# DATABASE OPTIMIZATION FOR LARGE-SCALE WEB ACCESS LOG MANAGEMENT

Minjae Lee, Jaehun Lee, Dohyung Kim, Taeil Kim, Sooyong Kang
Hanyang University {Dept. of Electronics & Computer Engineering}
{stkment, ljhokgo, cased,cardia87,sykang}@hanyang.ac.kr

## ABSTRACT

In this research, we investigate an optimal method of commercial DBMS for analyzing large-scale web access log. Also, we develop the pre-processor in memory structure to improve performance and the user management tool for user friendliness. The optimal method of DBMS for management of large-scale web access log is the key issue in this research. We have three stages for this research. The first stage in research is data collection. In this stage, we study characteristics of large-scale web access log data and investigate commercial DBMS tuning techniques. The next stage, we design the system model. The proposed system has three components including pre-processor, DBMS and management tool. To improve the effectiveness the system, Pre-processor hash and sort log data in memory. And we perform DBMS tuning to improve performance. The final stage, we implement system and evaluate the performance. And we develop system management tool for user friendliness. The research output of this project can be divided into analysis of large-scale DBMS, DBMS tuning and system management software. From output, we provide DBMS tuning techniques for large-scale data log. Also it is expected to use of collected data and analyzed information in other area such as network security.

## KEYWORDS

Web access log, Pre-Processor, Database, Data mining, Hash table

## I. INTRODUCTION

Log generally collects some data for analysis, such as, click rates, type of users, how many users access each web page, and time-of-use [1]. A large amount of data about web access log information is consistently stored. Therefore, Database for big data promptly saves big data and needs to be optimized performance to provide rapid analysis and accurate results.

Commercial DBMS with high performance and high scalability is used by many companies in the industry, such as Oracle, Microsoft, IBM, and etc. It had better focus on efficiency of DBMS through data tuning and performance management solutions than the new introduction of DBMS.

In order to optimize the performance of DBMS with high performance, it needs to widely and deeply understand OS, hardware system, DMBS, and application etc. because the inappropriate optimization cause the performance degradation. Even if it can be to get the high performance by using costly DBMS without optimization method, it's a drawback that initial introduction and maintenance cost are costly. However, it can be to get more advantage if it's possible to get the same performance financially when optimized DBMS is used to make the efficient data architecture through analyzing the characteristics of data.

The main point of this study is about how to optimize DBMS to efficiently store and to manage web access log data. We figure out which factor makes bottleneck when web access log data is stored. We also set up Pre-Processing, the efficient record architecture, and the characteristics of data to get rid of bottleneck. Furthermore, we study on the optimization of DBMS technology about web access log to get the optimized performance for creating index and processing data according to the pattern of the query language.

## II. RELATED WORK

### *A. Web Access Log Data*

The collected web log data is a typical log includes source and destination IP, the time of collection, etc. This study sets the standard data is source IP, destination IP, Domain, URL and generates statistical information about the whole URL. The data related to a web page on the part

This work was supported in part by **M**id-career Researcher Program through NRF grant funded by the MEST (No. 2011-0029181)

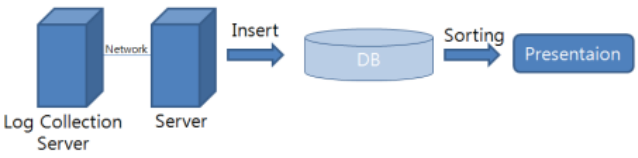


**Figure 1 The system using only DB**

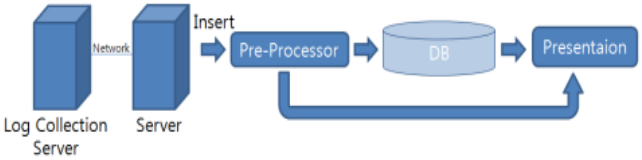


**Figure 2 The proposal system using Pre-Processor**

of a user request is requested to a server so each of a user request leaves various web log data. These web log data are recorded at different times, so they are recorded even if the same page is requested because they are considered the different requests.

