Operating System Process Modeling: An Implementation of Association Learning Algorithms using Router Kernel Simulated Data

Adamade Peter Simon[#1], Sadiq Mobolaji Abubakar[*2], Anyama Oscar Uzoma[#3]

[#]*Department of Computer Science, University of Port Harcourt, Nigeria*

Abstract — *Large chunk of dispersed data exists in several databases and data marts, these amount of data if not properly gathered and analyzed will lead to total loss of useful knowledge. With the existence of the problem of an efficient scheduling and resource management techniques in Operating System, there is a dire need to provide a rule-based scheme to help optimize and maintain the operating system process modeling in a very efficient manner. To help improve on this issue, data mining techniques such as data extraction, cleaning and association rules have been used, Hence, this paper aims at investigating two of the most efficient learning association algorithms, FP-Growth and Apriori algorithms with the objective of helping understand the process of association learning in a network environment using router kernel data.. This is implemented using Rapid Miner tool to model the kernel data and further comparison of the two methods.*

Keywords — *FG-Growth, Apriori Algorithm, Machine Learning, Data mining, Multiprogramming.*

I. Introduction

Operating system software goes through a series of user and software based procedures, which usually begins with requirements specification, planning, designing, coding and testing. With each phase having proper design specifications. These specification ensure that high end quality control measures are taken.

With the development of supercomputer computers playing more significant roles in different parts of our daily life and with software systems getting more difficult, running several processes on a has become very necessary.

In single user systems, more than one potentially distinct program may need to be active at the same time. This gives the effect of parallel execution or running several processes on a single core of the programs, or pseudo parallelism.

[15], defined a process as the encapsulation of all information about a running program, allowing the CPU to be switched between. It usually has a program, input, output and a state.

In computing tasks, multi-processing are completed during the same period of time, this means that they are executed concurrently (in overlying time periods, new processes starting before others have ended) instead of sequentially (one completing before the next starts).

The processes share common processing resources, such as central processing units, memory and time accessories.

This provides for a basic form of multiple processing regarded as in multi programming in which several programs can run at the same time on a single processor, [18]. Since there is only one processor, there can be no true concurrent execution of different programs. The operating system executes part of one program, then part of another, and so on. To the user it appears that all programs are executing at the same time.

A. *Components of a Process*

- Text/code - executable program instructions
- Data – variables used in the program
- Stack - work area used the program
- Parameter passing to subroutines/system calls
- Process Control Block, PCB entry in Process

1) Statement of the Problems:

Improper management in an operating system can lead to the total failure of computer systems. Leading to loss of service, loss of valuable data, loss of equipment, valuable and lots more.

The summary of the problem are stated as thus:

- Ability to comprehend process pattern in order prevent failure
- Identifying possible working system multi-processing failure prone conditions.

1) Process Model

Process model introduces the concept of how processes are designed. Designs with respect to the following:

- Physical program counter
- The autonomous sequential processes
- Process pseudo parallelism

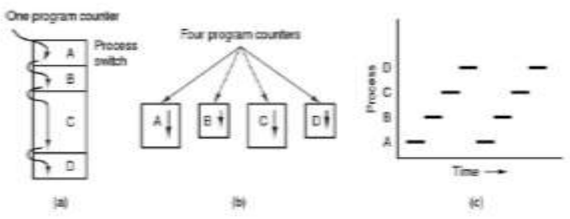


Fig 1. Process Model, [17]

C) Process Performance Optimization

Process performance is the time it takes for certain processes to be completed and this gives room for quality in the provision of service. The less time it takes to complete these processes, the better the performance of the process.

Some of the metrics used in performance include response time, throughput and scalability.

Response time is the time a user of a computer system submits a request to the time it takes for a result is produced e.g. a user clicks a button on a word processor and it carries out an action such as changing the font of an entire document. The lower the response time value, the better the performance is deemed to be. This is particularly important in interactive systems, [3].

