Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
October 2013, 20(5): 97–103
www.sciencedirect.com/science/journal/10058885
The Journal of China Universities of Posts and Telecommunications
http://jcupt.xsw.bupt.cn

Offline traffic analysis system based on Hadoop

QIAO Yuan-yuan[1] (✉), LEI Zhen-ming[1], YUAN Lun[2], GUO Min-jie[1]

1. Beijing Key Laboratory of Network System Architecture and Convergence, Beijing University of Posts and Telecommunications, Beijing 100876, China
2. Produce Ads, Amazon Joyo Co. Ltd, Beijing 100025, China

Abstract

Offline network traffic analysis is very important for an in-depth study upon the understanding of network conditions and characteristics, such as user behavior and abnormal traffic. With the rapid growth of the amount of information on the Internet, the traditional stand-alone analysis tools face great challenges in storage capacity and computing efficiency, but which is the advantages for Hadoop cluster. In this paper, we designed an offline traffic analysis system based on Hadoop (OTASH), and proposed a MapReduce-based algorithm for Top*N* user statistics. In addition, we studied the computing performance and failure tolerance in OTASH. From the experiments we drew the conclusion that OTASH is suitable for handling large amounts of flow data, and are competent to calculate in the case of single node failure.

Keywords MapReduce, Hadoop, cloud computing, traffic analysis

1 Introduction

For the current explosive growth of information in Internet, storage capacity and computing efficiency have already become the two major challenges for offline traffic analysis. Traditional traffic analysis is generally conducted in a single high-performance server, in which we couldn't analyze data at TB or PB level in a short time. And if the server fails, we couldn't recover it quickly without affecting the ongoing analysis. As a result, we use Hadoop cluster(http://hadoop.apache.org/) for analyzing the large amounts of flow data. Hadoop is an open source framework for writing and running distributed applications that processing large amounts of data [1]. The two most important components of Hadoop are distributed file system (HDFS) [2] and MapReduce [3] framework. There are many companies using Hadoop cluster for network log analysis and machine learning, such as FaceBook, Yahoo, and IBM. General users would like to use the Hadoop or experience the power of the cloud platform by software as a service (SaaS) (http://wiki.apache.org/hadoop/PoweredBy) in a 'pay-as-you-go' way, such as Amazon's elastic compute cloud (EC2) [4].

In the field of traffic monitoring and measurement, Youngseok Lee [5] proposed a MapReduce flow analysis program for destination port breakdown, and verified that the MapReduce-based flow analysis method improves the flow statistics computation time by 72%, when compared with the popular flow data processing tool 'flow-tools' [6] on a single host. And Yeonhee Lee [7] presented a scalable Hadoop-based parallel packet processor that could analyze large packet trace files based on above study.

In this paper, we introduce an OTASH, and study the performance and efficiency of the OTASH by analyzing a large number of real data from mobile Internet. In addition, we propose a MapReduce-based algorithm for Top*N* user statistics. With our experiments, we can conclude that OTASH is efficient and reliable for analyzing large amounts of data.

2 Model and architecture of OTASH

In this paper, our datasets were collected and analyzed by using our self-developed traffic monitoring system (TMS), which has been placed between the packet

Received date: 05-03-2013
Corresponding author: QIAO Yuan-yuan, E-mail: qyybupt@126.com
DOI: 10.1016/S1005-8885(13)60096-5

data serving nodes (PDSNs) and the backbone routers in the same city, as shown in Fig. 1. TMS is a hardware and software platform used for network traffic monitoring. The hardware platform has physical probe which collect the packets from link with the rate of 10 Gbit/s, and the original mirrored packets are pre-processing and pre-analysis by software platform, which uses detection and classification techniques such as deep packet inspection (DPI) and deep flow inspection (DFI). TMS grouped the original flow record packets into flow records, which were aggregated and extracted according to the 5-tuple (IP source address, IP destination address, source port number, destination port number, transport protocol). There are multiple fields in a flow record, include user identity (ID), the time when user access the network, the time when user log out from the network, the website address, the number of packets and bytes of uplink traffic (from user's mobile phone or computer to server), the number of packets and bytes of downlink traffic (from server to user's mobile phone or computer), etc. These fields are called the attribute information of Internet users. Attributes are separated by fixed-separated keys in each line.

