Performance evaluation of apriori with memory mapped files

Anuradha.T[1], Dr.Satya Pasad.R[2]and Dr.Tirumalarao.S.N[3]

[1]Department of ECM,KL University
Guntur,A.P.,India

[2] Department of CS &Engineering, Acharya Nagarjuna University
Guntur,A.P.,India

[3] Department of CSE,Narasaraopeta Engineering college
Guntur,A.P.,India

Abstract

The concept of memory mapped files reduces the I/O data movement by mapping file data directly to the process address space. This is best suitable for the data mining applications which involve accessing large data files. The recent improvement in parallel processor architectures is the multi-core architectures. To get the real benefit from these architectures we have to redesign the existing serial algorithms so that they can be parallelized on multi-core architectures. OpenMP is an API for parallel programming which make a serial program to run in parallel without much redesigning job. Our main concern in this paper is to evaluate the performance of apriori using linux mmap() function compared to fread() function in both the serial and parallel environments. Experiments are conducted with both simulated and standard datasets on multi-core architectures using openMP threads. Our experiments show that mmap() function gives better results than fread() function with both serial as well as parallel implementations of apriori on dual core.

Keywords: *apriori, fread(), mmap(), multi-core, OpenMP*

1. Introduction

The concept of memory mapping of files was introduced to reduce the overhead of file management. Mmap() function is a unix/linux function which simplifies processing of file data (1).The applications which need huge input/output overhead like network or other applications are using mmap()(2).Many unix/linux functions like grep, fgrep, egrep and the unix pipe facility use memory mapping concept for large data files. Avadis Tevanian describes an approach of file mapping facility under Mach operating system and mentions that useful performance gains can be achieved by using Mach's memory mapping(1). John Heidemann explains that CPU utilization can be reduced by using memory mapped files instead of stdio when sending large files(3).Joseph Jang in his blog has clearly shown the better performance results of mmap() over fread() and iostream(4).

A multi-core processor contains two or more actual processors integrated on the same chip and the performance gains from multi-core processors can be obtained based on the software that can run on multiple cores simultaneously(5).OpenMP is an application program interface for developing shared memory parallel programming(6). It works on the concept of multithreading. Master thread works sequentially and when the parallel region encounters, master thread forks child threads and work along with them(7). In our previous papers we have evaluated the performance of the popular data mining algorithm apriori on dual core with OpenMP threads compared to the serial implementation (8,9).Our present paper mainly concentrates on comparing the benefit of mmap() over fread() in the implementation of the apriori algorithm. The results will compare the performance of mmap() over fread() in serial and parallel implementations of apriori with different datasets at different support counts.

2. Related Work

Apriori is the popular algorithm for achieving the important functionality of data mining known as frequent

itemset mining(FIM) or association rule mining(ARM) (10).As data mining deals with large volumes of data, scalability can be achieved by parallelizing the algorithm. (11,12,13)M.J.Zaki has presented a survey paper on parallel and distributed association rule mining.(12) Rakesh Agrawal and John C.Shafer proposed two parallel algorithms known as count distribution and data distribution based on apriori.(11). Zaıane et al has proposed a parallel algorithm for finding frequent item-sets using FP-growth algorithm (13).Pattern mining researchers are also designing parallel algorithms on the recent multi core architectures.(14) Li Liu2, et.al., proposed a cache conscious FP array mechanism for implementing FP-growth algorithm on multi core processors.(15).S.Tatikonda et al., proposed frequent subtree mining from a tree structured data on multi-core architecture.(16) Research is also going on implementing data mining algorithms on multi-core architectures using openMP threads.

