Approximate Frequent Itemsets Mining on Data Streams Using Hashing and Lexicographic Order in Hardware

Lázaro Bustio-Martínez
René Cumplido
Martín Letras-Luna
Claudia Feregrino Uribe
National Institute for Astrophysics, Optics and Electronic.
Luis Enrique Erro No 1,
Sta. Ma. Tonantzintla, 72840, Puebla, México.
Email: lbustio, rcumplido, mlteras, cferegrino @inaoep.mx

Raudel Hernández-León
José M. Bande-Serrano
Advanced Technologies Application Center.
21812 e/ 218 y 222,
Rpto. Siboney, Playa, C.P. 12200, Havana, Cuba.
Email: rhernandez, jbande @cenatav.co.cu

***Abstract*—Frequent Itemsets Mining is a Data Mining technique that has been employed to extract useful knowledge from datasets; and recently, from data streams. Data streams are an unbounded and infinite flow of data arriving at high rates; therefore, traditional Data Mining approaches for Frequent Itemsets Mining cannot be used straightforwardly. Finding alternatives to improve the discovery of frequent itemsets on data streams is an active research topic. This paper introduces the first hardware-based algorithm for such task. It uses the top-frequent -itemsets detection, hashing and the lexicographic order of received items. Experimental results demonstrate the viability of the proposed method.**

I. Introduction

We are in the Big Data era. In order for these big data to be useful, they must be processed to obtain their hidden knowledge. Data Mining (DM) gathers the tools needed to face such immense data volumes. One DM technique that has achieved outstanding results is Frequent Itemsets Mining (FIM).

Datasets were the most used data sources for FIM, but recently data streams are gaining attention due to the novel applications that employ them (financial analysis and data centers among others) [1]. In streams, data evolves over time and are transmitted at high speeds making impossible to store them for offline processing. This situation provokes that traditional approaches for mining datasets cannot be used straightforwardly. Therefore, finding alternatives to improve this task on data streams is an active research topic. One of such alternatives is to develop parallel algorithms that can be implemented in custom hardware architectures taking advantage of the intrinsic parallelism of such devices.

The main goal of this paper is to propose a method for frequent itemsets mining based on hashing and the lexicographic order of the received items. This approach is oriented to discover frequent itemsets in data streams composed of small transactions in large alphabets. It is valid to say that, upon to our knowledge, that this is the first algorithm proposed to solve this subproblem and the first one to use the lexicographic order of received items and hashing for the frequent itemset detection.

II. Theoretical basis

Let be a set of items and be a transactional data source: An *itemset* is a set of items over such that . A *transaction* over is a couple where is the transaction identifier, and . The *support* of an itemset is the fraction of transactions in containing . An itemset is called *frequent* if its support is greater than a given minimal support threshold *minsup*. An important characteristic of frequent sets is that they are composed of subsets that have been also frequent (Downward Closure property) [2]

A *data stream* is a continuous, unbounded and not necessarily ordered, real-time sequence of data items. In data streams, three major constraints can be verified [1]: *Continuity:* items in streams arrive continuously at a high rate; *Expiration:* items can be accessed and processed just once, and *Infinity:* the total number of data is unbounded and potentially infinite.

In data streams, a *window* is an excerpt of transactions that can be created using one of the following three approaches: *Landmark*, *Damped* or *Sliding* Window Models. This paper is focused on Sliding Window Model (SWM). In SWM, given a window of size only the latest transactions are utilized in the mining process. As new transactions arrive, the old ones in the sliding windows are excluded [1].

