*International Journal of Information Technology and Knowledge Management*
*July-December 2009, Volume 2, No. 2, pp. 365-370*

# PERFORMANCE EVALUATION OF MEMORY MAPPED FILES WITH DATA MINING ALGORITHMS

**S. N. Tirumal Rao*, E. V. Prasad** & N. B. Venkateswarlu*****

The concept of memory mapped files is widely supported by most of the modern operating systems. A study is carried out with computationally intensive applications such as data mining, to see the benefit of this concept over the conventional file I/O which calls standard library function fread( ). Experiments are carried out in memory mapping with both simulated and real data. The observations indicated that the memory mapping based versions taking less CPU time compared to fread() based ones.

***Keywords:*** Memory Mapping, Virtual Address Space, mmap(), Fread(), Data Mining, Clustering.

## 1. Introduction

The goal of Data Mining is to discover knowledge hidden in data repositories. This activity has recently attracted a lot of attention. High energy physics experiments produce hundreds of TBytes of data. Credit card banking sector hold large databases of customer's transactions and web search engines collect web documents worldwide. Regardless of the application field, Data Mining (DM) allows to 'dig' into huge datasets to reveal patterns and correlations useful for high level interpretation. Finding clusters, association rules, classes and time series are the most common DM tasks. Evidently, classification algorithms and clustering algorithms are employed for this purpose [14]. All require the use of algorithms whose complexity, both in time and in space, grows at least linearly with the dataset size. Because of the size of the data, and the complexity of the algorithms, the DM algorithms are reported to be time consuming [1] and hinder quick policy decision making. There are many attempts to reduce CPU time requirement of the DM applications [19, 9, 6].

In recent years much research and development (R&D) has been focused on the design of hardware and software systems that can cope with the growth in the dimensions of the data. Nevertheless, the amount of physical memory of modern computers is still in orders of magnitude lower than the size of many databases. Various strategies can be applied to tackle this problem. One approach investigates the development of new algorithms that reduce the need for data, either by exploiting sampling techniques or by limiting direct access to the database. These methods usually introduce some inaccuracy or useless CPU overhead. Another possibility is the exploitation of High Performance Computing Systems, which can extend the range of applicability of DM algorithms, without changing the conceptual limitation.

Many DM algorithms require a computation to be iteratively applied to all records of a dataset. In order to guarantee scalability, even on a serial or a small scale parallel platform (workstation cluster), the increase in the input or output (I/O) activity must be carefully taken into account. Paolo Palmerini [21] recognized two main categories of algorithms with respect to the patterns of their I/O activities. Read and Compute (R&C) algorithms, which will be useful for the same dataset at each iteration, and read, compute and write (RC&W) ones, which at each iteration rewrite the dataset to be used at the next step. Paolo Palmerini [21] suggested the employment of 'Out-of-Core' (OOC) techniques which explicitly take care of data movements which reported to be showing low I/O overhead. An important OS feature is time-sharing among processes; widely known as multi-threading, with which one can overlap I/O actions with useful computations. Kilin *et. al.* [16], Gregory Buehrer [12] demonstrated the advantage of such features in order to design efficient DM algorithms.

In the recent years, many network and other applications, which demand huge I/O overhead, are reported to be using a special I/O feature known as mmap() to improve their performance. For example John *et. al.*, [20] studied the performance of Apache Server. In addition, Avadis Tevanian *et. al.*, [7] have been reported that there would be the CPU time benefit by making use of memory mapping rather than conventional I/O in Mach Operating Systems. Nevertheless Joseph, *et. al.*, [11] had also reported that there was clear cut performance advantage of mmap over fread and iostream. Because of the huge datasets to be accessed, most data mining algorithms also must consider the I/O actions carefully, hiding or minimizing their effects.

* Dept of C.S.E, Rao & Naidu Engg. College, Ongole, Andhra Pradesh, India. *Email: naga_tirumalarao@yahoo.com*
** J.N.T.U.C.E, Kakinada, A. P, India. *E-mail: drevprasad@yahoo.co.in*
*** AITAM, Tekkali, A. P, India. *E-mail: venkat_ritch@yahoo.com*

Most of the commercial data mining tools and public domain tools such as Clusta [2], Xcluster [26], Rosetta [5], FASTLab [10], Weka [25] etc., support DM algorithms which accepts data sets in flat file form or CSV form. Thus, they use standard I/O functions such as fgetc(), fscanf(). However, fread() is also in wide use with many DM algorithms [27, 23]. Moreover, earlier studies Jayaprakash Pisharth [23] indicated that kernel level I/O fine tuning was very important in getting better throughput from system while running DM algorithms.