Anuradha et al., presented the performance evaluation of parallel apriori on dual core compared to serial execution with different data sets and also by changing the number of threads.(8,9).S.N Tirumalarao et al., studied the performance of k means clustering algorithm on multi-core architectures.(2) S.Mohanavalli et al., implemented parallel k-means algorithm with openMP and distributed algorithm with MPI . They have also compared the performance of these implementations with hybrid model which is the combination of openMP and MPI.(17)Memory mapped files concept was initially used in designing the unix based operating system internals. Avadis Tevanian et al., explains the system call for file mapping in Mach operating system.(1) Direct accessing of user programs to device memory and the files and its advantages in Linux memory management are explained in (18).But only a little research is done in finding the effect of memory mapped files compared to normal file reading operation on data mining algorithms. S.N.Tirumalarao et al.,studied the performance of memory mapped files on k-means clustering algorithm.(2)

3. Theoretical background

3.1 Apriori

Apriori is the popular algorithm for finding frequent itemsets from a transactional database. It was proposed by Agrawal and Srikant(10,19).It consists of two functions :

1. Finding the candidate k-itemsets: Initially, every item in the given database will be in the candidate k-itemset where k=1. For finding the next candidate k-itemsets (k=2 ,3 etc.), we have to join previous frequent k-itemset with itself. For example for finding candidate 2-itemset, we have to join frequent 1-itemset with itself.
2. Finding the frequent k-itemset: For finding frequent k-itemset, we have to find the count of each item in the candidate k-itemset. If the count of the item is more than a pre-specified threshold called minimum support count, it will be placed in the frequent k-itemset.

The major principle of apriori is that all the subsets of a frequent itemset should also be frequent.(10)

3.2 OpenMP

OpenMP is an API for shared memory parallel programming for C,C++ and Fortran. It is very easy to port it on different shared memory architectures.(20) OpenMP has different pragmas which direct the compiler to use the openMP constructs. If the compiler does not support openMP, the program will run sequentially. (6,21,22) To use the openMP constructs in a c program we have to include omp.h header file in our program. The parallelism in openMP is achieved by multiple threads. By using the fork-join model it makes the program to run in serial and parallel modes. Initially master thread runs the program in serial mode and when #pragma omp parallel construct is encountered, the master thread fork the child threads and runs the program in parallel mode along with child threads. Once the parallel part is over, again the child threads join the master thread and the program runs in the serial mode. The number of threads will be decided by the omp-set_num_threads () library function .

3.3 Mmap()

Mmap() function is useful when the process need to access the data from a large file. In fread() function, data must be first copied to the user space buffer before it is being copied to the process address space. Mmap() function avoid this extra copy operation as the file is directly mapped to the address space of a process. (23,2).In the mmap() function, we have to specify the starting address in the process address space from where the file should be mapped and how many bytes of the file we have to map starting from the offset.

4. Mapping Of Apriori On Dualcore

The parallelization of apriori is done based on the count distribution algorithm proposed by agrawal and shafer.(11) Here we follow the partitioning concept and

data parallel strategy. The transactional database is partitioned into number of parts equal to the number of threads.

4.1 Algorithm for parallelizing on dual core with 2 threads:

Input: Transactional database,TDB with transactions $TR_1, TR_2, \ldots, TR_n$
where a transaction TR_i is the random combination of any items from $item_1$ to $item_{10}$
and n is the number of records in the database,
Minimum support count, msc
Output:frequent k-itemsets where k=1,2,3 etc.
Step1 : k=1
candidate k-itemset ={$item_1$, $item_2$, $item_3$, $item_4$, $item_5$, $item_6$, $item_7$, $item_8$, $item_9$, $item_{10}$ }.
Step2: SET_ OMP_NUM_THREADS =2
partition the given database into three partitions.

```
#pragma omp parallel
/* begin parallel region
#pragma omp sections
{
 omp section
{
   Find the local count of each item in the candidate k-
   itemset in partition1
}
omp section
{
    Find the local count of each item in the candidate k-
    itemset in partition2
 }
}
 /* end of parallel region */
```

Global count of each item in candidate k-itemset=sum of the local counts
If the global count of any item is>msc, place the item in frequent k-itemset
Step 3:
K=k+1
Join frequent k-itemset with itself to find the next candidate k-itemset
Step4:Repeat steps 2 and 3 until any subset of candidate k-itemset is not frequent.