As it was stated before, data streams are infinite and it is unrealistic to store all the data transmitted, but it is indeed possible to store an excerpt of them. Besides, the number of itemsets obtained grows exponentially with the input data size, so the excerpts of data stream must be larger enough to be useful and at the same time shorter enough to fit in memory. A valid solution to this dilemma is using *hash functions*. In

978-1-5090-5859-4/17/$31.00 c 2017 IEEE

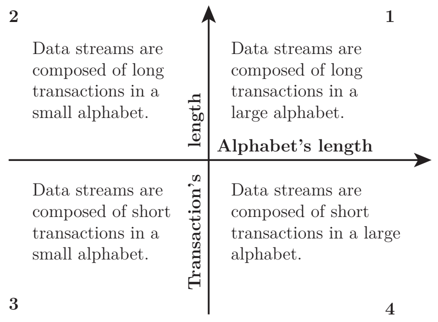
2
1
3
4
Data streams are composed of long transactions in a small alphabet.
Data streams are composed of long transactions in a large alphabet.
Data streams are composed of short transactions in a small alphabet.
Data streams are composed of short transactions in a large alphabet.
Alphabet's length
Transaction's length

Fig. 1. Different subproblems derived from the Frequent Itemsets Mining on data streams.

this paper, the hash family is used due to its advantages which are described in [3].

III. Related Work

FIM has been applied to discover frequent itemsets on datasets and recently, on data streams. Algorithms based on Apriori [2], Eclat [4] and FP-Growth [5] were the most used algorithms for FIM on datasets. For data streams, from the reviewed literature was noticed that Space-Saving [6] and systolic trees-based approaches were preferred (for space reason, the state-of-the-art section only refers the most relevant paper).

Apriori-based approach iterates several times over the data and needs to download the dataset into the hardware: this violates all the constraints of data streams [7]. Algorithms derived from Eclat represent the data using the vertical dataset representation, which avoids the itemsets generation stage from Apriori, but data must be known before processing. Once again, this violates the three constraints of data streams [8]. FP-Growth derived approaches uses a tree-like data representation which data can go through the tree without delays. But two traverses over the data are needed, one to build up the tree and other for the mining itself. Although the tree-like data structure is very well suited for data streams processing, these two traverses violate the Infinity, Continuity and Expiration constraints of data streams [9]. Also, due to the immense itemsets explosion of FIM problem, a tree-based approach can be only used in problems where alphabets are short. Space-Saving was used for FIM in data streams, but all derived papers were focused on detecting frequent 1-itemsets, which is a reduced version of general FIM [10]. In [11] authors adopt a tree-based approach using LWM and SWM. Also, a preprocessing stage, formed by a top frequent itemsets detection process performed in software), was introduced and a parallel algorithm designed to be implemented in hardware devices (such as FPGAs) was proposed.

From the review of the state-of-the-art it is derived that the frequent itemsets detection problem can be divided into four distinct subproblems as it is presented in Figure 1. This paper is oriented to solve the fourth subproblem of FIM on data streams.

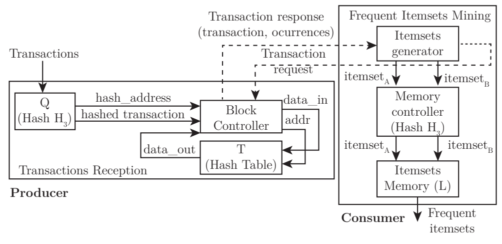

Fig. 2. Schematic of the proposed method based on hashing and lexicographic order of received items.

IV. Proposed Method

When the number of elements in is large, the number of itemsets produced is intractable, so new mining strategies according to Figure 1 should be adopted. One of these strategies is the use of the lexicographic order of received items and the detection of top frequent itemsets (which can be seen as a preprocessing stage). Using this preprocessing, all itemsets detected as infrequent are removed, this avoids the use of hardware resources to process unpromising data (supported on the Downward Closure property). Figure 2 shows a diagram of the proposed method.

Following a proper order, the tree-based data structure used in [11] can be mapped into a linear array, where nodes can be directly accessed just by knowing their positions. Let be an item in position of a transaction received from a data stream assuming a lexicographic ordering in items of . Let be a linear array of generated itemsets. Let be a FIFO queue of generated itemsets represented by their positions in . Therefore, the generated itemsets from and can be known using the following expressions: (1) , (2) and (3) where is the positions in of generated itemsets which are stored in ; and and are the current and previous items received from respectively. is *lexicographic distance* which is defined as the number of items following the lexicographic order that separates two items.

