Online Association Rule Mining over Fast Data

Erdi Ölmezoğulları, Ismail Ari
Computer Engineering Department
Özyegin University
Istanbul, Turkey
erdi.olmezogullari@ozu.edu.tr, ismail.ari@ozyegin.edu.tr

***Abstract*— To extract useful and actionable information in real-time, the information technology (IT) world is coping with big data problems today. In this paper, we present implementation details and performance results of *ReCEPtor*, our system for "online" Association Rule Mining (ARM) over big and fast data streams. Specifically, we added Apriori and two different FP-Growth algorithms inside Esper Complex Event Processing (CEP) engine and compared their performances using LastFM social music site data. Our most important findings show that online ARM can generate (1) more unique rules, (2) with higher throughput, and (3) much sooner (lower latency) than offline rule mining. In addition, we have found many interesting and realistic musical preference rules such as "George Harrison→Beatles". We demonstrate a sustained rate of ~15K rows/sec per core. We hope that our findings can shed light on the design and implementation of other fast data analytics systems in the future.**

Keywords- Fast data, big data, association rule mining, complex event processing, FP-Growth.

I. WHAT IS FAST DATA?

The term "Big Data" is used to refer to challenging data management problems that arise due to the high Volume, Velocity, Variety, and Veracity (4Vs) of the data [1]. "*Fast data* is the *velocity* part of big data" said Tony Baer and thus coined the term [2]. Today, many organizations including social networks, telecommunication operators, media networks, financial institutions and governments generate Terabytes of data each day and attempt to insert this high "volume" data into databases that already contain Petabytes. Even worse, data is replicated by different departments for analysis. Data come from different sources such as sensors, mobile phone call data records, web/system logs, which also results in a wide "variety" of data to be processed and stored (*e.g.* unstructured text, semi-structured XML-JSON, structured CSV, or binary audio/video data). Traditional database management systems cannot cope with this mixed variety of data very well.

New distributed data processing frameworks such as Apache Hadoop [3] (with HDFS and MapReduce) and the NoSQL <*key,value*> stores (*e.g.* HBase [1], Cassandra [2], MongoDB[3], *etc.*) have been invented to cope with the high volume and variety of data (*i.e.* big data batches) within the last decade. However, they were neither designed to process data in-flight or fast streaming data, nor were they designed for doing interactive or looping analytical studies. Recently, this deficiency is also being addressed via projects that would be placed in the fast path of the enterprise data warehouse (EDW) architecture as shown in Fig. 1. Just to name a few, these projects include Google Dremel [4], Apache Drill, and Hadoop Online Prototype (HOP).

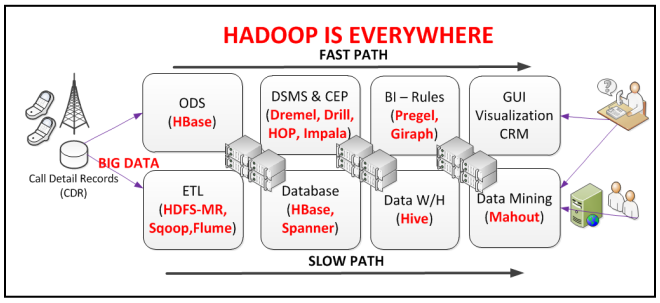


Figure 1. Hadoop and related Apache projects are taking over the enterprise data warehouse (EDW) architecture in both slow and fast paths.

While volume and variety had always been a challenge for data management, its increasing *"velocity"* is the new issue of the 21st century. Attempting to store these data first to analyze them later creates additional costs, unwanted delays to actionable information, and loss of business opportunities. Fortunately, there are now tools to process data on-the-fly as it moves from distributed sources of collection (*e.g.* sensors and routers) to selected destinations. Since the number of data sources and sampling frequencies is steadily increasing, processing in-flight data in-memory is still a challenge. As a result, enterprises increasingly employ Data Stream Management Systems (DSMS) [5,6] and further want to extend them with complex online analytical and mining capabilities [7]. The resulting software tools are sometimes called Complex Event Processing (CEP) engines in the literature [8]. There are now distributed versions of these continuous stream analysis frameworks such as Yahoo S4[4] and Twitter Storm[5], in which each query plan operator is placed in a separate processing engine. The benefits of using CEP engines for stream analytics are at least three-fold:

- They can eliminate unwanted data early in the pipeline, saving further CPU, memory, storage and energy costs.

[1] Apache Hbase project, http://hbase.apache.org/
[2] Apache Cassandra project, http://cassandra.apache.org/
[3] MongoDB by 10gen, http://www.mongodb.org/
[4] Yahoo (and Apache) S4 project http://incubator.apache.org/s4/
[5] Twitter Storm, http://storm-project.net/

978-0-7695-5006-0/13 $26.00 © 2013 IEEE
DOI 10.1109/BigData.Congress.2013.77

- They can turn raw data into actionable information quickly, thus helping businesses catch profitable opportunities or avoid losses due to fraud or operational inefficiencies.
- They can catch transient or emerging patterns, which never show up in an offline data mining analysis.