In the above algorithm, The #pragma omp parallel construct directs the compiler to enter into the parallel region. There are two types of work sharing constructs in openMP to divide the work parallel- Loops and Sections. We are using sections construct. Here we are creating two threads and each thread will work on each section. Because we are using multi-core processors each thread will run on separate cores there by making the execution faster compared to serial execution. For running the algorithm with three threads, we divide the database into 3 partitions and set number of threads equal to 3 and for four threads, we divide the database into 4 partitions and set number of threads equal to 4. When the number of threads are more than the number of cores , the threads will share the cores.

5. Experimental Work

The experimentation is carried out on Intel Pentium Dual-core with processor speed 1.6GHz and 3GB RAM . To get openMP compatibility, we have used Fedora 9 Linux (Kernel 2.6.25-14, Red Hat nash version 6.0.52) equipped with GNU C++(*gcc* version 4.3) for our experimentation. Different randomly generated transactional datasets with 2,4,6,8 and 10lakh records are used. Each dataset consists of any random combination of items from $item_1$ to $item_{10}$. Our algorithm is also tested with the standard accident dataset (24) from UCI repository. We have used all 3,40,183 transactions and 1 to 10 items of the accident dataset for testing purpose. The experimentation is done at different support counts. To test the effect of memory mapped files on apriori, the algorithm is run in serial mode by taking different datasets and different support counts separately with fread() and mmap() functions.The real time, user time and system time results of fread() versus mmap() are also compared by running the program parallelly on dual core processor using OpenMP threads by setting number of threads=2,3 and 4. The speed up of parallel apriori is compared with fread() and mmap() functions. The %of mmap benefit of the parallel implementations of apriori are compared by changing the number of threads .

6. Experimental Results

The following notations are used in the tables and graphs in the paper.
nrl- number of records in lakhs
SAF –serial apriori with fread()
SAM-serial apriori with mmap()
pmsc-percentage of minimum support count
PAFD-parallel apriori with fread() on dual core
PAMD-parallel apriori with mmap() on dual core
Prmb-percentage of real time mmap() benefit
Pumb-percentage of user time mmap() benefit
Psmb- percentage of system time mmap() benefit

The following results are observed from the experiments:

1. Percentage of real time mmap() benefit, Prmb values of SAM are compared to PAM by changing pmsc and keeping nrl constant for

different random data sets. Prmb values of PAM are more compared to SAM for all data sets.(Fig.6, Table.1) Prmb values of PAM are also increased compared to SAM for accident dataset at different pmsc values.(Table.3).Prmb values of PAM and SAM are also compared at each support count by changing nrl values. Prmb values are more for PAM compared to SAM at all support counts.(Fig.7,Table.4) Scalability of PAM is more compared to PAF for different datasets at different support counts(Fig.8, Fig.9, Table.2, Tabe.5).

2. Percentage of user time mmap() benefit, Pumb values and system time mmap() benefit psmb values of SAM are also compared to PAM for different data sets at different support counts. PAM gives more benefit compared to SAM in all the cases.(fig.10-13,Table.6-11)