### B. *Hash*

Hash is that a value is stored on a specific address using hash function. After a value is processed by hash function, it is mapped to a hash table. A hash table is composed of a bucket that store data and a hash address that corresponds to a key value. When a hash address is formed by hash function, each different key might have the same hash address, which is called Hash Collision [2]. It might be good to appropriately use hash function not to occur hash collision, but it can't be avoided. And, it's a waste of space if there are too much arranged addresses to use, because there are too much unused addresses. Therefore, a hash table size is commonly from 1.2 to 2 times bigger than given data. This hash has an advantage that it's possible to get specific data with only one access using a key.

### C. *Bucket*

A value is mapped to a hash table. A hash table is composed of n buckets. Each bucket has m slots and each slot can store only one value. Table 1 shows the hash table is composed of n bucket and 2 slots per a bucket. Each bucket can store two data. That is, it can be possible that two values are stored in the same address by hash function. In table 1, R1 and R2 are stored in the same hash address. However, should other value is already mapped to address 0, it will be hash collision. To avoid it, there are open addressing and chain.

**Table 1 Hash table structure**

| | slot 0 | slot 1 |
|---|---|---|
| 0 | R1 | R2 |
| … | | |
| n | | |

## III. THE OPTIMIZATION OF DATABAE DESIGN

This chapter describes the design of Pre-Processor, the analysis system of web access log, how it works, and web log analysis.

### A. *System Analysis and Design*

Like we said, to promptly store data and to provide rapid analysis, accurate result, DB needs to be optimized. However, it takes a long time to store all URL to DB because there are a lot of data and it takes a long time to access data.

Figure 1 illustrates log collection server sends data to DB through the network. The server gets data from log collection server and sends it to DB, and then the data is directly analyzed in DB. The main problem of this system is that all data are put in DB to handle [3].

It needs a lot of time to store and to analyze big data. If saving big data, which often accesses to disk, is processed in memory, an amount of data is decreased. And, it gains a benefit of time because it reduces the number of access time. To get this benefit, Pre-Processor is needed to improve run- time and to relieve the workload of DB by understanding what type of data is stored to DB and handling it appropriately.

Figure 2 illustrates the proposal system decreases an amount of data and maintains all information of data, also, it focuses all run time. It can relieve the workload of DB using Pre-Processor without sending data directly from log collection server to DB.

Should redundant data in collected web access log data, based on URL, is removed, all amount of data are decreased. The redundant data is one of element to cause bottleneck when data is stored in DB, so all amounts of data are decreased and statistical information about all URL is generated by removing the redundant data. By counting a number of redundant data, the entire input data is maintained and the effect of reducing the result data can be obtained. The result data can maintain statistical information using a hash table and sort data in counting order using a list. Because data is stored into random location in a hash table by hash function,

it costs a lot to access data in sorted order [4]. That is, it should check all slots because of the characteristic of a hash table. To make up for this problem, it uses a list for sorting statistical information. The use of hash is expected to quickly process big data because O(1), which is time complexity of a hash table, promptly accesses to stored data in memory [5]. In order to maintain statistical information in a different way, we use dataset. Dataset not only represents DB in itself, but also is an object to provide some information for analyzing and handling data. It's possible to operate the same as connectionless DB, but it's not good for big data because accessing speed will be slow depending on increasing the amount of data