Throughput is the number of processes that complete per unit time.

Scalability is also important. A system is said to be scalable if its performance degrades gracefully (i.e. not significantly) as the load increases, [13].

Some other important metrics, include

i average waiting time

D) *Advantages of Multiple Processes*

The advantage of multiple processes is that there are more processes competing for control of the processor(s).

In general, choose multiple processes when a significant amount of overlap between the processes can be achieved.

The application is too complex to develop as a single process, or would be too complex and difficult to maintain.

E) *Disadvantages of Multiple Processes*

High overhead time and more resource consumption by the operating system. This does not directly contribute to the processing needed by the application.

F) *Process States*

During their lifetime, processes will always exist in one of three states: running, ready, or blocked.

The state of a process is determined by the operating system, and is recorded in its process table entry.

Many factors can affect the state of a process and they are -

1) *Process States: Running*

A process in the running state is presently being executed by a CPU with all necessary resources required for execution

2) *Process States: Ready*

A process is in the ready state has all necessary resources required for execution except the CPU.

3) *Process States: Blocked*

A process in the blocked state is not capable of being executed because one or more required resources are unavailable.

G) Multithreading

In general, a computer program is a series of instructions that are executed by a CPU. Multithreaded programming takes this idea and replicates it. A multithreaded program has many sequences, each sequence within an independent thread, [12].

It is like having different small programs all running together in parallel. In case of a single processor system e.g. Intel HT processor, the operating system has a procedure called scheduler that provides the illusion that multiple threads are running in parallel, [11].

Hence for an operating system like Windows, a software application must have a multithreaded architecture in order to benefit from multiple CPUs. The main task is to divide the processing load among several threads in order to achieve maximum performance.

II. RELATED WORK

[4], wrote a report on the state of the art of CPU designs. Current CPU designs have multicore architectures, also known as Chip Multi-Processors (CMP). They consist of several cores which function like individual CPUs. Each core has a fast Level 1 cache, and can share a larger Level 2 cache with the other cores, as in the Intel Core 2 Duo CPUs. Some designs have each core with its own Level 1 and Level 2 cache, and sharing a Level 3 cache with the other cores, as in the AMD Phenom and Intel Core i7 processors. Also, a Level 2 cache could be shared by a subset of the CPU cores, as in the Intel Core 2 Quad CPU. The caches store data and instructions for access faster than fetching them from main memory, thereby feeding the fast cores quickly. Schedulers typically implement time sharing, where each thread gets a small time slice on a processor or core, and space sharing, where each job is assigned to a subset of the cores available.

[5], described the current challenges facing the parallel job scheduling community. For commodity parallel computers such as desktops and servers they identify several scheduling challenges, including variable loads (interactive applications, media applications such as video, background programs, and parallel applications that require synchronization or co-scheduling). The underlying problem is that co-scheduled processes suffer degraded performance, while collaborating processes suffer performance loss if not co-scheduled. For desktops the scheduler must also respond to changes in user priorities quickly.

[8], describes the internals of the Linux CFS (Completely Fair Scheduler). The Linux CFS scheduler keeps track of how much tasks are being treated unfairly. A task is being treated unfairly if it is not being executed by the CPU. It maintains a binary search tree (a red-black tree) that orders tasks according to how unfairly they are being treated.

A) Process Models for Parallel Programming Architectures

There are three main models for parallel programming multi-core architectures, [1]. These models are the

- Message-Passing Paradigm (MPI),
- Shared Memory Programming model and
- Partitioned Global Address Space (PGAS) programming model

1) *Message Passing Interface (MPI)*

It is the most frequently used model in representing parallel programs since it can be employed not only within a single processing node but can be used via several connected ones. MPI standard has been designed to project compactness in parallel applications, as well as to bridge the gap between the performance offered by a parallel construction and the actual performance delivered to the submission, [2]. Two critical areas regulate the overall performance level of an MPI implementation. The first area is the low level communication layer that the upper layers of an MPI implementation can use as foundations. The second area covers the communication progress and management, [2].