A. *Motivation: Why is "online" rule mining critical?*

We are at an age, where the trends do not hold for long. Therefore, the temporal aspects of recommendations are extremely important. Unfortunately today, when the streaming data volumes increase, the reaction of data analysts is to increase the minimum support and confidence thresholds to obtain fewer rules with stronger lift (refer to Section 4 for details). Yet, rules that present themselves with high lift values over a long time period (*e.g.* a month) may have become outdated by the end of that period. For example ice-cream, cold drinks, and plastic cups will be extremely popular for the hottest month of the year, just as sand-bags will be popular before a hurricane. However, there will be no sales opportunity after the trend is gone. These temporally emergent patterns occur even faster for online sales or in stock markets where millions of transactions occur every second. Stream rule mining can extract a trend such as "*when stocks HPQ and MSFT go down by >1%, DELL follows*" within a single hour or day. Assume a recent trend that emerges only in a certain timeframe. Its confidence value may not be the highest globally, but its local business value may be quite high. Our suggested systems are designed to catch these rules timely as they occur.

The rest of the paper is as follows. Section 2 describes the related work and compares them against our proposed methodology. Section 3 gives an overview of the ReCEPtor system architecture. Section 4 describes the rule mining algorithms. Section 4 gives performance analysis and results. Finally, Section 5 concludes the paper.

II. RELATED WORK

The concept of "stream mining" did not exist before 2000's. Association Rule Mining (ARM) algorithms such as Apriori [9] and FP-Growth [10] were designed for "offline" or traditional data mining. Our paper utilizes these ARM algorithms, but applies them over streams by integrating them with a CEP engine and testing their performance with different sliding windows. The tools which claim "real-time analytics" capabilities mostly depend on an offline rule extraction phase (*i.e. the training)* using historical data followed by online pattern sequence detection (*i.e. the testing or scoring phase*). We mine the rules themselves in real-time to be able to detect changes in trends and not just detect the existence of already mined historaical rules.

Some of the seminal work on pattern and rule mining over streams [11][12][13] only talk about the challenges, models and issues without any specifics of implementation. Issues mentioned include the need for one-pass algorithms and incremental updates among windows. The former issue (one-pass) arises only when the incoming stream rate is much faster than the stream mining algorithm can process and the latter issue (incremental updates) are only needed if we want to exactly imitate the offline version of the mining algorithm. We demonstrate in this paper that neither of these may be an issue for many rel-life applications.

Frequent pattern mining [11] is not exactly the same task as rule mining, since finding rules requires an additional confidence calculation. In this paper, we only focus on doing ARM over streaming text log data and do not address processing of already stored (offline) data or encoded media (image and audio/video) types. We also do not discuss other fields of stream mining including stream clustering and stream classification. These topics constitute interesting future work.

ARM over streams can be regarded yet another way of reducing raw data size to extract useful information. Other data-based techniques for reducing raw data size without losing the patterns carried inside include sampling, load shedding, sketching, synopsis and aggregations. An overview of these techniques can be found in [14]. Aggregations over streams can include basic counting and summations as well as more complex mean, variance and correlation [15] calculations. StatStream [16] is among the first in literature to discuss scalable stream correlations by using Discrete Fourier Transform (DFT) coupled with approximation techniques. In our previous works, we also applied product-moment correlation techniques to spatio-temporal data and shown that such correlations may not strictly guarantee spatial locality [15,17].

III. RECEPTOR SYSTEM ARCHITECTURE

DSMS engines [5][6] provide effective queuing, scheduling, time and count-window support, and fast in-memory processing of high-speed, continuous, unbounded data streams. They parse, optimize and execute queries written in declarative languages such as Event Processing Language (EPL) in Esper [8]. EPL syntax and semantics are quite similar to that of Structured Query Language (SQL) in databases, but there are additional clauses such as WINDOWs to support sliding or tumbling window-based analysis over data streams. Figure 2 shows these two types of windows. Sliding-time windows are used to buffer event tuples whose occurrence times fall within a certain time period (*e.g.* last 1 minute) and to replace events that are older than the time window. The window will move or "slide" in time with a period that is usually smaller than the size of the window and therefore the two event epochs overlap. Similarly, sliding-count windows buffer last X events.

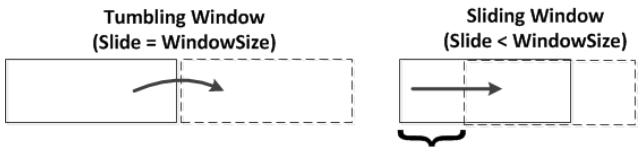


Figure 2. Tumbling vs. Sliding Window semantics.