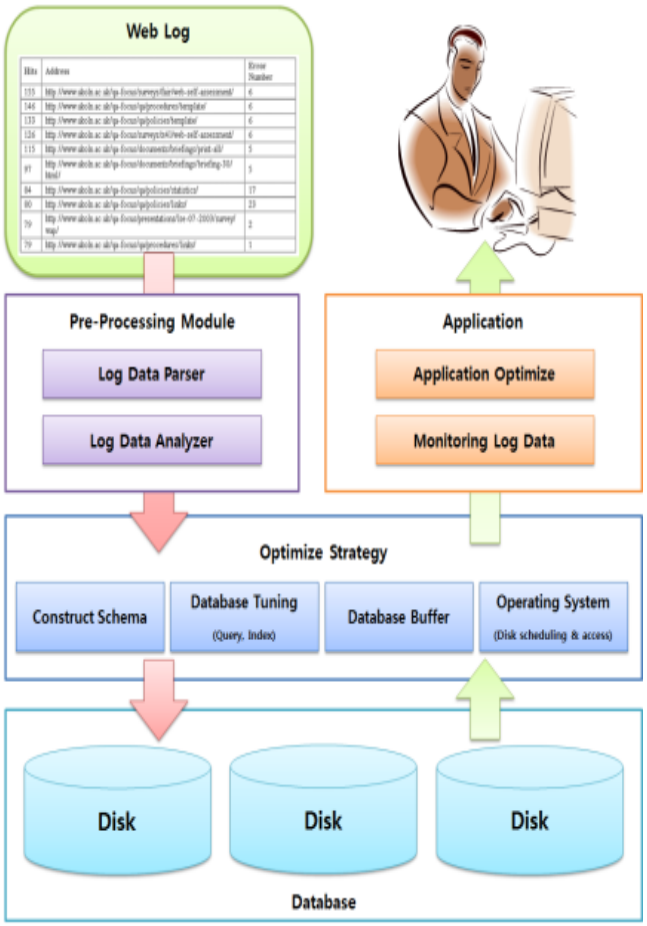


**Figure 3 The system structure**

### B. *Pre-Processor*

Pre-Processor remove redundant by reading the collected web log data and is the role of preprocessing for storing data in DB. Pre-Processor transfers collected data through log collection server and read data is stored in a hash table using a key. At the moment, if redundant data about URL is in a hash table, then increase a count value and make statistical information about collected URL [6].

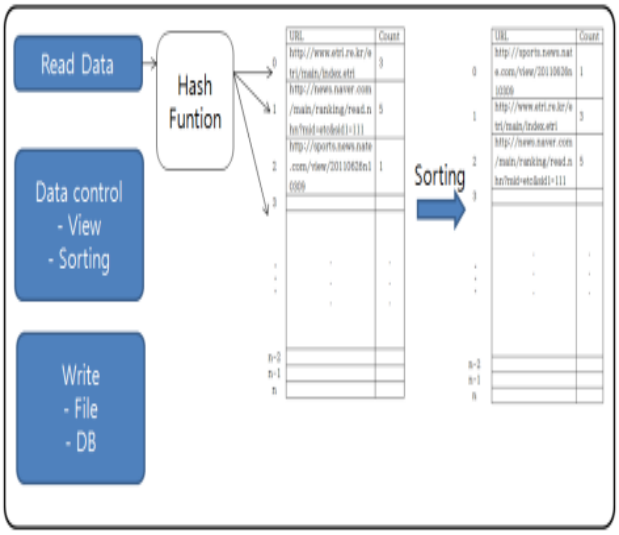


**Figure 4 Design of Pre-Processor**

Also, it performs the same task for domain to reduce an amount of data. Statistical information about URL is helpful than statistical information about domain to accurately analyze.

### C. *Log Analysis Algorithm*

With key/value pair, data is divided into a key and a value. Time and source IP are stored into a key, on the other hand, other information and a count value are stored into a value. Basically, we compare a key using key/value pair and increase a count value if the key exits in a hash table.

Because the hash table in this study uses 32-bit hash value, collision probability is 1/4.284.967.296.

### D. *Transaction Processing*

Statistical information made by Pre-Processor has mostly a lot of URL that access frequency is 1. In case of URL that access frequency is high is because it brings information related to a web page without a direct request, such as, images, advertisement, etc.

Figure 5 illustrate that 42 log is saved when a sing page is connected. The 80% of full URL is URL access frequency is 1, and is mostly associated with the stored data related one request. We specify one normal data is one transaction per a request and apply to reduce URL, which is URL access frequency is 1.

Transaction processing is a way to sort out some irrelevant data as one transaction is considered all data related to a request page. Also, it processes log in n seconds, which is considered a unit of one transaction. After processing, the processed log is split among two

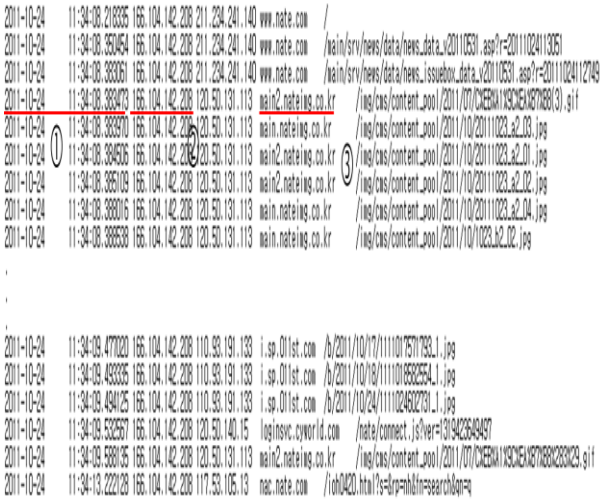

**Figure 5 Access Log**

results, such as, a result list and a filter list. The result list is a list without filtering, that is, an aggregate of unfiltered URL in the whole log. And, the filter list performs a role of filtering redundant URL in transaction per n seconds.