MPI offers several functions such as point-to-point operations, logical process topology, data exchange, gathering and reduction operations to combine partial results from parallel processes, and synchronization capabilities manifested in barrier and event operations.

The shared memory programming model permits a simpler programming of parallel applications, as the control of the data location is not required. OpenMP [6], is the most used solution for shared memory programming, as it allows an easy development of parallel applications through compiler directives. Moreover, it is becoming more important as the number of cores per system increases. However, as this model is limited to shared memory architectures, the performance is bound to the computational power of a single system, [16].

2) *Partitioned Global Address Space (PGAS) programming model*

It combines the main features of the message passing and the shared memory programming models. In PGAS languages, each thread has its own private memory space, as well as an associated shared memory region of the global address space that can be accessed by other threads, although at a higher cost than a local access. Thus, PGAS languages allow shared memory-like programming on distributed memory systems.

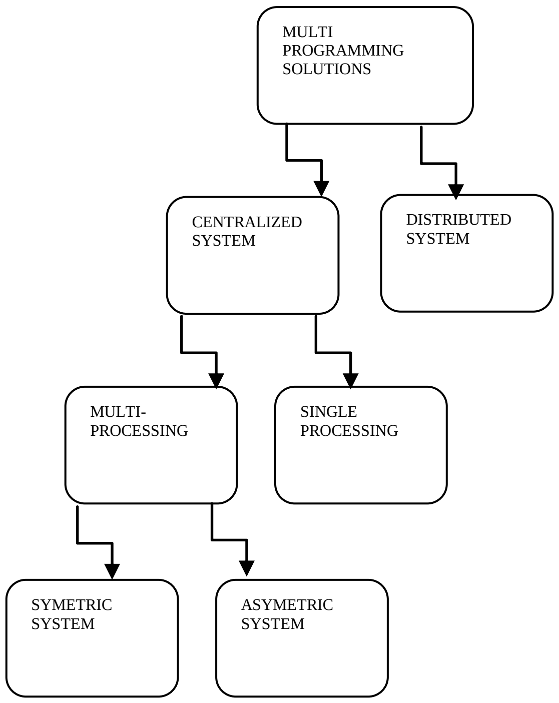


Fig 2. Multiprogramming System Framework

III. ASSOCIATION LEARNING FRAMEWORK

This will involve the use of large item sets to generate the desired of rules from processes data pool, [18]. This helps provide for optimization of the various process modules such as the context switching, scheduling and memory management. These rules help in the process management framework and it further ensures that best possible set of rules, approaches are designed and defined for the process model optimization scheme with respect to the existing scheme. In the experiment, processA, processC, processD and processAprocessB are large item sets, then we can determine if rule processAprocessB = processCprocessD holds.

For the purpose of this research work, the two algorithms will be investigated using the rapid miner modelling tool.

i APRIORI algorithm.
ii FG-Growth

APRIORI algorithm

Iteratively reduces the minimum support until it finds the required number of rules with the given minimum confidence. The algorithm has an option to mine class association rules. It is simple, fast, and very good at finding interesting rules of a specific kind in baskets or other transaction data, [14].

The algorithm works by finding frequent item sets using candidate generation. It uses Apriori property that all nonempty subsets of a frequent item set must also be frequent.

Algorithms for discovering large item sets make multiple passes over the data

In the first pass, we count the support of individual items and determine which of them are large (with minimum support).

In each subsequent pass, a seed set of item sets found to be large in the previous pass will then use this seed set for generating new potentially large item sets, called candidate item sets, and count the actual support for these candidate item sets during the pass over the data

At the end of the pass, we determine which of the candidate item sets are actually large, and they become the seed for the next pass.

This process continues until no new large item sets are found

The following steps are taken.