The proposed mining method is implemented using two separate parallel processes considering the Producer-Consumer design pattern. One of the proposed processes (named *Transactions Reception*) receive transactions from the processing window and store them in an intermediate buffer. The other process (named *Frequent Itemsets Mining*), simultaneously generates the itemsets using the transactions stored and obtains the frequency counting of the generated itemsets.

When a transaction is received by the *Transactions reception* process, then a hash function is evaluated.

For each transaction received:

1) If a collision does not occur, then is stored in the slot of . In consequence, the collisions field is set to and the occurrences field is set to 1.
2) If a collision occurs, then it is verified if the slot is already occupied by or by another transaction already stored.

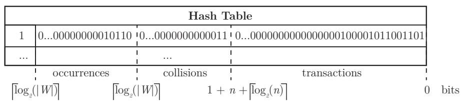

Fig. 3. A tuple in hash table .

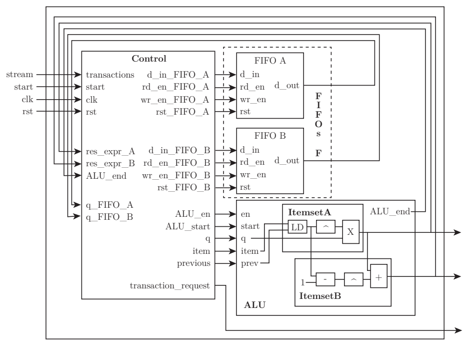

Fig. 4. Hardware-level schematic describing the *Frequent Itemsets Mining* process.

a) If the slot is already occupied by , then the occurrence counting is incremented.
b) If the slot is already occupied by a transaction , then a new hash function is randomly generated and this is repeated using (this procedure is called *rehashing*). The rehashing is repeated until an empty slot is found, or until the stop condition is reached (the maximum number of rehashing allowed is reached). If the stop condition is reached and no empty slot is found, then the collision counting for in is incremented by .

The *Frequent Itemsets Mining* process requests a transaction to the *Transactions Reception* process, and using Equations (1),(2) and (3) itemsets are generated and stored using the same hashing approach applied in *Transactions Reception*. A diagram of the *Frequent Itemsets Mining* process is represented in Figure 2.

Figure 4 shows a hardware-level schematic of the itemsets generation process. When a transaction is received, it is separated in items in the *Control* block. Each item and the previously received are used to generate itemsets, which are stored in *Itemsets Memory* and they are pushed into the *FIFOs F* block. When an item is processed, then it is needed to read and to store some values simultaneously in *FIFOs F*. The *FIFOs F* block was implemented using two FIFOs to allow simultaneous accesses. When an item is processed, then the generated value is read from *FIFO A* and the obtained itemsets' indices are stored in *FIFO B*. Following, when the item is processed, then the value is read from *FIFO B* and produced itemsets' indices are stored in *FIFO A*. For the next items, FIFOs and are cyclically switched for reading and writing. This switching process is controlled in the *Control* block. The itemsets' indices are calculated in the *ALU* block using Equations (1), (2) and (3). The hardware architecture that implements the proposed method is named *LexOrd*.

V. Evaluation

The LexOrd architecture was described using the High-Level Synthesis (HLS) with C language using the Vivado Design Suite 2015.2 and targeted for the Xilinx Zynq-7000 XC7Z020-CLG484 FPGA device. This device was selected because it allows the use of external memories by using HLS. After the LexOrd architecture was synthesized and implemented, the utilization report is showed in Table I and the maximum operating frequency obtained was 114 MHz. From the synthesis results, it was noticed that the critical resource are memories (BRAM_18). By using the 92% of total memories available, the LexOrd architecture can handle processing windows up to 1000000 transactions with 1650825 items allowing a maximum of 4 collisions. Higher processing windows can be handled by using external memories.

The throughput obtained by the Lexord architecture must be analyzed separately for the *Transactions Reception* process and the *Frequent Itemsets Generator* process. Considering the theoretical throughput is determined by
——————————— where is the maximum operating frequency obtained and are the clock cycles needed to complete the task. The maximum number of items in each transaction is determined by the available memory in the selected FPGA, and the LexOrd architecture can handle transactions up to 32 items. As the *Transactions Reception* process implements a pipeline, after six clock cycles it is considered that the processing rate is constant, so, the throughput for the this process is 4.3 Gbps.