The study of performance for popular clustering algorithms, *k*-means [14], max-min [15], *k*-medoid [14] by employing mmap() facility existing under popular operating systems such as Linux and Windows$^{XP}$ is reported with this paper. Comparisons are also made with those of fread() based implementations. Experiments were carried out with both synthetic and real data.

## 2. TRADITIONAL FILE I/O

The traditional way of accessing files is to first open them with the open system call and then use read, write and lseek calls to do sequential or random access I/O. In general, a system call leads to a "context switch", forcing the application program to stop, while the kernel program performs the requested operation. With these views, a system call takes longer than a normal function call within our own program. However, standard library calls *fopen()*, *fread()*, often optimize operations to minimize the number of system calls and thus speeds up the program execution [11].

In addition, the standard I/O library function fread() is a buffered one [18, 24, 17, 8]. It refers that, when a fread() call is made, a block of data is read, which may be more than the requested amount of data (by calling read() immediately). The extra bytes are held in the buffer. When our program next calls fread(), it may be able to satisfy the request using the bytes already in the buffer. It delineates the elimination of the need for another read() system call. Thus, the fread() is more efficient than the read() [9]. Because of these reasons, many public domain, commercial DM algorithms are making use of fread() [4].

### 2.1 Memory Mapping

A memory mapping of a file is a special file access technique that is widely supported in popular operating systems such as Unix and Windows$^{XP}$ [18, 24, 17, 8]. It is reported that the mapping of a large file in to the memory (address space) can significantly enhance the I/O system performance [20]. There is a significant performance difference between accessing a cached disk record through disk service call (fread) and accessing a memory record (with mmap()) [1]. Disk service requests require a context switch from user to supervisory mode, even if the data is in the cache, memory access calls do not. Such context switches are relatively expensive operations; for example reading a byte from a file by using fread() incur up to three I/0 operations [18] (Figure 1) such as: (1) Removing a cache block to accommodate new disk block. (2) Reading this disk block into the buffer cache. (3) Copying the same block into the process address space from buffer.

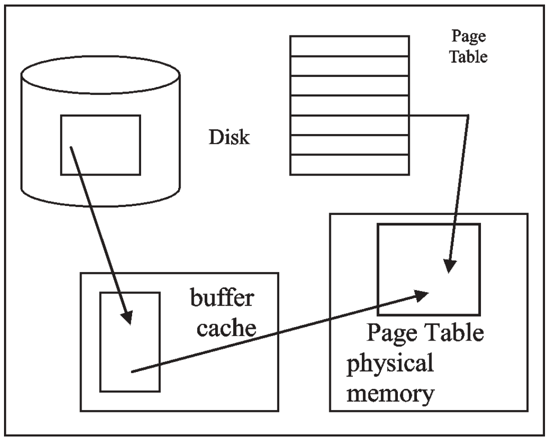

**Figure 1: Steps Involved in fread()**

If the removed cache block is dirty, it needs another I/O to update dirty block to disk. However in the memory mapping, entire file is mapped into the process address space such that the file is treated as an extension of virtual address space of the process. Here, when we need a byte from the mapped file, the block (Having the byte) is directly copied to memory. Only if, the process is not having enough frames then one of the frames is released to accommodate this new block. Hence, we may need at the worst two I/O operations (Figure 2). When the removed block is dirty, it needs another I/O to update dirty frame to disk. Additionally the program code is smaller with file mapping, because the file is accessed through a pointer, like random access memory and no file system calls need to be used [18].