Step 1:

The Prune Step: To find the count of each candidate in Ck the entire database is scanned. Candidate k-item set is represented by Ck. To find whether that item set can be placed in frequent k-item set Lk to count each item set in Ck is compared with a predefined minimum support count, [7].

Step 2:

The join step: Lk is natural joined with itself to get the next candidate k+1- item set Ck+1. The major step here is the prune step which requires scanning the entire database for finding the count of each item set in every candidate k-item set. If the database size is large, so to find all the frequent item sets in the database, it requires more time (Han and Micheline, 2006).

Pseudocode

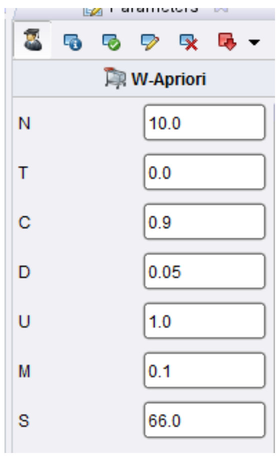
```
    Find all large 1-itemsets
    For (k = 2 ; while Lk-1 is non-empty; k++)
                     {Ck = apriori-gen(Lk-1)
           For each c in Ck, initialise c.count to zero
            For all records r in the DB
             {Cr = subset(Ck, r);  For each c in Cr , c.count++ }
              Set Lk := all c in Ck whose count >=  minsup
           }  /*  end   -- return all of the Lk sets.
```

The major step here is the prune step which requires scanning the entire database for finding the count of each Item set in every candidate k-item set. If the database size is large, so to find all the frequent item sets in the database, it requires more time.

A) Implementation of Apriori using Rapid Miner

Steps – Import datasets
Transform datasets from nominal to binomial

Metrics used –	Number of required rules
	Confidence
	Transformations
	Delta
	Upper bound
	Minimum bound
	Significance



N	10.0
T	0.0
C	0.9
D	0.05
U	1.0
M	0.1
S	66.0

Fig4. The NTCDUMS APRIORI Metric

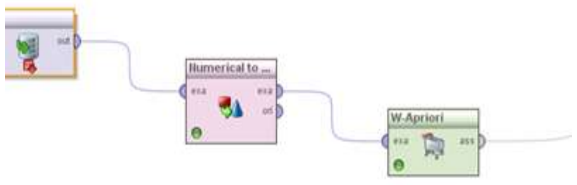
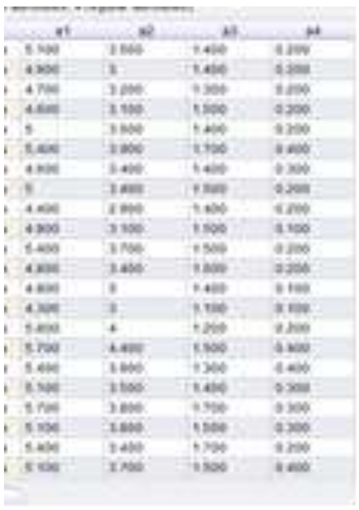


Fig5. Apriori Algorithm Model blocks

Features categories used:

- TCP/IP connection
- Traffic features
- Same host
- Protocol behaviour
- Service
- Time-based
- Content features.



a1	a2	a3	a4
5.100	3.500	1.400	0.200
4.900	3	1.400	0.200
4.700	3.200	1.300	0.200
4.600	3.100	1.500	0.200
5	3.600	1.400	0.200
5.400	3.900	1.700	0.400
4.600	3.400	1.400	0.300
5	3.400	1.500	0.200
4.400	2.900	1.400	0.200
4.900	3.100	1.500	0.100
5.400	3.700	1.500	0.200
4.800	3.400	1.600	0.200
4.800	3	1.400	0.100
4.300	3	1.100	0.100
5.800	4	1.200	0.200
5.700	4.400	1.500	0.400
5.400	3.900	1.300	0.400
5.100	3.500	1.400	0.300
5.700	3.800	1.700	0.300
5.100	3.800	1.500	0.300
5.400	3.400	1.700	0.200
5.100	3.700	1.500	0.400