Tumbling windows on the other hand jump to the next epoch by moving as long as the window size and there is no data overlap between the two event epochs. Other types of windows include Landmark and Damped [9], where the former considers events from a past landmark time until present and the latter gives more weights to recent events.

EPL queries can be used for continuous filtering (*e.g.* **SELECT** x,y **FROM** Stream<x,y,z> **WHERE** …) as well as aggregations: algebraic (COUNT, SUM, AVERAGE) or holistic (MIN, MAX). Complex aggregation functions such as TOP-K, DISTINCT, QUANTILES, and SKYLINE can also be implemented. We slightly modified some of the open source ARM algorithms and integrated them with Esper CEP engine. We also implemented new EPL clauses for these algorithms so that they can be used just as easily as any other window-based (sliding or tumbling) aggregation query.

Figure 3 shows the components and high-level architecture of our data stream analytics system. High-speed raw data streams are first subjected to Select and Project operators inside the DSMS to get rid of tuples unwanted further in the data pipeline. The data volume can be reduced row-wise (Select) and/or column-wise (Project). Basically, the preprocessing stage of the stream mining is done at this stage. The rest of the complex analytics and mining clauses are implemented inside the CEP system, which is deployed either as a private or public cloud system. Data from streams can be correlated among each other or with data stored in persistent repositories such as DBMS and NoSQL systems. NoSQL ("Not-only SQL") is a generic term for highly-scalable, distributed data storage and processing engines coupled with a declarative or procedural language for analytics on top. NoSQL systems are used in support of the event queues to extend the correlation windows.

New CEP operators developed in this and recent previous work are denoted with the *butterfly* sign in Figure 3: aggregate (correlation) and stream mining (Apriori & FP-Growth). The statistical results and alerts are sent to Business Process Management systems and visualization tools for further actions. The feedback loop shown at the CEP system output denotes the registration of rules learned through predictive mining back into the CEP engine for faster descriptive processing. For example, an association rule such as (A&B⇒C) can be registered as a sequence <<A,B>,C>. Over time the system will be more sensitive against complex events, patterns or rules that it has seen before (similar to reinforcement learning), hence the name ReCEPtor was given to it.

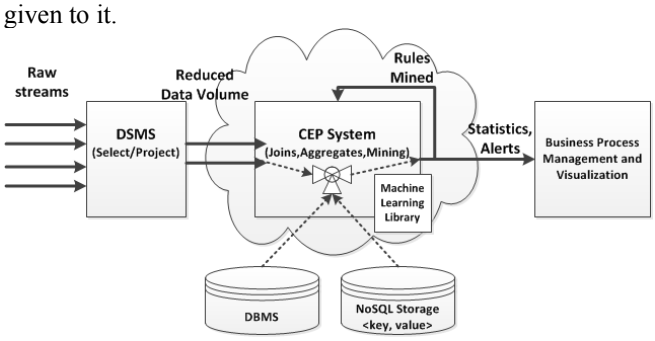


Figure 3. Data stream analytics and mining system architecture.

IV. RULE MINING OVER STREAMS

This section describes implementations of popular ARM algorithms, Apriori and FP-Growth, over data streams and their use for real-time rule generation.

A. Association Rule Mining (ARM)

Assume there are items or *itemsets* $\boldsymbol{I} = i_1, i_2 \ldots i_m$, in a database ***D***. An *association rule* is denoted as ***X⇒Y (S,C),*** where X & Y are two disjoint frequent itemsets (*i.e.* $X \subset I$, $Y \subset I$ and $X \cap Y = \emptyset$). The total *support* **S** and *confidence* **C** values of the itemsets are calculated using Equ. (1) and Equ. (2). *Support* of an itemset is the percentage of records in a database that contain that itemset (either X, Y, or both). *Confidence* of the above rule is calculated as the percentage of records that contain X that also contain Y.

$$Support(X \Rightarrow Y) = \frac{s(X \cup Y)}{|D|} \qquad (1)$$

$$Confidence(X \Rightarrow Y) = \frac{s(X \cup Y)}{s(X)} \qquad (2)$$

In Apriori [9], all items in the database are transformed into transactions with as transaction id (TID) and their support values are calculated using Equ. (1). Next, items with (*support* < *minSupport)* threshold values are eliminated. The remaining items are used to construct itemsets (*i.e.* all subsets) with *<2,3, ... ,n> tuples* combinatorically. The confidence values for each itemset is calculated using Equ. (2) and itemsets with (*confidence* > *minConfidence)* threshold are saved as current *rules*.