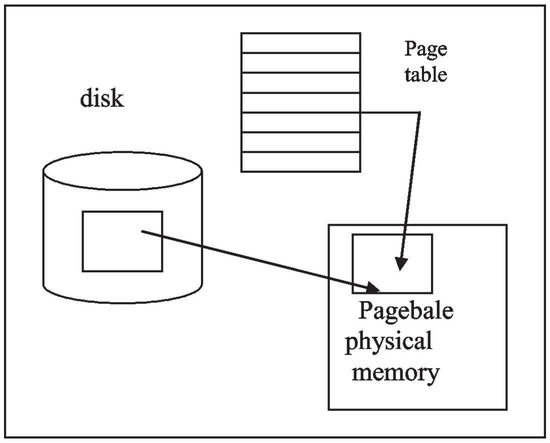

**Figure 2: Steps Involved in mmap()**

## 3. PERFORMANCE STUDY OF DM ALGORITHMS WITH MMAP()

In order to investigate the performance of mmap() over fread() in DM, we have selected clustering algorithms *k*-means, max-min and *K*-Mediod.

### 3.1 *k*-means Algorithm

The *k*-means algorithm is based on minimizing the sum of squared distances dis(*X*) from all input vectors *X* in the cluster domain to their cluster centres [19].

The steps of the *K*-mean algorithm are given below [14].

i) Select randomly *k* points (it can be also examples) to be the seeds for the *centroids* of *k* clusters.

ii) Assign each example to the *centroid* closest to the example, forming in this way *k* exclusive clusters of examples.

iii) Calculate new *centroids* of the clusters. For that purpose average all attribute values of the examples belonging to the same cluster (*centroid*).

iv) Check if the cluster *centroids* have changed their "coordinates". If yes, start again form the step ii. If not, cluster detection is finished and all examples have their cluster memberships defined.

### 3.2 max-min Algorithm [15]

1. Pick up randomly one of the sample as one cluster centre.
2. Calculate the distance of all other samples from the available clusters.
3. Calculate the maximum of the distances.
4. If the maximum distance is greater than the threshold take the respective sample vector as a cluster centre.
5. Repeat till a no more cluster is created.

### 3.3 *k*-medoid Algorithm [14]

The basic strategy of *k*-medoid clustering algorithm is to find *K* clusters in *N* objects by first arbitrarily finding a representative objects (medoids) for each cluster. Each remaining object is clustered with the medoid to which it is most similar. The strategy then iteratively replaces one of the non-medoids as long as the quality of the resulting clustering is improved. This quality is estimated using a cost function that measures the average dissimilarity between an object and the medoid of its cluster [14]. The steps of the *k*-mediods algorithm are given below. Arbitrarily choose *K* objects as the initial medoids;

***Repeat***

Assign each remaining object to the cluster with the nearest mediod;

Randomly select a non-medoid object, Orandom;

Compute the total cost, *S*, of swapping *Oj* with Orandom;

If *S* < 0 then swap *Oj* with Orandom to form new set of *K* mediods;

**Until no change.**

### 3.4 Experimental Set-Up

In this study, we have used randomly generated data set and Pocker hand data set [22] with the selected algorithms; *k*-means [14], max-min [15] and *k*-medoid [14]. Random data set is generated to have 10 million records with the dimensionality of 1026. Pokers hand set data has 1 million records with ten attributes (dimensions). It is in ASCII format with comma separated values, which is converted to binary format before applying our algorithms. It is reported widely that the formatted I/O operations to consume more time than unformatted I/O [8]. In order to have further support, we had executed our algorithms with fgetc() and fread() and CPU time requirements are observed. We found fread() based implementations taking less CPU time (Table 1) than fgetc() based ones. Thus, we have carried out all our experiments with the converted binary data.

**Table 1**
**Comparison of fgetc() and fread() Performance with *k*-means Algorithm and Poker Hand Set Data under Linux Environment. *k*-means.**

*Comparison of fgetc() and fread() Performance with k-means Algorithm and Poker Hand Set Data under Linux Environment*

