016). A survey on indexing techniques for big data: taxonomy and performance evaluation. Knowledge and information systems, 46(2): 241-284. https://doi.org/10.1007/s10115-015-0830-y

[2] Glöckner, F. O., Yilmaz, P., Quast, C., Gerken, J., Beccati, A., Ciuprina, A., & Ludwig, W. (2017). 25 years of serving the community with ribosomal RNA gene reference databases and tools. Journal of biotechnology, 261: 169-176. https://doi.org/10.1016/j.jbiotec.2017.06.1198

[3] Qin, Y., Sheng, Q. Z., Falkner, N. J., Dustdar, S., Wang, H., & Vasilakos, A. V. (2016). When things matter: A survey on data-centric internet of things. Journal of Network and Computer Applications, 64: 137-153. https://doi.org/10.1016/j.jnca.2015.12.016

[4] Abdullah, M., & Zamil, M. G. (2018). The Effectiveness of Classification on Information Retrieval System (Case Study). arXiv preprint arXiv: 1804.00566.

[5] Makhoul, A., Harb, H., & Laiymani, D. (2015). Residual energy-based adaptive data collection approach for periodic sensor networks. Ad Hoc Networks, 35: 149-160. https://doi.org/10.1016/j.adhoc.2015.08.009
[6] Wang, J., Liu, S., & Song, H. (2018). Fractal Research on the Edge Blur Threshold Recognition in Big Data Classification. Mobile Networks and Applications, 23(2): 251-260. https://doi.org/10.1007/s11036-017-0926-6
[7] Maa, H., Cakmak, H. K., Bach, F., Mikut, R., Harrabi, A., Süß, W., & Hagenmeyer, V. (2015). Data processing of high-rate low-voltage distribution grid recordings for smart grid monitoring and analysis. EURASIP Journal on Advances in Signal Processing, 2015(1): 14.
[8] Chandaka Babi, D., Rao, M. V., & Rao, V. V. (2017). Study of Association Rule Mining for Discovery of Frequent Item Sets on Big Data Sets. International Journal of Applied Engineering Research, 12(22): 12169-12175.

7 Authors

Jun Tian is a Researcher of Foshan Polytechnic, Foshan City, Guangdong, 528137, China. His research interests include big data.

Lirong Huang is a Researcher of Foshan Polytechnic, Foshan City, Guangdong, 528137, China. His research interests include clustering.

Article submitted 27 October 2018. Resubmitted 21 November 2018. Final acceptance 12 December 2018. Final version published as submitted by the authors.