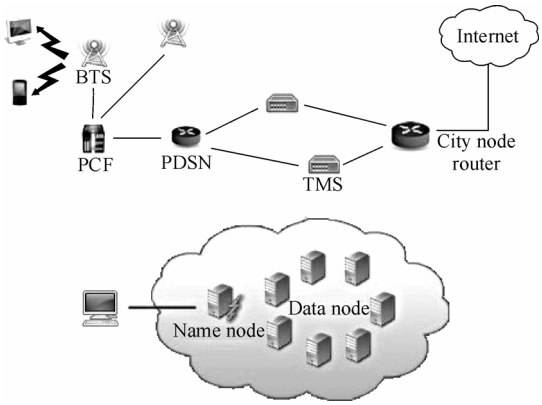

Fig. 1 Deployment of TMS in network and the deployment of OTASH

OTASH provides a distributed file system and distributed computation capabilities, it consists of a master node (Name node), a backup node for master node (Secondary name node), and a number of data nodes (Data nodes) [8]. Name node keeps track of all the files stored on Hadoop distributed file system (HDFS) and overall health of the distributed file system.

The architecture of OTASH is shown in Fig. 2.

Traffic analyzer is a self-developed flow record processing tool based on MapReduce, with the flow records as its input. We use this tool to do various types of basic statistics.

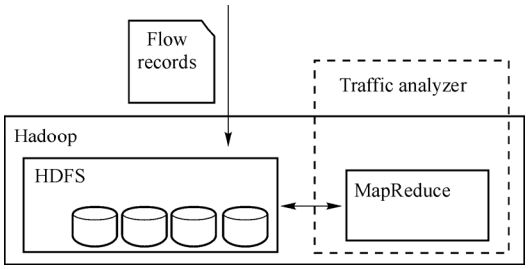

Fig. 2 The architecture of OTASH

HDFS [2] is an open source implementation based on Google's Google file system (GFS) [9], with high fault tolerance and high throughput of data read and write, which is suitable for large data sets applications. In addition, HDFS is a master/slave structure, and as the usual deployment, there is only a Name node running on the master, and one Data node for each slave. HDFS supports a traditional hierarchical file structure that is very similar to some of the existing file system in operation. For example, one can create or delete a file, move a file from one directory to another directory, and rename a file or a directory. The Name node manages the all distributed file system, and controls all the operations of file system (such as create and delete the file or folder).

MapReduce [3] is a programming model, firstly proposed by Google, who is using it to process 20 PB data every day. For the calculation of the large amount of data, parallel computing is the most common way. Meanwhile, MapReduce is simple enough for developers that have little experience in parallel computing to develop parallel applications without the need to manage inter-process communication. There is a Jobtracker running on the Name node to schedule the MapReduce jobs, and the job is broken down into a lot of tasks which are assigned to each Tasktracker on each Data node. As a user, we just need to submit the job and wait for the results, and all the works between them such as job breaking down, tasks assignment, data transmission are done by Hadoop.

There are two steps for MapReduce, Map (mapping) and Reduce (simplify). Mappers can break down one task into multiple tasks for processing, and Reducers aggregate all the output from last step to get the desired results. The middle of these two phases is Shuffle, which collects all the output from Mappers, and then aggregates, sorts and transfers these data to the specific Reducers. As shown in Fig. 3.