| *No of Records* | *Time Taken To Read Records Using* ***fgetc()*** | *k-means Alg Time Clock Ticks* | *Total Clock Ticks* | *Time Taken To Read Records Using* ***fread()*** | *k-means Alg Time* | *Total in Clock Ticks* | *Benefit of fread() Clock Ticks* |
|---|---|---|---|---|---|---|---|
| 1.0 Million | 1610000 | 630000 | 2240000 | 140000 | 630000 | 770000 | 1610000 |
| 0.9 Million | 1470000 | 560000 | 2030000 | 120000 | 570000 | 690000 | 1460000 |
| 0.8 Million | 1240000 | 510000 | 1750000 | 110000 | 500000 | 610000 | 1250000 |
| 0.7 Million | 1130000 | 420000 | 1550000 | 90000 | 440000 | 530000 | 1110000 |
| 0.6 Million | 1030000 | 380000 | 1410000 | 70000 | 370000 | 440000 | 1040000 |
| 0.5 Million | 790000 | 310000 | 1100000 | 60000 | 310000 | 370000 | 790000 |
| 0.4 Million | 630000 | 240000 | 870000 | 50000 | 250000 | 300000 | 620000 |
| 0.3 Million | 490000 | 180000 | 670000 | 40000 | 190000 | 230000 | 480000 |

The *k*-means and max-min algorithms are tested with file size of 2 GB. Computational time requirements of these algorithms with fread() and mmap() functions is observed with both the data sets under various conditions. Intel Pentium dual core 2.80 GHz processor with 1 GB RAM, 1 MB Cache memory is used in our study. Fedora 3 Linux equipped with GNU C++ gcc version 4.1.0 20060304 (Red hat 4.1.0-3) and Windows$^{XP}$ with VC++$^{2}$ in Cygwin gcc version 3.4.4-3, environment is installed on this machine with dual booting option to study the performance of the mmap() under same HW setup. We have carried out experiments with Cygwin under Windows$^{XP}$ to see the performance of mmap() function supported by GNU C compiler family in relation to Linux environment and the CPU time is measured in clock ticks.

In order to see the performance of mmap() over fread() with varying dimensionality, we have carried out these experiments. From Figure 3 to 9 demonstrate our observations. We could observe that mmap() is consistently taking less time than fread() independent of the dimensionality.

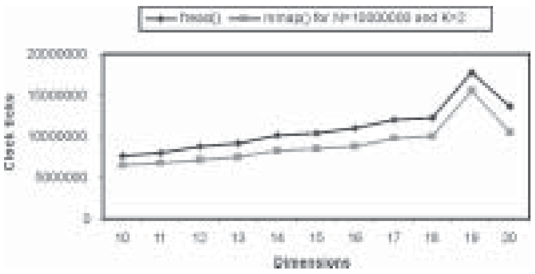

**Figure 3: *k*-means Algorithm with Random Data for 10 Million Records and also Clusters 2 under Linux.**

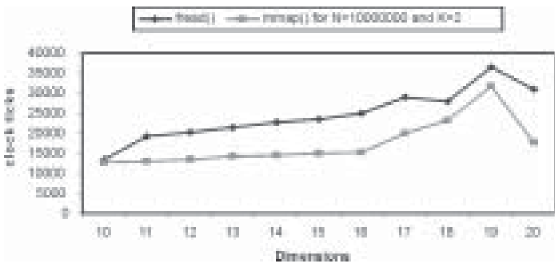

**Figure 4: *k*-means Algorithm with Random Data for 10 Million Records and also Clusters 2 under Windows$^{XP}$.**

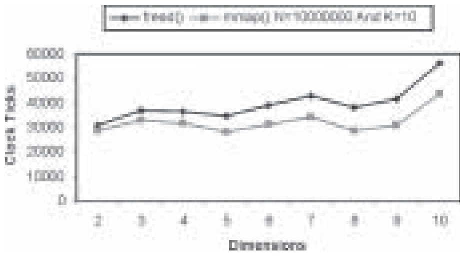

**Figure 5: *k*-means Algorithm with Random Data for 10 Million Records and also Clusters 10 under Cygwin.**

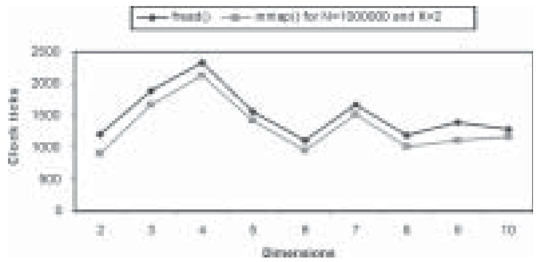

**Figure 6: *k*-means Algorithm with Pokers Hand Set Data for 1 Million Records and also Clusters 2 under Windows$^{XP}$.**