The *Frequent Itemsets Mining* process generates itemsets from transactions composed of items (in the worst case) of bits each item. Thus, the throughput is
——————————— where . So, the throughput for the Frequent Itemsets Mining process is 2.28 Gbps.

From the review of the state-of-the-art, it was noticed that the reported results are hard to replicate due to the lack of required implementation details and the lack of precise experimental designs. Also, researches oriented to FIM are focused to discover frequent 1-itemsets instead obtain frequent k-itemsets. So, to verify the performance of the LexOrd architecture, traditional software-based algorithms for FIM were used. To do so, the SPMF Data Mining Library [12] was used. Hardware architectures reported in [11] were also used for comparison.

TABLE I
Result of synthesis and implementation of the LexOrd architecture.

Name	Avaible	Total	Utilization (%)
BRAM_18	280	259	92
DSP48E	220	2	0
FF	106400	1423	1
LUT	53200	45811	86

TABLE II
DESCRIPTION OF THE SELECTED DATASETS.

Datasets	Size (MB)	# Trans.	# Items	Ave. Trans.	Density
Connect	9.039	67557	129	43	33.33
MSNBC	4.217	989818	17	1.68	9.88
T40I10D100K	15.116	100000	942	39.6	4.20

TABLE III
PROCESSING TIME (IN SECONDS) COMPARING THE LEXORD ARCHITECTURE AGAINST THE SELECTED BASELINE ALGORITHMS FOR THE TOP FREQUENTS ITEMSETS ON SELECTED DATASET.

Datasets	Sup. Thres.	Apriori	Eclat	FP-Growth	SysTree	SysTree	LexOrd
Connect	1%	77.9380	41.7930	0.3790	0.0024	0.0037	*3.4452*
	5%	74.8530	37.7800	0.3700	0.0024	0.0037	*3.4452*
	10%	65.4130	37.7080	0.3240	0.0024	0.0037	*3.4452*
	15%	48.6870	1.5310	0.3100	0.0024	0.0037	*3.4452*
	20%	47.8200	1.3920	0.2860	0.0024	0.0037	*3.4452*
MSNBC	1%	0.8740	0.9730	0.8240	0.0044	0.0068	*0.0283*
	5%	0.8380	0.5290	0.7890	0.0044	0.0068	*0.0283*
	10%	0.5650	0.4130	0.7510	0.0044	0.0068	*0.0283*
	15%	0.5410	0.3900	0.7110	0.0044	0.0068	*0.0283*
	20%	0.4130	0.3760	0.6570	0.0044	0.0068	*0.0283*
T40I10D100K	1%	0.5680	0.1570	0.1880	0.0008	0.0012	*0.0067*
	5%	0.2310	0.1170	0.1380	0.0008	0.0012	*0.0067*
	10%	0.1880	0.0740	0.1270	0.0008	0.0012	*0.0067*
	15%	0.1690	0.0690	0.1310	0.0008	0.0012	*0.0067*
	20%	0.1000	0.0670	0.1160	0.0008	0.0012	*0.0067*

Datasets were chosen from related works, and they are used for comparing the performance of the proposed method against the state-of-the-art implementations. Details of the selected datasets are given in Table II.

Using the SPMF, several experiments were conducted using a Dell Precision T7500 workstation. This workstation is equipped with an Intel Xeon E5620 processor (2.4 GHz, 12 MB cache, 4-cores, 8-CPU model) and 8 GB of RAM. Datasets where preprocessed to obtain the top frequent itemsets and then were reduced keeping those top frequent itemsets and excluding the others. For experiments, frequents itemsets were obtained for 1%, 5%, 10%, 15% and 20% of support threshold using Apriori, FP-Growth, Eclat from [12] and SysTee from [11].