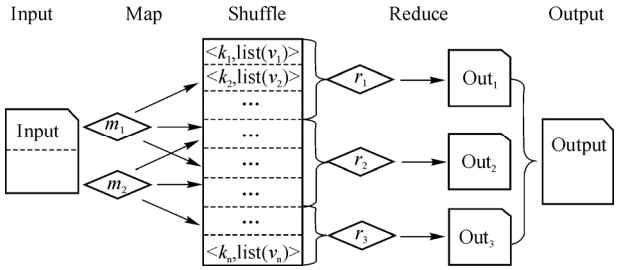

Fig. 3 The processing data model of MapReduce

The input file is broken down into multiple data blocks, each data block is processed by a mapper (m_1, m_2, as shown in Fig. 3), and Mappers output a series of key-value(<*k*,*v*>) pairs according to the functions defined by user. On the shuffle step, the *k*-*v* pairs are sorted by the value of the key, and the <*k*,*v*> pairs with same key are gathered together to <key-list(value)> pairs, which would be transferred into one Reducer. Reducers process the <key-list(value)> pairs by user-defined function for the final results. In addition, in many situations with MapReduce applications, before the map step ends, there could be some 'local reduce' steps called combine steps being done in each mapper. A combiner completes the same computation just like a reducer, and the <*k*,*v*> pairs are combined and processed in combiners, that could reduce the amount of data transferred in the network on the shuffle step and improve the computation efficiency.

According to the content described above, we can summarize the process of flow data analysis for OTASH. Firstly, OTASH upload the flow records collected by TMS to HDFS on Hadoop. Secondly, we use traffic analyzer which contains multiple data records analysis module to initiate a MapReduce job. And then, there would be a MapReduce job runing on the Hadoop cluster following our instruction. Finally, the analysis results are storaged in HDFS for further analysis.

3 The Top*N* algorithm for user traffic statistics

There is some of the peak information that we are interested in when analyzing the traffic data, such as the users with the largest download bytes in one day, the service owned the most users in each day, and the servers with the most accessing frequency, which are called Top*N* information usually. The user-dimensional statistical (the statistics of user ID for one day would be one billion or more) needs large amounts of computer memory, and if we only use one computer, this type of statistics will be running for a pretty long time or running out of memory. But this Top*N* model with large input data could be handled well by distributed computing and MapReduce model. The Top*N* algorithm based on MapReduce needs three parameters, the object for statistics, the object for sorting, and the value of *N* for Top*N*. As a result, we need to compute the sum of download traffic bytes for each user and get the *N* users from the top with the largest download bytes. The algorithm structure is shown in Fig. 4.