6.1 Observations of serial vs parallel apriori with mmap()

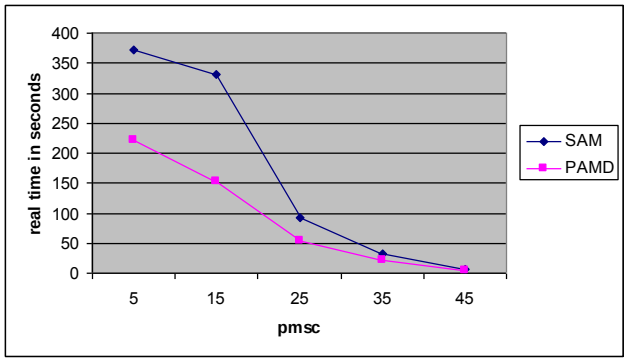

Fig. 1: SAM vs PAMD real time values for 8 lakh data

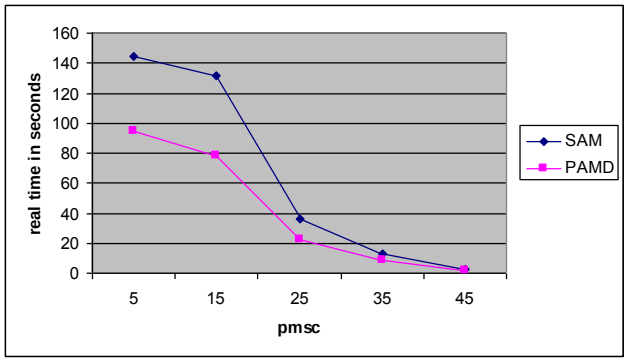

Fig. 2: SAM vs PAMD real time values for accident data

As the percentage of real time mmap() benefit , prmb is more with 3threads compared to 2 and 4 threads in most of the cases, all the PAF and PAM values in graphs and tables indicated in this paper correspond to the values obtained by parallelizing apriori on dual core with three threads.