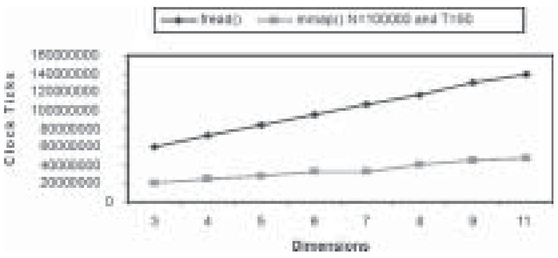

**Figure 7: max-min Algorithm with Pokers Hand Set Data for 0.1 Million Records and also Threshold 50 under Linux.**

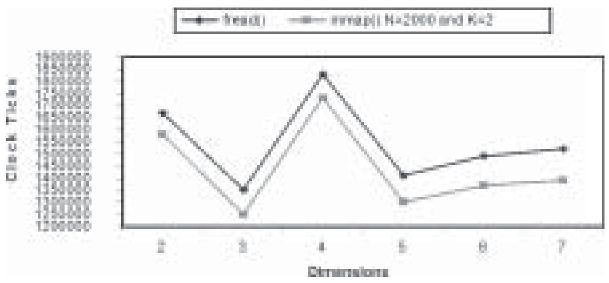

**Figure 8: *k*-medoid Algorithm with Pokers Hand Set Data for 0.002 Million Records and also Clusters 2 under Linux.**

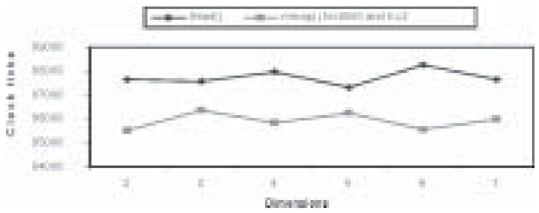

**Figure 9: *k*-medoid Algorithm with Pokers Hand Set Data for 0.009 Million Records and also Clusters 2 under Windows$^{XP}$.**

To see the benefit of mmap() over fread() with varying number of samples, we have conducted experiments, which were presented from Figure 10 to 15. We could observe that the advantage of mmap() over fread() with all our experiments on all the selected algorithms.

Experiments are carried out to see the performance of mmap() with varying number of clusters. The observations from Figure 16 to 18 indicate that the mmap() is showing its benefit over fread() independent of varying number samples.