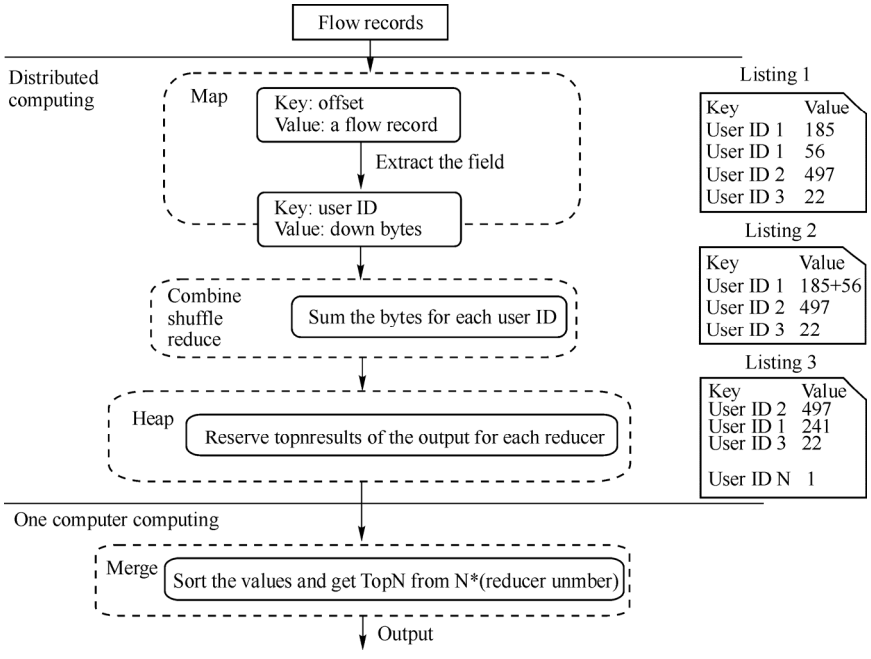

Fig. 4 The flow diagram for Top*N* algorithm based on MapReduce

The inputs of MapRecuce job for Top*N* algorithm are flow records. In the Map step, firstly, the offset of a flow record in flow records file is computed as the key, and this flow record as the value. Then, the user ID and download bytes were extracted as the key and value respectively. Therefore, in this step, the <*k*,*v*> pairs outputted from the mappers are <user ID, download bytes>, as show in listing 1 of Fig. 4. In the combine, shuffle and reduce step, we get the sums of download bytes for the same key (user ID). Combine step sum the download bytes for each user ID on each node(computer). And then, all the results from combine step on each node are gathered together in the shuffle step. At last, the sum of all the download bytes on every node with the same key (user ID) are computed in the reduce step, as show in listing 2 of Fig. 4. The MapReduce program has completed after the reduce step, and we get the sum of download bytes for each user. Since we have more than one reducer on the OTASH, and each reducer has an output file, we get more than one output file. If we merge all the output files into one file, we could get a list of the sum of download bytes for each user ID, as shown in listing 3 of Fig. 4. However, what we need is users with the Top*N* download bytes. Therefore, as for every node of OTASH, we put all the output data into the 'heap' on this node, and select *N* <*k*,*v*> pairs with the biggest download bytes kept in each 'heap', all <*k*,*v*> pairs from these 'heaps' are transmitted to one node to sort. Finally, we could get the Top*N* users with the biggest download bytes. In addition, by using 'heap' on each node, *N* <*k*,*v*> pairs with the biggest download bytes on each node are selected before data get transmitted. Therefore, the size of data transmitted between the computers is reduced. In this way, considerable computation time and resources are saved.

4 Experiment

4.1 Experimental environment

For the performance evaluation of OTASH, we built up a Hadoop cluster with one Name node (with a Jobtracker), one Secondary name node, and ten Data node (with a Tasktracker for each one). Each node has two 2.40 GHz E5620 CPU and Eight-core for each one, 16 GB memory, 4 TB hard disk, and 1 Gigabit ethernet cards. All Hadoop nodes are connected with a 1 Gigabit switch.

HDFS is used for the cluster file system, and the Hadoop version is 0.202 which is widely used in commercial development. Hadoop has hundreds of parameter to adjust the configuration, in this paper, we use the default configuration.

The size of flow records in this experiment are 50 GB, 100 GB, 200 GB and 500 GB, as shown in table below.

Table 1 The number of flow records for different size of flow records

Data set	50 GB	100 GB	200 GB	500 GB
Flow records	1.57×10^8	3.14×10^8	6.27×10^8	1.57×10^9

4.2 Computation time for flow statistic

In order to compare the computation performance for traffic analysis with the different number of Hadoop worker nodes (Data node), we analyze Top 10 users for download bytes in five Hadoop clusters with 2, 4, 6, 8, 10 Data nodes respectively, the results are as follows.

Fig. 5 shows that, with the same input flow records size, the computation time decreased with the increase of the Datanodes, and in the same Hadoop cluster, as the flow records size increased, the computation time increase as well.

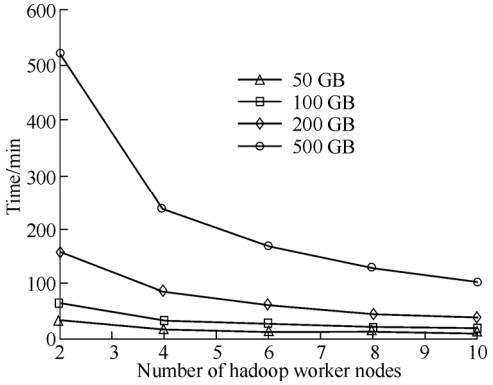

Fig. 5 Time vs. the number of Hadoop worker nodes for Top 10 users for download bytes