In Figure 5, ① is a collected time in one transaction, ② is source IP, and ③ is domain address. Both of a result list and a filter list are composed of three of them, source IP, domain address. If the same log doesn't exist in a result list, that is, some log is the fastest collected time in one transaction, it will store in the result list. If not, log will be stored in a filter list

Every time one transaction is ended, a filter list is initialized. However, a result list holds log until all log data are processed. So, it might be possible that some redundant log data exits in a result list, because even if some log data is filtered while processing some transaction, it might not be filtered while processing the other transaction.

All data in a result list are used for input data of Pre-Processor to reduce redundant data and to do counting, therefore, a number of URL, URL access frequency is 1, is decreased.


**Figure 6 Filter list and Result list**

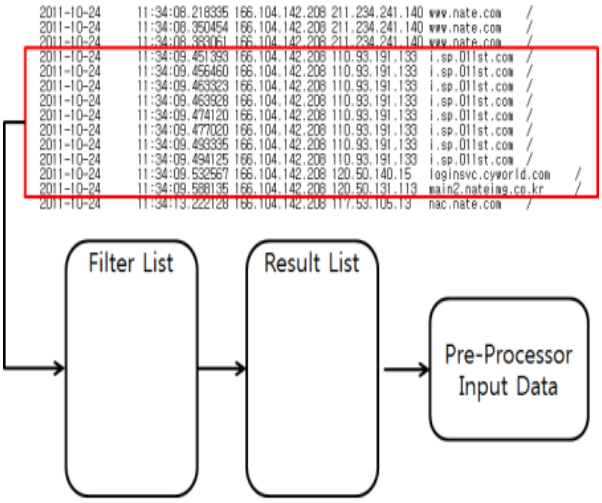


**Figure 7 Transaction Filter**

## IV. PERFORMANCE EVALUATION

### A. *Experiment Environment*

This study is that Pre-Processor for the analysis of web access log access is created and how to make statistical information through collected data. To know how redundant data, makes a bottleneck and influences run time, we study the creation of statistical information and how to sort and store it. In addition, we compare the performance with Pre-Processor and the traditional system. We set up the sufficient memory size so that we can save web access log data sufficiently, because this study is based on memory unlike the traditional system that accesses a hard disk. And, we try to keep reliability about the result numerical value as the average of experiment 10 times is used. The system for this study is as follows

**Table 2. Specification of the system**

| | |
|---|---|
| CPU | Intel i5 760 |
| RAM | DDR3 8G |
| OS | Windows 7 64bit |
| DB | Oracle 11g |
| Language | c# |

### B. *Experiment Data*

1. Experiment Data

For this study, we make URL Generator and some experiment data about Zipf distribution, Worst case, and Log file. URL Generator is intended for recording collected web access log and creating a similar experiment data as Zipf distribution is applied to an experiment data generator.

2. Zipf Distribution

Zipf's Law is that the frequency of a specific item is proportional to a priority item as creating the optimized data for an experiment. That is, the frequency of n priority item is that a proportion of some element is proportional to 1/priority with the law that the frequency is 1/n.

An example data is created in order of the high frequency from some collected sample data (URL). Experiment data is generated by applying the Zipf distribution to the example data and has the same distribution as shown in Figure 9. The distribution of the experiment data is similar to the frequency distribution of URL [7].

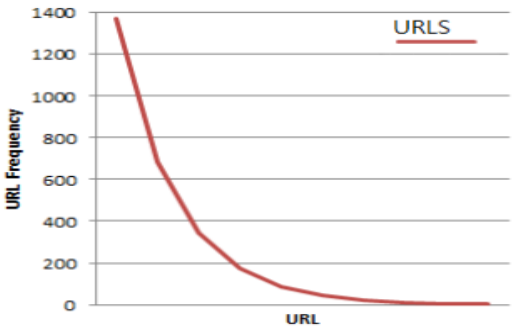


**Figure 8 Zipf distribution**

### C. *Worst Case*

Worst case (WC) is that collected web access log is not duplicated and there is no constant distribution for web access. We expect the worst performance because all collected data are stored in DB and data de-duplication is not carried out.