Fig6. Results Apriori Algorithm

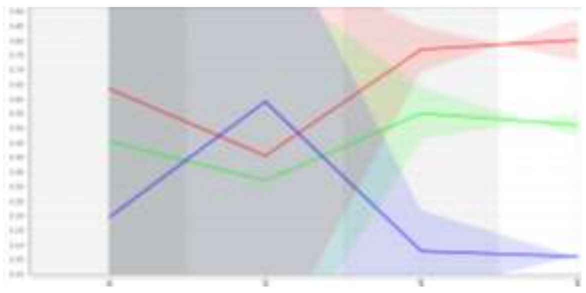


Fig7. Graph of label against attributes spread

B) FG-GROWTH

This operator efficiently calculates all frequent item sets from the given Example Set using the FP-tree data structure. It is compulsory that all attributes of the input Example Set should be binominal. In simple words, frequent item sets are groups of items that often appear together in the data. It is important to know the basics of market-basket analysis for understanding frequent item sets.

The market-basket model of data is used to describe a common form of a many-to-many relationship between two kinds of objects. On the one hand, we have items, and on the other we have baskets, also called 'transactions'. The set of items is usually represented as set of attributes. Mostly these attributes are binominal. The transactions are usually each represented as examples of the Example Set. When an attribute value is 'true' in an example; it implies that the corresponding item is present in that transaction. Each transaction consists of a set of items (an itemset).

Usually it is assumed that the number of items in a transaction is small, much smaller than the total number of items i.e. most of the examples most of the attribute values are 'false'. The number of transactions is usually assumed to be very large i.e. the number of examples in the ExampleSet is assumed to be large. The frequent-itemsets problem is that of finding sets of items that appear together in at least a threshold ratio of transactions. This threshold is defined by the 'minimum support' criteria. The support of an itemset is the number of times that itemset appears in the ExampleSet divided by the total number of examples. The 'Transactions' data set at "Samples/data/Transactions" in the repository of RapidMiner is an example of how transactions data usually look like.

The discovery of frequent itemsets is often viewed as the discovery of 'association rules', although the latter is a more complex characterization of data, whose discovery depends fundamentally on the discovery of frequent itemsets. Association rules are derived from the frequent itemsets. The FP-Growth operator finds the frequent itemsets and operators like the Create Association Rules operator uses these frequent itemsets for calculating the association rules.

Pseudo code

```
Input: D, minsupp, J ⊆ I
Output: F[J]
 F[J]={};
 forall i I ∈ occurring in D {
 F[J]=F[J]∪ {J∪ {i}};
 //Create Di;
 Di={};
 H={};
 forall j I ∈ occurring in D such that j>I
 if (support(J∪ {i,j})≥minsupp)
 H=H ∪ {j};
 forall (tid,X)∈ D with i∈ X Di = Di ∪
{(tid,X ∩ H)};
 //Depth-first recursion
 Compute F[J ∪ {i}];
 F[J]=F[J]∪ F[J ∪ {i}];
 }
```

1) Implementation of FP-Growth using Rapid Miner

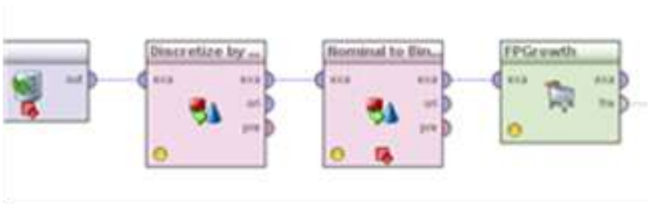
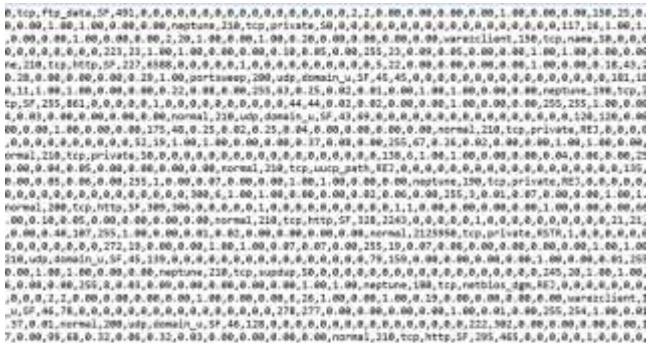