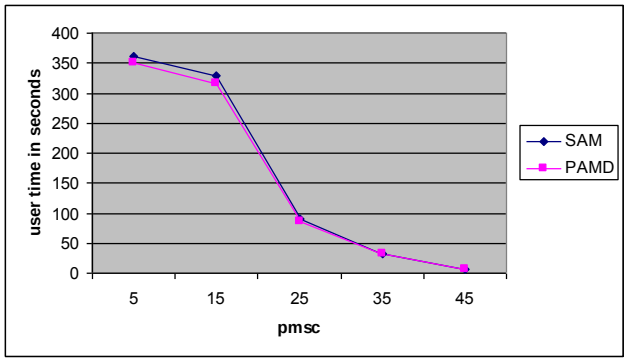

Fig. 3: SAM vs PAMD user time values for 8 lakh data

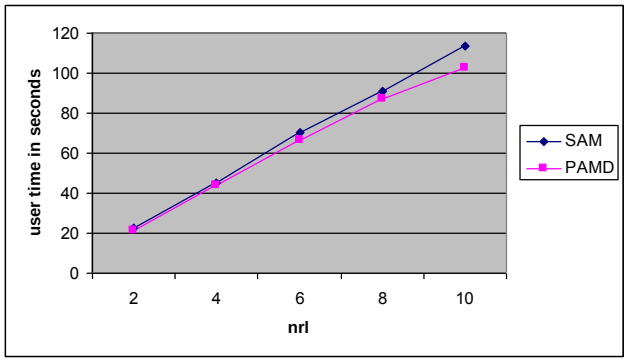

Fig. 4: SAM vs PAMD user time values at pmsc=25

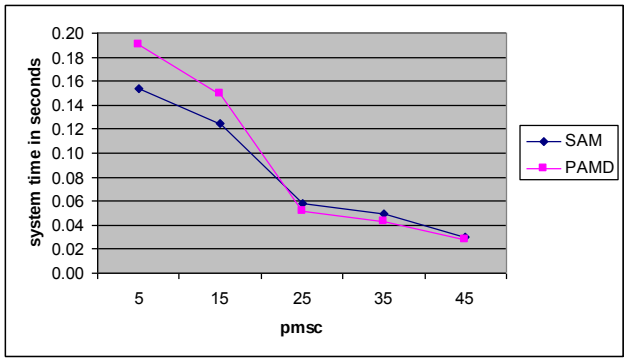

Fig. 5: SAM vs PAMD system time values for accident data

6.2 Observations of real time mmap benefit for serial vs parallel apriori with mmap()

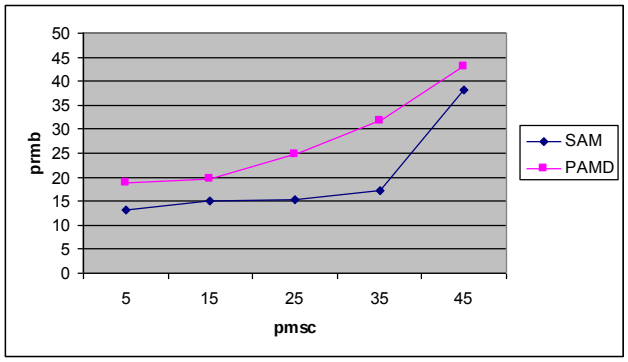

Fig. 6: SAM vs PAMD real time mmap benefit for 8 lakh data

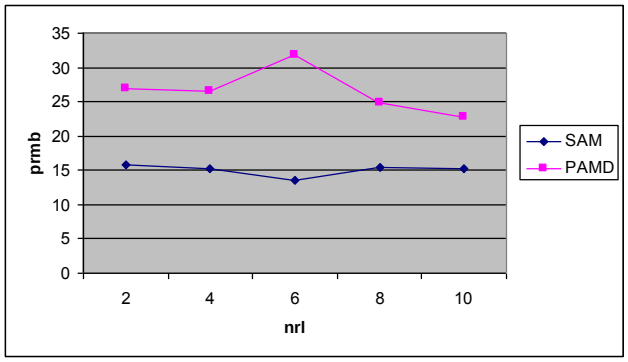

Fig. 7: SAM vs PAMD real time mmap benefit at pmsc=25

6.3 Observations of real time speed up for parallel fread() Vs parallel mmap()

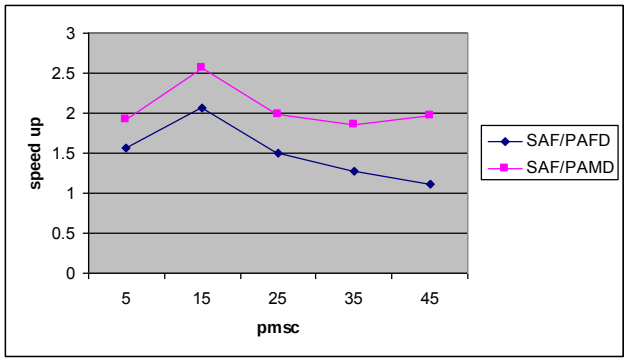

Fig. 8 comparison of speed up of PAFD vs PAMDfor 8 lakh data

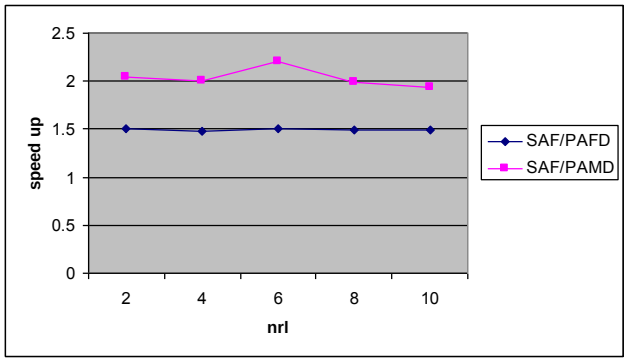

Fig. 9 comparison of speed up of PAFD vs PAMD at pmsc=25%

6.4 Observations of user time mmap benefit for serial vs parallel apriori with mmap()

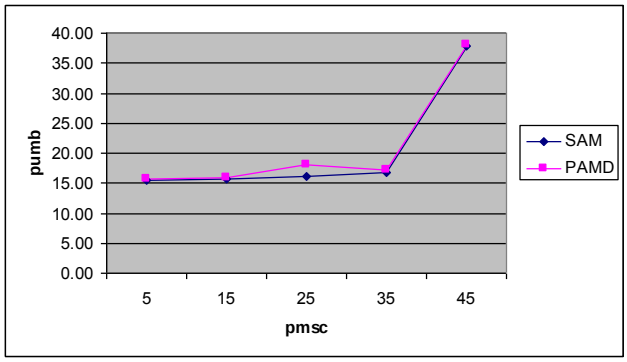

Fig. 10: SAM vs PAMD user time mmap benefit for 8 lakh data

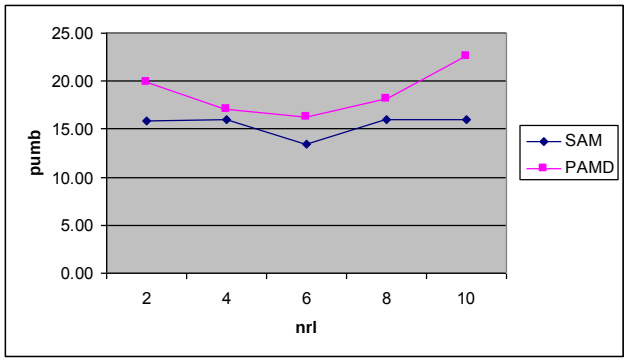

Fig. 11: SAM vs PAMD user time mmap benefit at pmsc=25