### D. *Log File*

A sample data for creating experiment data uses URL Snooper and is composed of collected time, source IP, destination IP, and domain, URL. We set up to collect over a million data in a server.

## V. EXPERIMENT RESULT

### A. *Data Input Time*

Figure 9 illustrates the data input time to DB. Zipf data and WC data are the input time after de-duplication at Pre-Processor and the traditional system is a million of the input time without data de-duplication.

Zipf parameter 0.2, which includes the most redundant data, takes 437.5520 seconds to store 376,934 logs to DB is and the worst case, includes no redundant data, takes 1044.4392 seconds.

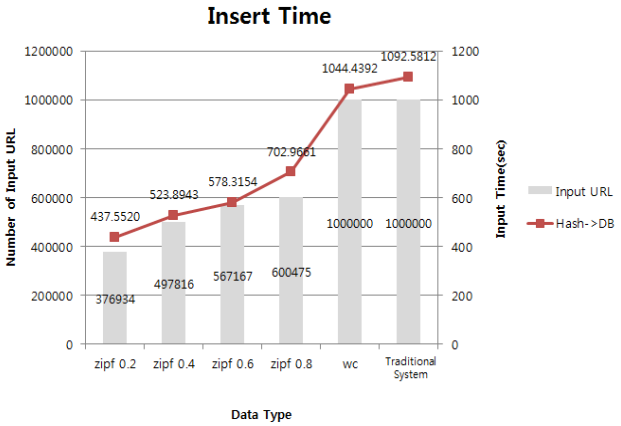


**Figure 9 Input time**

### B. *Writing Data to File*

We measure how long it takes Pre-Processor doing data de-duplication and writing statistical information to a file. We don't consider using DB because the schema of statistical information stored in DB is simple.

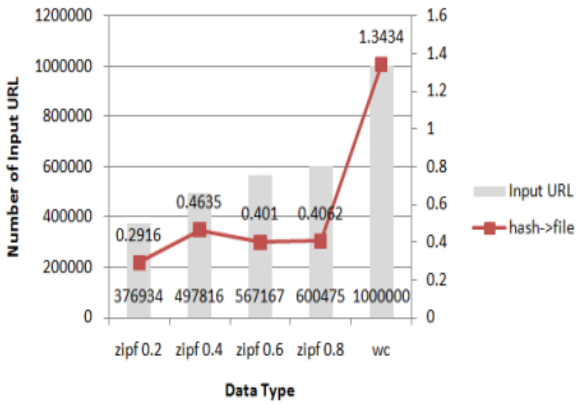


**Figure 10 File Write time**

### C. *Creation of Statistical Information*

Pre-Processor generates statistical information by reading data from a hash table. Figure 11 illustrates how long the counting job takes while doing data de-duplication.

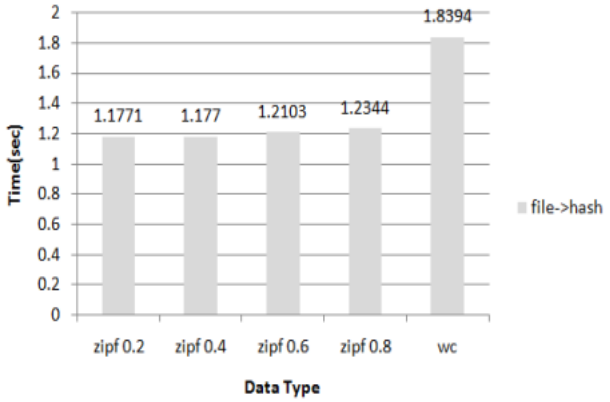


**Figure 11 Data input time to a hash table**

### D. *Sorting Time*

We measure how long Pre-Processor takes to sort statistical information in counting order and DB, which is the traditional system, takes to store a million data in it. The operation in DB is Pre-Processor sends a query to DB, also, DB reads all stored data and returns a result after de-duplication and sorting. After that, the result is stored in list data structure to be read by memory.