Fig 8. Model blocks FP-Growth



```
0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,150,25,0.
0,0.00,1.00,1.00,0.00,0.00,neptune,210,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,117,16,1.00,1.
,0.00,0.00,1.00,0.00,0.00,2,20,1.00,0.00,1.00,0.20,0.00,0.00,0.00,0.00,warezclient,150,tcp,name,S0,0,0,0
0,0,0,0,0,0,0,0,0,223,23,1.00,1.00,0.00,0.00,0.10,0.05,0.00,255,23,0.09,0.05,0.00,0.00,1.00,1.00,0.00,0.0
ne,210,tcp,http,SF,227,6588,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,5,22,0.00,0.00,0.00,0.00,1.00,0.00,0.18,43,2
0.28,0.00,0.00,0.00,0.29,1.00,portsweep,200,udp,domain_u,SF,45,45,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,101,10
0,11,1.00,1.00,0.00,0.00,0.22,0.08,0.00,255,67,0.25,0.02,0.01,0.00,1.00,1.00,0.00,0.00,neptune,190,tcp,1
tp,SF,255,861,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,44,44,0.02,0.02,0.00,0.00,1.00,0.00,0.00,255,255,1.00,0.0
4,0.03,0.00,0.00,0.00,0.00,normal,210,udp,domain_u,SF,43,69,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,120,120,0.0
00,0.00,1.00,0.00,0.00,175,48,0.25,0.02,0.25,0.04,0.00,0.00,0.00,0.00,normal,210,tcp,private,REJ,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,53,19,1.00,1.00,0.00,0.00,0.37,0.08,0.00,255,67,0.26,0.02,0.00,0.00,1.00,1.00,0.00,
ormal,210,tcp,private,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,138,6,1.00,1.00,0.00,0.00,0.04,0.06,0.00,25
0.00,0.04,0.05,0.00,0.00,0.00,0.00,normal,210,tcp,uucp_path,REJ,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,135,
0.00,0.05,0.06,0.00,255,1,0.00,0.07,0.00,0.00,1.00,1.00,0.00,0.00,neptune,190,tcp,private,REJ,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,300,6,1.00,1.00,0.00,0.00,0.02,0.06,0.00,255,3,0.01,0.07,0.00,0.00,1.00,1.
normal,200,tcp,http,SF,309,306,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,1,1,0.00,0.00,0.00,0.00,1.00,0.00,0.00,60
.00,0.10,0.05,0.00,0.00,0.00,0.00,normal,210,tcp,http,SF,328,2243,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,21,21,
,0.00,0.48,107,255,1.00,0.00,0.01,0.02,0.00,0.00,0.00,0.00,normal,2125950,tcp,private,RSTR,1,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,272,19,0.00,0.00,1.00,1.00,0.07,0.07,0.00,255,19,0.07,0.06,0.00,0.00,0.00,0.00,1.00,1.00
210,udp,domain_u,SF,45,139,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,79,159,0.00,0.00,0.00,0.00,1.00,0.00,0.01,255
0.00,1.00,1.00,0.00,0.00,neptune,210,tcp,supdup,S0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,245,20,1.00,1.00,
6,0.00,0.00,255,8,0.03,0.00,0.00,0.00,0.00,0.00,1.00,1.00,neptune,180,tcp,netbios_dgm,REJ,0,0,0,0,0,0,0,0,
,0,0,0,2,2,0.00,0.00,0.00,0.00,1.00,0.00,0.00,6,26,1.00,0.00,1.00,0.19,0.00,0.00,0.00,0.00,warezclient,1
_u,SF,46,78,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,278,277,0.00,0.00,0.00,0.00,1.00,0.01,0.00,255,254,1.00,0.01
.37,0.01,normal,200,udp,domain_u,SF,46,128,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,222,302,0.00,0.00,0.00,0.00,1
7,0.00,99,68,0.32,0.06,0.32,0.03,0.00,0.00,0.00,0.00,normal,310,tcp,http,SF,295,465,0,0,0,0,0,0,1,0,0,0,0,
```