The time for computing 500 GB flow records in 10-datanodes cluster is 58 s shorter than computing 200 GB flow records in 2-datanodes cluster. If the input data size is 200 GB, the time consuming for 10-datanodes cluster is 4.25 times faster than 2-datanodes cluster. However, there is not a linear relationship between computation time and the number of Data nodes in cluster. In a 10-datanodes cluster, the time for computing 500 GB and 100 GB flow records is 6 118 s and 952 s, 56% and 42% faster than 4-datanodes cluster respectively with the same input. In order to quantify the difference of

computing performance between 10-datanodes cluster and other clusters (the number of Data nodes are 2, 4, 6, 8), we compute the ratio of computation time for other clusters to 10-datanodes cluster when the size of input flow records are 50 GB, 100 GB, 200 GB and 500 GB. The table is listed below.

Table 2 The ratios of computation time for different clusters to 10-datanodes cluster with different size of input flow records

The number of datanodes	50 GB	100 GB	200 GB	500 GB
2	3.44	3.46	4.25	5.15
4	1.69	1.72	2.26	2.28
6	1.19	1.44	1.61	1.63
8	1.14	1.13	1.18	1.25

From Table 2 we can conclude that, with the increase of input data size, the ratios of computation time for different clusters (the number of Datanodes are 2, 4, 6, 8) to 10-datanodes cluster are increased too. It shows that, with the increase of input data size, the advantage of multiple-datanodes is more obvious, same as the computation performance. As a result, Hadoop cluster is suitable for analyzing the large amount of data, such as offline traffic analysis. In addition, we statistic the number of user ID with the same method and get the same conclusion.

4.3 Node failure recovery

There are multiple computers in OTASH, if the processes or machines failed when job is running, we should make sure that the job is not interrupted. Fortunately, Hadoop provides the recovery mechanism to handle the node failure. Hadoop cluster considers the hardware failure as a common situation. Therefore, there has block redundant storage mechanism for high data reliability. In other words, every file that stored in HDFS has more than one copy. If one node failed, the data block on this node still can be regained from other nodes. In this section, we test the recovery mechanism of OTASH, and study the impact on OTASH during the single node failure.

In the experiment, we compute the Top 10 users with 50 GB flow records input. We shutdown one Data node when job is running at a random time. After several tests, we find that OTASH can successfully finish the job under the single node failure, however, the recovery mechanism acts differently when single node failure happened in map step and reduce step. We select two typical experiments for an intensive study. In one experiment, one node in ten is forced to shut down when the reduce step just began, and in the other experiment we shut down the same node after the map step completed, as shown in Fig. 6 and Fig. 7.

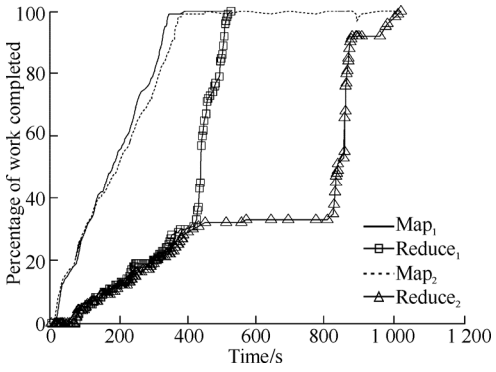

Fig. 6 The MapReduce recovery from the single node failure (failure at 112 s and the total computation time is 1 015 s)

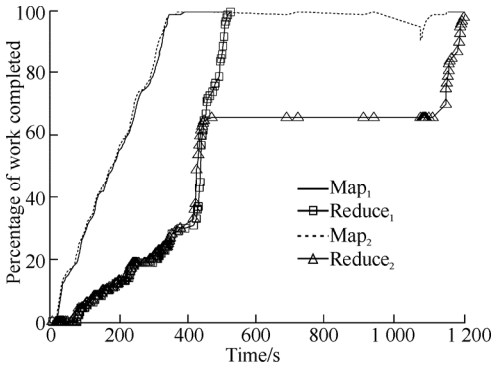

Fig. 7 The MapReduce recovery from the single node failure (failure at 470 s and the total computation time is 1 215 s)

In the figures, we draw the completion percentage of map step and reduce step with the growth of the time under the single node failure (Map_2 and $Reduce_2$ in Figs. 6 and 7). In order to compare with jobs that don't have the single node failure, the completion percentage of map step and reduce step with the growth of the time under a well-running cluster are also drew (Map_1 and $Reduce_1$ in Figs. 6 and 7). Under well-running cluster, the computing time of Top 10 users with the largest download bytes is 509 s.