**Table 3 Query for sorting data in DB**

| | |
|---|---|
| SELECT url, count(url)<br>as u_count<br>FROM url_data<br>GROUP BY url<br>ORDER BY u_count; | url: log record, url_data: log record table |

**Table 4 Normalize value**

| Zipf 0.2 | Zipf 0.4 | Zipf 0.6 | Zipf 0.8 | Worst Case |
|---|---|---|---|---|
| 20.6259 | 17.4486 | 18.3436 | 18.5592 | 16.4108 |

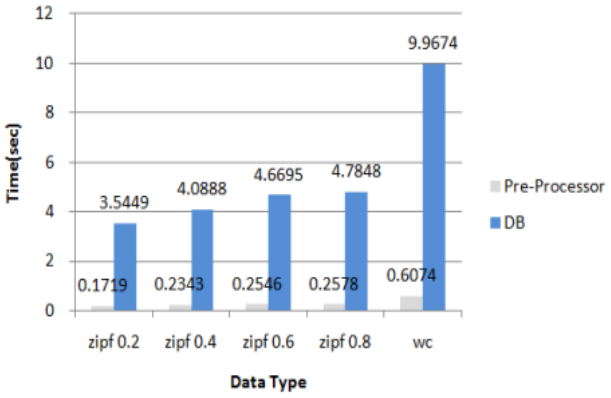


**Figure 12. Data sorting time**

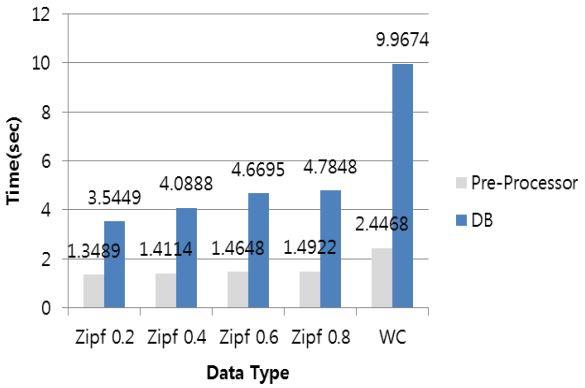


**Figure 13 Run-time**

### E. *Run Time*

Run time is the total amount of time for reading data, creating statistical information, and sorting. As shown in Figure 13, Pre-Processor is the amount of time for storing data in a hash table, creating statistical information, and sorting. And, DB is the amount of time for getting a query from Pre-Processor, grouping and sorting data, and returning a result to Pre-Processor.

### F. *Memory Usage*

As shown in Figure 14, it is the memory usage while composing a hash table and sorting at Pre-Processor. It is possible to do experiment in memory because the size of memory is enough for handling a million data. And, all of the usage is not big and it can gain an advantage of performance even if a memory space is considered to sort.

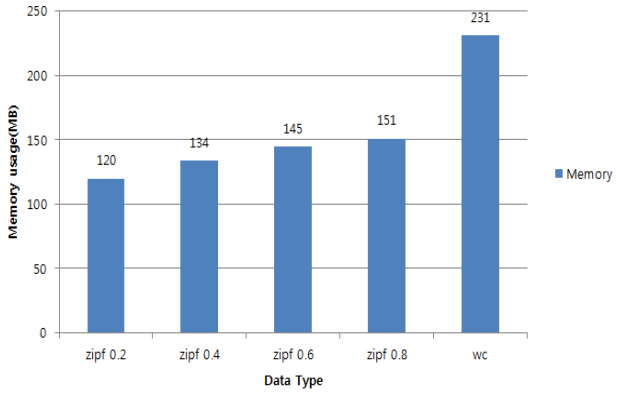


**Figure 14 Memory usage while sorting**

### G. *Experiment about Over A Million Data*

As shown in Figure 15 and 16, they are the experiment about over a million data. We test the performance, according to the size of data, by increasing data until 3 million.

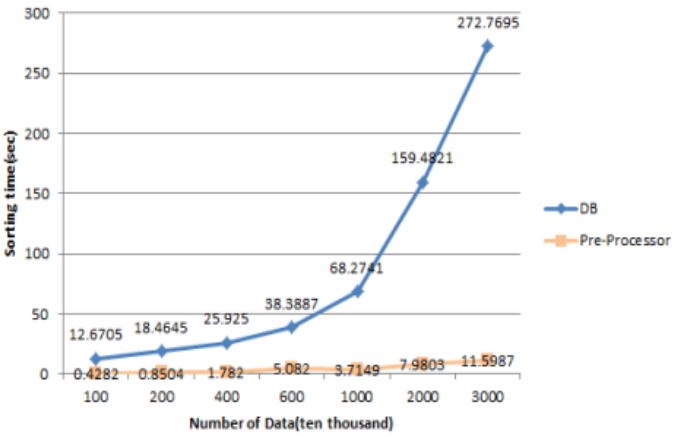


**Figure 15 Data sorting time of over a million data**

### H. *Performance of Transaction Processing*

Transaction processing is the role of reducing the number of data, access frequency is 1. It's an advantage to check and handle result data because the result data is decreased after transaction processing. However, it takes a long