Fig 9. Router Simulated Data Set used

Steps – Import datasets Discretise

Transform datasets from nominal to binomial, using a minimum support of 0.1 and maximum item of -1

2) Results obtained

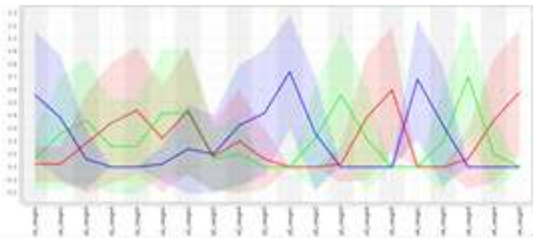


Fig 9. Deviation graph showing label against attribute range

IV. RESULT DISCUSSION

Applying both operators on the simulated data set, it was observed that the Apriori algorithm processes data in a different manner from the FPGrowth and it creates association rules by eliminating the sets of articles which are not frequent (with a minimum support smaller than the minimum support specified). There is a significant correlation between the outputs of both approaches. FP-Growth perform data scans and is therefore often applicable even on large data sets, which is totally different from the behaviour of the Aproiri algorithm.

Also, from the results, if the optimization value obtained from the FPGrowth is smaller than 0.1, then the test is 80% reliable, i.e. the null hypothesis can be rejected at a trust level of 80%.

If the value is smaller than 0.05, the test is 85% reliable, i.e. the null hypothesis can be rejected at a trust level of 85%.

Table 1. Comparison of results of both algorithms

APRIORI	**FPGROWTH**
Significant correlation yes	Significant correlation yes
Generates Frequency set yes	Generates Frequency set no
Reducing data by minimum support yes	Reducing data by minimum support no
Learns from large dataset no	Learns from large dataset yes
Candidate set generation no	Candidate set generation yes
Faster	Slower
Discards infrequent item no	Discards infrequent item yes
Less memory management	Less memory management
Varaiable order is used during variable scanning	Fixed order is used during variable scanning
Uses recursion	Uses recursion

V. CONCLUSION

The association procedures play a key role in many data mining tasks, by trying to find interesting patterns in databases. In order to obtain these association rules the frequent sets of articles must be previously generated.

The study also shows the usefulness of using both techniques in association mining. At the end of the experiment, Apriori yielded to be a better technique using same datasets and same platform. This has further shown the effectiveness of the use data mining techniques in association rule.

A) Research Highlights

The research highlights of this paper are:

- This paper investigates the effect of using FP-Growth and APRIORI Algorithms on Router Simulated Datasets
- The study also projects the better approach by comparing the results

REFERENCES

[1] I. Alaa. Parallel Performance of MPI Sorting Algorithms on Dual–Core Processor Windows-Based Systems. International Journal of Distributed and Parallel Systems. Vol.2, No.3. DOI: 10.5121/ijdps.2011.2301, 2011.

[2] D. Buntinas, G, Mercier. and W.. Gropp. Implementation and Evaluation of Shared- Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem. Journal of Parallel Computing, vol. 33, no. 9, pp. 634-644., 2011.

[3] D. Dhamdhere. (2007). Operating Systems: A Concept-Based Approach. McGraw-Hill. Retrieved: 7/05/2015: http://arxiv.org/ftp/arxiv/papers/1011/1011.1735.pdf, 2007.