In Fig. 6, a Data node is failed when the MapReduce program has been running for 112 s, at this time, Map_2 has completed 33%, and $Reduce_2$ has only completed 6%. Map_2's running time is a little more slowly than Map_1 after the failure. When Map_2 reached 100% at the first time, it is 59 s slower than Map_1's completion time. After that, Map_2 does not remain 100% all the time. Since the output from Map_2 is stored in local, and Hadoop cluster could not connect to the failed Data node, the map tasks that has completed on the failed Data node must be re-assigned to other Tasktrackers on other Data nodes and re-running,

that is called map recovery. When Map_2 is recovering, $Reduce_2$ is at a standstill, which is from 428 s to 800 s and from 872 s to 953 s. The entire recovery lasts 525 s. Fig. 7 is a little different from Fig. 6, in Fig. 7, it is at 470 s that a Data node failed, when the $Reduce_2$ has completed 66%. At this time all Map tasks have completed, and the line of Map_1 and Map_2, $Reduce_1$ and $Reduce_2$ is nearly the same. After the single node failure, $Reduce_2$'s percentage of completion is unchanging until 1 149 s, when the Map_2 is under recovery. At these times, though Map_2 has completed, the outputs from mappers on the failure Data node are no longer reachable, all these map tasks must be completed on the other well-running Data nodes before the reduce tasks restarted.

Under the single node failure above, the computation could be successfully completed to get the correct results. This shows that the Hadoop cluster could complete the tasks under the single node failure, with the cost of longer computation time, about 2~2.4 times longer. Through analyzing the log of every process on each Data node during the MapReduce recovery phase, we find that the well-running Data nodes attempt to connect the failure Data node and wait for response in the entire recovery phase. In addition, we also study the recovery mechanism of Hadoop-0.202 version by analyzing the source code. There are some defects with the recovery mechanism; firstly, the information is not shared between each process, even if the Name node has known that there is a failed node, other nodes still attempt to connect to that node repeatedly until reaching the timeout value. Secondly, the default maximize timeout is 10 min, and this timeout is in order to tolerate the network congestion and computer overloaded in a maximum way. It makes the whole Hadoop cluster waiting for a long time when single node failure happened, that is a huge waste. In order to improve these defects of recovery mechanism in Hadoop-0.202 version, we could set two kinds of timeout, connection timeout and reading timeout, reading timeout should be shorter than connection timeout, once the reading timeout is reached when attempting to get the data from some nodes, the cluster should re-assign the tasks whose result is not available at that time according to the nodes' load, even if the connection timeout is not reached. In addition, if Name node finds out the node failure, other nodes should be informed to avoid waiting for timeout.

5 Conclusions

In this paper, we designed an OTASH, and proposed a MapReduce-based algorithm for Top*N* users statistics. We studied the relationship between the computation ability, the size of input flow records and the number of nodes in OTASH. In addition, we tested the recovery mechanism of Hadoop 0.202 version when conducting the statistics of Top*N* users. The experimental results show that OTASH is very suitable for large data processing such as traffic analysis; it could solve the problem of big storage and low efficiency of one computer computation. And the computation ability for OTASH grows with the increase of input flow records size and the number of Data nodes in Hadoop cluster. Moreover, OTASH could get the correct results under the single node failure with the increase of computation time for about 10 min. This is the defect of recovery mechanism for Hadoop 0.202 version.