6.4 Observations of system time mmap benefit for serial vs parallel apriori with mmap()

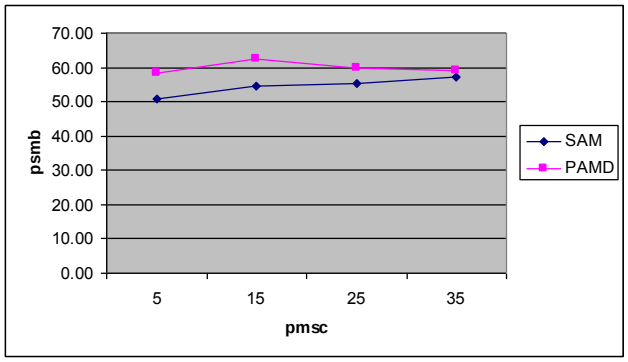

Fig. 12: SAM vs PAMD system time mmap benefit for 8 lakh data

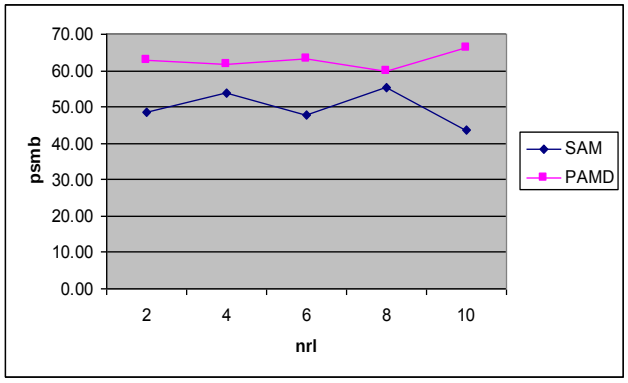

Fig. 13: SAM vs PAMD system time mmap benefit at pmsc=25

Table 1: Real time values for random data with nrl=10

Pmsc	SAF	SAM	pmb	PAFD	PAMD	pmb
5	537.54	455.95	15.18	342.01	263.65	22.91
15	487.97	418.32	14.27	236.5	184.77	21.87
25	135.77	114.98	15.31	90.9	70.13	22.85
35	49.5	40.15	18.89	38.6	26.25	32.00
45	11.4	7.05	38.16	10.2	5.34	47.65

Table 2: Real time speed up val ues for random data with nrl=10

pmsc	SAF	PAFD	PAMD	SAF/PAFD	SAF/PAMD
5	537.54	342.01	263.65	1.57	2.04
15	487.97	236.5	184.77	2.06	2.64
25	135.77	90.9	70.13	1.49	1.94
35	49.5	38.6	26.25	1.28	1.89
45	11.4	10.2	5.34	1.12	2.13

Table 3: Real time values for accident data

Pmsc	SAF	SAM	pmb	PAFD	PAMD	pmb
5	169.12	144.1	14.79	118.32	94.27	20.33
15	153.39	131.33	14.38	98.52	78.52	20.30
25	43.25	36.22	16.25	29.5	22.44	23.94
35	15.87	13.12	17.33	12.2	8.59	29.6
45	3.6	2.2	38.89	3.24	1.80	44.56