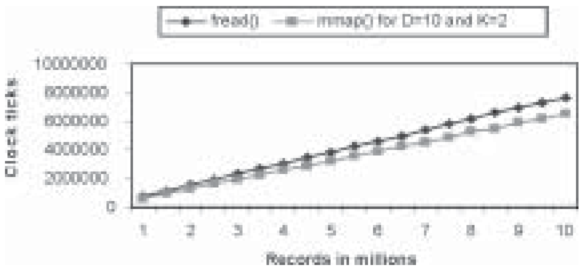

**Figure 10: *k*-means Algorithm with Random Data for Dimensionality 10 and also Clusters 2 under Linux.**

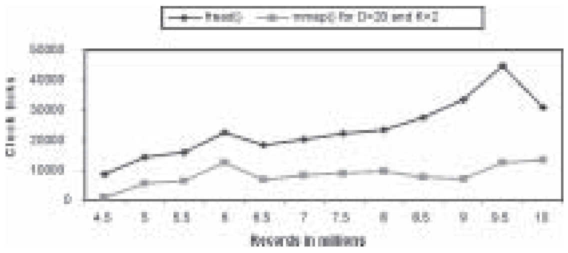

**Figure 11: *k*-means Algorithm with Random Data for Dimensionality 20 and also Clusters 2 under Windows$^{XP}$.**

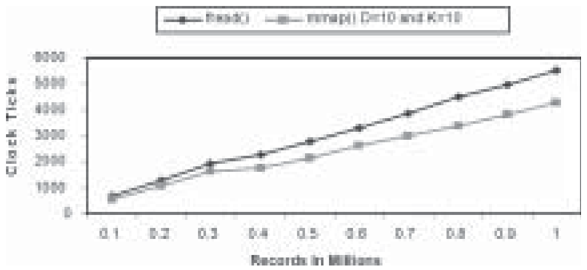

**Figure 12: *k*-means Algorithm with Pokers Hand Set Data for Dimensionality 10 and also Clusters 10 under Cygwin.**

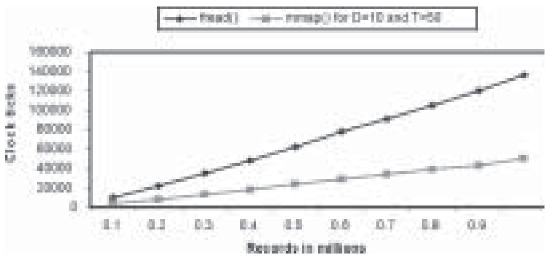

**Figure 13: max-min Algorithm with Pokers Hand Set Data for Dimensionality 10 and also Threshold 50 under Windows$^{XP}$ .**

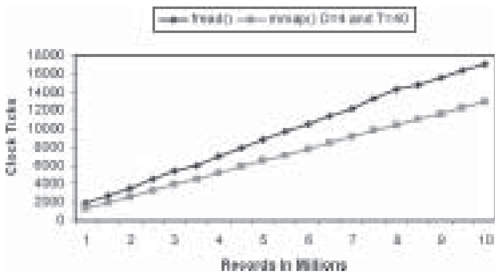

**Figure 14: max-min Algorithm with Random Data for Dimensionality 4 and also Threshold 40 under Cygwin.**

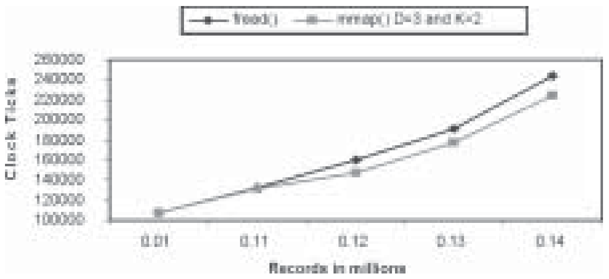

**Figure 15: *k*-medoid Algorithm with Pokers Hand Set Data for Dimensionality 3 and also Clusters 2 under Windows$^{XP}$.**

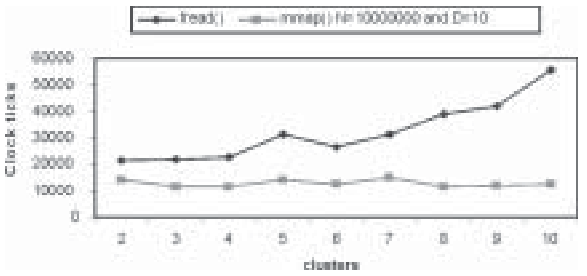

**Figure 16: *k*-means Algorithm with Random Data for 10 Million Records and also Dimensionality 10 under Cygwin.**

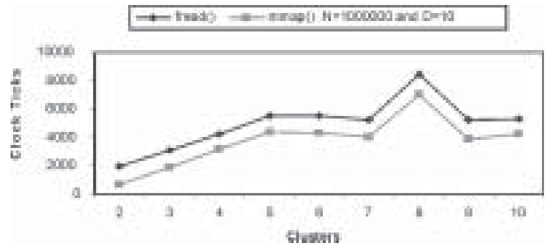

**Figure 17: *k*-means Algorithm with Pokers Hand Set Data for 1 Million Records and also Dimensionality 10 under Cygwin.**

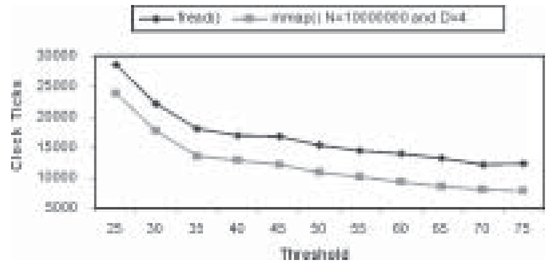

**Figure 18: max-min Algorithm with Random Data for 10 Million Records and also Dimensionality 4 under Cygwin.**

## 3.5 CONCLUSION

The memory mapping concept supported by operating systems is studied in comparison to commonly used fread() based I/O with clustering algorithms *k*-means, max-min and *k*-medoid. Experiments show that memory mapping of files gives better results than fread() based implementation under all selected operating systems. Also, the computational benefit of mmap() over fread() based algorithms is independent of number of dimensions, number of samples and number of clusters.