[4] K. Fax´en, C. Bengtsson, M. Brorsson, G. H°akan, E. Hagersten, Jonsson B., Kessler C. Lisper B., Stenstr¨om P. and Svensson, B. (2008). Multicore computing - the state of the art. Retrieved: 22/05/2015: http://arxiv.org/ftp/arxiv/papers/1011/1011.1735.pdf

[5] E. Frachtenburg and U. Swiegelshohn. Job Scheduling Strategies for Parallel Processing. Springer. Retrieved: 22/05/2015: http://arxiv.org/ftp/arxiv/papers/1011/1011.1735.pdf

[6] Gabriel et al. Open MPI: Goals, Concept, and Design of a Next Generation MPI Iimplementation. Proceedings of 11th European PVM/MPI Users' Group Meeting, Budapest, pp. 97–104., 2004.

[7] I. Han and K. Micheline Data Mining concepts and Techniques. Morgan Kaufmann Publishers, San Francisco. 2nd ed. Retrieved: 22/05/2015: http://dl.acm.org/citation.cfm?id=355013

[8] A. Kumar. Multiprocessing with the completely fair scheduler. Technical report. IBM Developer Works. Retrieved: 22/05/2015: http://scholarworks.bgsu.edu/cgi/viewcontent.cgi?article=1000&context=ms_tech_mngmt

[9] B. Lyer and D. Dias, (2003). System Issues in Parallel Sorting for Database Systems. Proceedings of the International Conference on Data Engineering, pp. 246-255, 2003.

[10] D. Mallón, G. Taboada, C. Teijeiro, J. Touriño, A. Fraguela. A. Gómez, R. Doallo and J. Mouriño. Performance Evaluation of MPI, UPC and OpenMP on Multicore Architectures. EuroPVM/MPI LNCS 5759, pp. 174-184., 2009.

[11] S. Manu, S. Preeti and M. Vijay. Genetic Algorithm Optimal Approach for Scheduling Processes In Operating System. International Journal of Engineering Research & Technology (IJERT). Vol. 2 Issue 5, 2013.

[12] T. Martinez and S. Parikh. Understanding dual-processor, Hyper Threading Technology and Multi-Core Systems, Technical White Paper, 2005.

[13] D. Menasc´e, V. Almeida and L. Dowdy. (2004). Performance by Design. Pearson Education, Retrieved: 2/05/2015: http://arxiv.org/ftp/arxiv/papers/1011/1011.1735.pdf

[14] S. Nilesh, S. Shailendra S. and W. Shende. Implementing Apriori Algorithm in Parallel. IJCAT International Journal of Computing and Technology, Volume 1, Issue 3, April 2014. ISSN: 2348 – 6090, 2014.

[15] A. Stanley. Operating Systems lecture note. Retrieved: 2/05/2015: http://csalpha.unomaha.edu/~stanw/101/csci4500/m03_6.pdf

[16] H. Umar, S. Haroon and I. Muhammad. Parallel Implementation of 1-D Complex FFT Using Multithreading and Multi-Core Systems. International Journal of Computer and Communication Engineering, Vol. 2, No. 2., 2013.

[17 N. Tiina. Operating Systems lecture note. Retrieved: 27/06/2015: https://www.cs.helsinki.fi/u/niklande/opetus/kj/2008/lectures/os-2008-l02-handout2.pdf runtime storage management in compiler design VOL.10 No.8, 2010.

[17] S. Khasim. New Vision of the Computer Operating System. International Journal of Computer Trends and Technology (IJCTT) - volume4. Issue4. ISSN: 2231-2803. pp 739-743, 2013.

[18] A. Priyadarshni. Heterogeneous Multi Core Processors for Improving the Efficiency of Market Basket Analysis Algorithm in Data Mining. International Journal of Computer Trends and Technology (IJCTT). Volume 15, ISSN: 2231-2803, pp 16, 2014.