Table III shows the result obtained from the experiments conducted. Column *Sup. Thres.* contains the minimum support threshold used. The processing time (in seconds) obtained are represented under the algorithms' names. From experiments conducted it was demonstrated that the LexOrd architecture outperforms all the selected software-based baseline algorithms for several support threshold values selected. As it was expected, the processing time needed for the LexOrd architecture grows as the size of the received transactions increase, but still are competitive compared against SysTree . Although the SysTree architectures outperforms the LexOrd architecture, it must be noticed that the SysTree is oriented to discover frequent itemsets when the number of items in is low (second subproblem). The LexOrd architecture does not take care about the number of items in , but the number of items in received transactions must be short (fourth subproblem). Considering the *Density* column (which is calculated using in Table II ————), the LexOrd architecture performs better in sparse datasets than in dense datasets.

Software-based algorithms are dependent of the support threshold where the use of a too low support threshold conduces to a heavy processing time. The LexOrd architecture is insensitive to minimum support value, which it is a very valuable feature of this architecture. In LexOrd, all hardware resources needed for mining incoming data streams are available and remain invariant whether the minimum support threshold is 1% or 99% leading to obtain the same processing time.

VI. CONCLUSIONS

The LexOrd architecture was proposed to deal with such cases where the number of items in alphabets is large and transactions are short. This architecture performs one order of magnitude faster than the selected software-based baseline algorithms in software. Although the systolic tree-based approach outperforms the LexOrd architecture, it should be considered that both architectures are oriented to solve different subproblems and is still competitive. By mean of this proposed method, the fourth subproblem of FIM on data streams is solved.

REFERENCES

[1] A. Cuzzocrea, "Models and Algorithms for High-Performance Distributed Data Mining," *Jour. of Par. and Dist. Comp.*, vol. 73, no. 3, pp. 281 – 283, 2013.

[2] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," in *Proc. of the 20th Intl. Conf. on Very Large Databases*, ser. VLDB '94, 1994, pp. 487–499.

[3] J. Carter and M. N. Wegman, "Universal Classes of Hash Functions," *Journal of Computer and System Sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[4] M. J. Zaki, "Scalable Algorithms for Association Mining," *Knowl. and Data Eng., IEEE Trans. on*, vol. 12, no. 3, pp. 372–390, 2000.

[5] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns Without Candidate Generation," *SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, May 2000.

[6] A. Metwally, D. Agrawal, and A. E. Abbadi, "An Integrated Efficient Solution for Computing Frequent and Top-k Elements in Data Streams," *ACM Trans. Database Syst.*, vol. 31, no. 3, pp. 1095–1133, Sep. 2006.

[7] Z. Baker and V. Prasanna, "Efficient Hardware Data Mining With the Apriori Algorithm on FPGAs," in *Proc. of the 13th Ann. IEEE Symp. on Field-Prog. Custom Comp. Mach.*, ser. FCCM '05, 2005, pp. 3–12.

[8] Y. Zhang, F. Zhang, Z. Jin, and J. D. Bakos, "An FPGA-Based Accelerator for Frequent Itemset Mining," *ACM Trans. Reconf. Technol. Syst.*, vol. 6, no. 1, pp. 1–17, may 2013.

[9] S. Sun and J. Zambreno, "Mining Association Rules With Systolic Trees," in *Intl. Conf. on Field Prog. Logic and Apps, 2008. FPL 2008.* IEEE, 2008, pp. 143–148.

[10] J. Teubner, R. Müller, and G. Alonso, "Frequent Item Computation on a Chip," *Knowl. and Data Eng., IEEE Trans. on*, vol. 23, no. 8, pp. 1169–1181, 2011.

[11] L. Bustio, R. Cumplido, R. Hernández, J. M. Bande, and C. Feregrino, *4th Intl. Workshop, NFMCP 2015, Rev. Sel. Papers*. Springer, 2016, ch. Frequent Itemsets Mining in Data Streams Using Reconfigurable Hardware, pp. 32–45.

[12] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu., and V. S. Tseng, "SPMF: a Java Open-Source Pattern Mining Library," *Jour. of Mach. Learn. Res. (JMLR)*, vol. 15, pp. 3389–3393, 2014.