### 3.6 ACKNOWLEDGEMENT

We extend our heartfelt thanks for selecting this article for the, "IADIS, International Conference on Applied Computing", ALGARVE, Portugal, 455-459, April 2008.

## 4. REFERENCES

[1] Advances in Web-Age Information Management: 4th International Conference, *www.books.google.com/books.*

[2] A DM Tool Clusta, *http://www.clusta.com.*

[3] A DM Tool Cluster, *http://rana.lbl.gov/EisenSoftware Source.htm.*

[4] A DM Tool Cluster-1.37, *http://bonsai.ims.u-tokyo.ac.jp.*

[5] A DM Tool Rosetta, *http://rosetta.lcb.uu.se.*

[6] Gray, A. and Moore, A. (2004), *"Data Structures for Fast Statistics"*, Tutorial Presented at the International Conference on Machine Learning, Banff, Alberta, Canada.

[7] "A UNIX Interface for Shared Memory and Memory Mapped Files under Mach", *www.72.14.235.104.*

[8] Brain W. Keringhan and Dennis M. Ritchie (2004), *"C Programming Language"*, PHI Publications, New Delhi, India.

[9] ECE 222 System Programming Concepts Lecturer Notes on System Calls, *www.parl.clemson.edu.*

[10] FASTLab, A DM Tool, *www.FASTLABinc.com.*

[11] fread/ifstream, read/mmap Performance Results *www.lastmind.net.*

[12] Gregery Bueherg (2006) *et. al.*, "Towards Data Mining An Enlights, Architectures", Proceedings of SIAM International Conference on *Data Mining*, (20-22 April), Bethesda, MD, USA.

[13] Jaya Prakash Pisharath, Josephlu Zambreno, Berkin Oziskylmaz, and Alok Choudhaly (2006), "Accelarating Data Mining Workloads: Current Approaches and Future Challenges in System Architecture Design", Proceedings of SIAM International Conference on *Data Mining*, (20-22 April), Bethesda, MD, USA.

[14] Jiawei Han and Micheline Kamber (2006), *"Data Mining Concepts and Techniques"*, 2nd edition Morgan Kaufmann Publishers, San Francisco.

[15] T. Tou, R. C. Gonzales (1974), "*Pattern Recognition Principles*", Addison-Wesley Publishers.

[16] Killen Stoffel and Abdelkades Belkoniene, *Parllel* (1999), "*k*-means Clustering for Large Data Sets", Proceedings of Europas.

[17] Maurice J. Bach (2004), "*The Design of Unix Operating System*", PHI Publications, New Delhi, India.

[18] N. B. Venkateswarlu (2005), "Advanced UNIX Programming", B. S. Publications, Hyderabad, 1st edition, (2005).

[19] N. B. Venkateswarlu, M. B. Al-Daoud and S. A. Raberts (1995), "*Fast k-means Clustering Algorithms*", University of Leads School of Computer Studies Research Report Series Report 95.18.

[20] "Optimized Performance Analysis of Apache-1.0.5 Server", *www.isi.edu.*

[21] Paolo Palmerini (2001), "Design of Efficient Input/Output Intensive Data Mining Application", ERCIM NEWS, 44.

[22] Rabert Cattral and Franz Oppacher Carleton University, Department of Computer Science Intelligent Systems Research Unit, Canada, *http://archive.ics.uci.edu/ml/datasets/Poker+Hand*.

[23] Tuba Islam (2003), "An Unsupervised Approach for Automatic Language Identification", Master Thesis, Bogaziqi University, Istambul, Turkey.

[24] Uresh Vahalia (2006), "UNIX Internals *The New Frontiers*", Pearson Education, New Delhi, India, Third Impression.

[25] Weka, A DM Tool, *http://www.cs.waikato.ac.nz.*

[26] Xcluster, A DM Tool, *http://genetics.stanford.edu.*

[27] Yen-Yu Chen, Dingquing Gasu, Torsten Suel (2002), "*I/O Efficient Techniques for Computing Page Rank*", Department of Computer and Information Science, Polytechnique University, Brooklyn, Technical Report-CIS-(2002-03).