However, in the actual production environment, we found that optimization is very necessary for improving the computational efficiency. In the future work, we will improve the performance of OTASH by adjusting the Hadoop configuration parameters, optimizing the MapReduce-based algorithm, modifying the recovery mechanism or other mechanism(storage, job assignment, resource allocation, etc.).

Acknowledgements

This work was supported by the Important National Science & Technology Specific Projects (2012ZX03002008), the National Natural Science Foundation of China (61072061) and The Fundamental Research Funds for the Central Universities (2012RC0121).

References

1. Lam C, Warren J. Hadoop in action. Greenwich, UK: Manning Publications, 2010
2. Borthakur D. The hadoop distributed file system: architecture and design. Apache Software Foundation, 2007
3. Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters. Communications of the ACM, 2008, 51(1): 107–113
4. Turner M, Budgen D, Brereton P. Turning software into a service. Computer, 2003, 36(10): 38–44
5. Lee Y, Kang W, Son H. An Internet traffic analysis method with MapReduce. Proceedings of the 2010 IEEE/IFIP Network Operations and Management Symposium Workshops (NOMS Wksps'10), Apr 19–23, 2010, Osaka, Japan. Piscataway, NJ, USA: IEEE, 2010: 357–361
6. Fullmer M, Romig S. The OSU flow-tools package and Cisco NetFlow logs. Proceedings of the 14th USENIX Conference on System Administration (LISA'00), Dec 3–8, 2000, New Orleans, LA, USA.

Berkeley, CA, USA: USENIX Association, 2000: 291–303
7. Lee Y, Kang W, Lee Y. A hadoop-based packet trace processing tool. Proceedings of the 3rd International Conference on Traffic Monitoring and Analysis (TMA'11), Apr 27, 2011, Vienna, Austria. LNCS 6613. Berlin, Germany: Springer-Verlag,2011: 51–63
8. White T. Hadoop: the definitive guide. Sebastopol, CA, USA: O'Reilly Media, 2009
9. Ghemawat S, Gobioff H, Leung S. The Google file system. Proceedings of the 19th ACM SIGOPS Symposium on Operating Systems Principles (SOSP'03), Oct 19–22, 2003, Bolton Landing, NY, USA. New York, NY, USA: ACM, 2003: 29–43

(Editor: ZHANG Ying)

From p. 90

3. Liu H, Yu L. Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(4): 494–502
4. Zhao Z. Spectral feature selection for mining ultrahigh dimensional data. Tempe, AZ, USA: Arizona State University, 2010
5. Das K. Privacy preserving distributed data mining based on multi-objective optimization and algorithmic game theory. Baltimore, MD,USA: University of Maryland, 2009
6. Gunarathne T, Wu T L, Qiu J, et al. MapReduce in the clouds for science. Proceedings of the 2nd IEEE International Conference on Cloud Computing Technology and Science (CloudCom'10), Nov 30–Dec 3, 2010. Indianapolis, IA, USA. Piscataway, NJ, USA: IEEE, 2010: 565–572
7. Das K, Bhaduri K, Kargupta H. A local asynchronous distributed privacy preserving feature selection algorithm for large peer-to-peer networks. Knowledge Information System Journal, 2010, 24(3): 341–367
8. HAY M G. Enable accurate analysis of private network data. Amherst, MA, USA: University of Massachusetts, 2010
9. Barak B, Chaudhuri K, Dwork C, et al. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. Proceedings of the Symposium on Principles of Database Systems (PODS'07), Jun 11–13, 2007, Beijing, China. 2007: 273–282
10. Dwork C, Mcsherry F, Nissim K, et al. Calibrating noise to sensitivity in private data analysis. Proceedings of the 3rd Conference on Theory of Cryptography (TCC'06), Mar 4–7, 2006, New York, NY, USA. New York, NY, USA: ACM, 2006: 265–284
11. Mcsherry F, Talwar K. Mechanism design via differential privacy. Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS'07), Oct 20–23, 2007, Providence, Rhode Island, USA. Piscataway, NJ, USA: IEEE, 2007: 94–103

(Editor: ZHANG Ying)