Table 4: Real time values for random data with pmsc=15

nrl	SAF	SAM	pmb	PAFD	PAMD	pmb
2	98.1	83.53	14.85	47.4	36.36	23.30
4	194.65	166.49	14.47	95.08	74.46	21.69
6	293.55	250.7	14.60	141.85	110.69	21.97
8	390.65	332.13	14.98	189.69	152.44	19.64
10	487.97	418.32	14.27	236.5	184.77	21.87

Table 5: Real time speed up values for random data with pmsc=15

nrl	SAF	PAFD	PAMD	SAF/PAFD	SAF/PAMD
2	98.10	47.40	36.36	2.07	2.70
4	194.65	95.08	74.46	2.05	2.61
6	293.55	141.85	110.69	2.07	2.65
8	390.65	189.69	152.44	2.06	2.56
10	487.97	236.50	184.77	2.06	2.64

Table 6: user time values for random data with nrl=10

pmsc	SAF	SAM	pmb	PAFD	PAMD	pmb
5	536.58	453.71	15.44	520.52	430.14	17.36
15	486.38	416.16	14.44	469.79	397.01	15.49
25	135.39	113.81	15.94	132.74	102.69	22.64
35	48.98	40.09	18.15	49.02	39.07	20.30
45	11.22	7	37.61	11.20	6.82	39.11

Table 7: user time values for accident data

pmsc	SAF	SAM	pmb	PAFD	PAMD	pmb
5	168.37	143.95	14.50	168.24	143.73	14.57
15	152.94	131.14	14.25	152.78	130.37	14.67
25	43.08	36.21	15.95	43.18	36.13	16.33
35	15.46	12.92	16.43	15.51	12.86	17.09
45	3.53	2.38	32.58	3.58	2.37	33.80

Table 8: user time values for random data with pmsc=15

nrl	SAF	SAM	pmb	PAFD	PAMD	pmb
2	97.66	83.33	14.67	94.04	79.74	15.21
4	193.99	166.12	14.37	187.70	160.64	14.42
6	292.63	249.4	14.77	282.06	237.36	15.85
8	389.55	328.59	15.65	376.21	316.68	15.82
10	486.38	416.16	14.44	469.79	397.01	15.49

Table 9: system time values for random data with nrl=10

pmsc	SAF	SAM	pmb	PAFD	PAMD	pmb
5	0.82	0.38	53.66	1.1	0.41	62.73
15	0.7	0.29	58.57	0.95	0.34	64.21
25	0.32	0.18	43.75	0.58	0.195	66.38
35	0.25	0.11	56.00	0.27	0.113	58.15
45	0.17	0.08	52.94	0.18	0.081	55.00

Table 10: system time values for accident data

pmsc	SAF	SAM	pmb	PAFD	PAMD	pmb
5	0.30	0.15	49.17	0.45	0.19	57.96
15	0.25	0.13	49.60	0.38	0.15	60.53
25	0.14	0.06	58.27	0.14	0.05	62.32
35	0.10	0.05	52.88	0.10	0.04	55.67
45	0.06	0.03	49.15	0.06	0.03	52.54

Table 11: system time values for random data with pmsc=15

nrl	SAF	SAM	pmb	PAFD	PAMD	pmb
2	0.17	0.08	52.94	0.19	0.08	57.89
4	0.28	0.14	49.29	0.43	0.16	62.79
6	0.53	0.22	59.43	0.55	0.2	63.64
8	0.57	0.26	54.74	0.77	0.29	62.34
10	0.7	0.29	58.57	0.95	0.34	64.21

7. Conclusions

The performance of apriori with memory mapped files concept compared to standard fread() function for reading data from the transactional database is identified by using linux mmap() function. The mmap() function shows better performance than fread() in real time, user time and system time.The percentage of mmap benefit is more in parallel apriori compared to serial apriori .

References

[1] Tevanian, Avadis, et al. "A unix interface for shared memory and memory mapped files under mach." Dept. of Computer Science Technical Report, Carnegie Mellon University (1987).