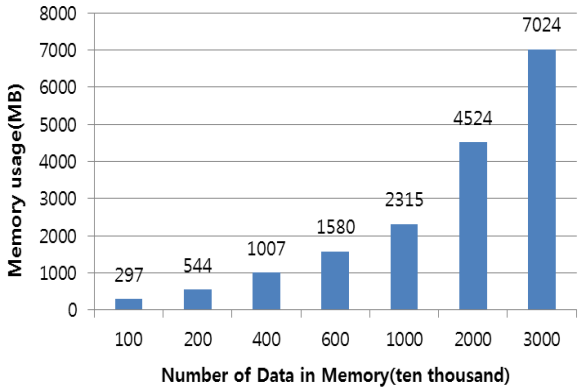


**Figure 16 Memory usage of over a million data**

time to operate because it needs to compare with each log data. Figure17 illustrates times of handling log files about transaction processing.

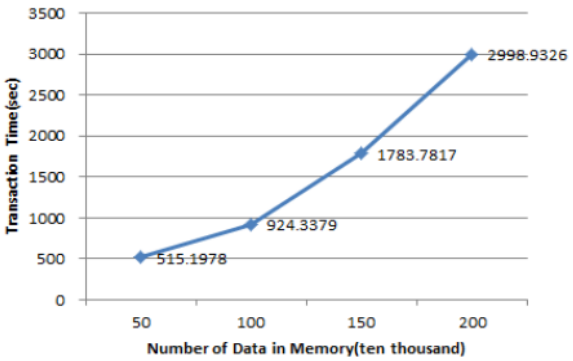


**Figure 17 Time of Transaction processing**

## VI. CONCLUSION AND FUTURE WORK

We understand the architecture and the feature of DBMS through how to store, index structure, and data storage structure of commercial DMBS. And, using the feature of DBMS, we make the optimized DBMS and the tool for handling web access log data and providing fundamental knowledge to develop software. The main experiment is about the case of a million data and the results of all experiments are the average of 10 times the result. In addition, we do experiment on over a million data.

The input time from Pre-Processor to DB depends on how often de-duplication occurs in a hash table. The memory usage is Pre-Processor uses the memory the twice time more than DB when creating statistical information. However, run time is Pre-Processor, which operate in-memory, is from 18 to 20 times better than DB. In addition, it can reduce the capacity of result data and gain more meaningful data with transaction processing even if it loses the time gain.

In summary, using Pre-Processor, works in-memory, than only using DB is expected to better performance, run-time, and the result data than the traditional system. Furthermore, this study is expected to be used from other area like network security.

## REFERENCES

1.Hyung-Woo Lee, Tae-Su Kim, "High-Speed Search Mechanism based on B-Tree index Vector for Huge Web Log Mining and Web Attack Detection", Journal of Korea Multimedia Society , vol.11, pp.1601-1614, Nov 2008

2. Chang-Wook Park, Sun-Young Hwang, "Fast URL Lookup Using URL Prefix Hash Tree", Journal of KIISE, vol.35, pp. 67-75, Feb 2008

3. Hsin-Chih Lin, Mei-San Choi, "Mining Web Usage within a Local Area Network", IACSIT, vol.2, Oct 2010

4. Z.Zhou, T.Song, Y.Jia, "A High-Performance URL Lookup Engine for URL Filtering Systems", Communications(ICC), 2010 IEEE International Conference on

5. N.Huang, R.Liu, C.Chen, Y.Chen, L.Huang, "Fast URL Lookup Enging for Content-Aware Multi-Gigabit Switches", AINA2005 19th International Conference

6. XIAO Ming-zhong, Min Bo-nan, WANG Jia-chong, DAI Ya-fei, "Practical Hashing Function for URLs Set", Journal of Chinese Computer Systems, Mar 2006

7.Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker, "On The Implications of Zipf's Law for Web Caching", Technical Report #1371, April 1998