[2] S. N. Tirumala Rao, E. V. Prasad, and N. B. Venkateswrlu, "A Critical Performance Study of Memory Mapping on Multi-core Processors: An Experiment with K-means Algorithm with Large Data Mining Data Sets", IJCA (0975-8887)2010 Volume1-No. 9.

[3]Optimized performance analysis of Apache-1.0.5 server,www.isi.edu

[4]fread/ifstream, read/mmap performance results www.lastmind.net.

[5] "Multi-core Procesor" From wikipedia ,the free encyclopedia.Available: en.wikipedia.org/wiki/Multi-core processor [Accessed: May 24,2012].

[6] OpenMP Architecture , "OpenMP C and C++ ApplicationProgramInterface", Copyright © 1997-2002 OpenMP Architecture Review Board.http://www.openmp.org/
[7]OpenMP® Programming forTMS320C66x multicore DSPs © 2011 Texas Instruments Incorporated Printed in U.S.A.
[8] Anuradha T ,Satya Prasad R, S.N Tirumalarao "Parallelizing Apriori on Dual Core using OpenMP". International Journal of Computer Applications 43(24):33-39, April 2012.
[9] Anuradha T ,Satya Prasad R,S.N. Tirumalarao "Parallelizing Apriori on Dual Core with multiple threads" International Journal of Computer Applications 50(16):9-16, July 2012.
[10] Agrawal R, Srikant R "Fast algorithms for mining association rules" In: Proceedings of the 1994 international conference on very large data bases (VLDB'94), 1994 Santiago, Chile, pp 487–499
[11] R. Agrawal and J. Shafer "Parallel mining of association rules" IEEE Trans. Knowl. Data Eng., vol. 8, pp. 962–969, Dec. 1996.
[12]M.J. Zaki 1997 "parallel and distributed association mining: A survey" IEEE Concur, vol. 7, pp. 14–25, Dec. 1997.
[13] O. R Zaiane,M. El-Hajj, and P. Lu "Fast parallel association rule mining without candidacy generation" in Proc. ICDM, 2001, [Online].Available: citeseer.ist.psu.edu/474 621.html, pp. 665–668.
[14] Laurent, Anne, et al. "Pgp-mc: Towards a multicore parallel approach for mining gradual patterns." Database Systems for Advanced Applications. Springer Berlin/Heidelberg, 2010.
[15] Liu, Li, et al.2007 "Optimization of frequent itemset mining on multiple-core processor." Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007.
[16]. Shirish Tatikonda, Srinivasan Parthasarathy 2008 "Mining Tree Structured Data on Multicore Systems", VLDB '08, August 2430, 2008, Auckland, New Zealand
[17] Mohanavalli, S., S. M. Jaisakthi, and C. Aravindan.2011 "Strategies for Parallelizing KMeans Data Clustering Algorithm." Information Technology and Mobile Communication (2011): 427-430.
[18] Memory management in Linux for linux device drivers Third edition eMatter Edition Copyright © 2005 O'Reilly & Associates
[19] Jiawei Han and Micheline Kamber "Data Mining concepts and Techniques", 2nd edition 2006 Morgan Kaufmann Publishers, San Francisco.
[20] Diaz, Javier, Camelia Muñoz-Caro, and Alfonso Niño. "A Survey of Parallel Programming Models and Tools in the Multi and Many-Core Era." Parallel and Distributed Systems, IEEE Transactions on 23.8 (2012): 1369-1386.
[21] Kent Milfeld 2011 "Introduction to Programming with OpenMP" September 12th 2011, TACC
[22] Ruud van der pas "An Overview of OpenMP" NTU Talk January 14 2009
[23]Chapter12 "Shared memory Introduction" www.kohala.com/start/unpv22e/unpv22e.chap12.pdf
[24] K Geurts, G Wets, T. Brijs and K. Vanhoof, "Profiling High Frequency Accident Locations Using Association Rules", Electronic Proceedings of the 82th Annual Meeting of the Transportation Research Board, Washington, January 12-16, USA, 2003,18p.