Apriori algorithm - although it is among the first invented ARM algorithm and the most famous one- is computationally costly because of its combinatorial approach. If the transaction count in the database is *N*, unique item count is *k*, the total itemset count is $M=2^k-1$, and maximum transaction length is *w*, then the time complexity of Apriori is O (*N.M.w*) [16], which depends on the unique item count, *k,* exponentially.

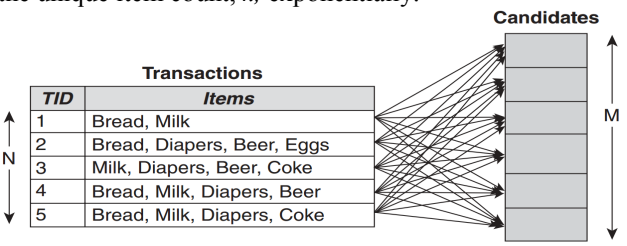


Figure 4. Construction of the candidates from transactions [16].

FP-Growth [10] is another popular algorithm designed to address the performance problems of Apriori. FP-Growth also uses the *minSupport* and *minConfidence* threshold values, but it builds the candidate set differently from Apriori. It first orders items in the database based on frequency, stores them in a sorted *FrequencyList or Flist*, and prunes the list using the *minSupport* threshold. It creates a Frequent-Pattern tree or *FP-tree* data strructure after pruning. The rules are extracted from the FP-tree using the minConfidence value.

B. *Methodology – Experiment Setup*

We obtained the Java implementations of the Apriori and FP-Growth algorithms from the Association rule library of the famous machine learning tool called Weka [19] and integrated these algorithms in Esper engine which is also Java based. Later we integrated a second, more-efficient version of FP-Growth algorithm called SPMF (Sequential Pattern Mining Framework) by Phillip Fournier-Viger into ReCEPtor. One needs to implement a custom aggregation function in Esper (AggreationSupport class) to add new operators. We implemented this interface for each algorithm. The EPL continuous query for Apriori looks like this:

```
SELECT Apriori(parameters, table.feature1, table.feature2)
FROM event.win:length(5) AS table
```

The parameters we used to test Weka's Apriori algorithm inside Esper were as follows: **'-N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1'**, where N is the number of rules to output for each window, T is the metric type by which to rank rules (0=confidence | 1=lift | 2=leverage | 3=Conviction), C is minimum metric score (e.g. min confidence = 0.9) of a rule, U/M are the upper/lower bounds for minimum support (defaults = 1.0 and = 0.1), D is the delta by which the minimum support is decreased in each iteration (default = 0.05), S is the significance level, and c is the class index (default = last). In dynamic streaming environments fixed manual settings could lead to either too many or too few rules to be extracted. Therefore, we initially selected to leave this U/M bounds adjustment to be dynamically made by the Weka system. Table.feature1 is the transaction schema and Table.feature2 is the item schema. The EPL continuous query for FP-Growth similarly looks like this:

```
SELECT FPgrowth(parameter, table.feature1, table.feat2)
FROM event.win:length(5) AS table
```

The parameters we used to initialize Weka's FP-Growth algorithm inside Esper were as follows: **'-P 2 -I -5 -N 10 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.7'**, where N, T, C, D, U/M are the same as Apriori; P is the attribute index for binary attributes in normal dense instances (default index 2 for sparse instances is used), I is the maximum number of items to include in large items sets (and rules) (default = -1, i.e. no limit.).

C. *Transactions and Sessions*

Most ARM studies use examples from the retail shopping domain: "a customer who buys X and Y also buys Z". In this domain a transaction can be clearly defined by a shopping receipt. How about unbounded sessions such as online music streaming? How can we define a transaction? A time-based "transaction" or session in LastFM listening dataset is defined as the list of songs listened by each user during the window of analysis. This view of session or transaction is analogous to the one-time visit of a customer when he/she goes to a retail store and buys a set of items. Just like the same customer may visit the retail store later to buy another set of items, the LastFM music user is assumed to listen to another set of songs if he/she has a batch of records in other windows.

Fig.5 illustrates the relations of tumbling windows of 10K (*N=10,000)* through 100K row count sizes with each other. A rule mining analysis can be done either on 10K tumbling windows, or 20K, *etc.* to represent a session. We actually tried even smaller windows sizes (only 100 or 1000) and upto the complete workload size (to simulate offline processing). We found that windows that are too small do not generate any rules and windows that are too big will take too long to process similar to an offline data mining operations. Before a window slides or tumbles to the nest epoch, rules that are found in the current window have to be published immediately. We plotted the transaction counts for different window sizes in the results section (Fig 6.). Fig.5 also shows where the extracted sample rules first appeared: for example *K0* first appeared in [10K-20K] interval, *K1* in [30k-40k] and so on. The goal should be to publish these rules as early as possible, so that the required business actions can be taken immediately.

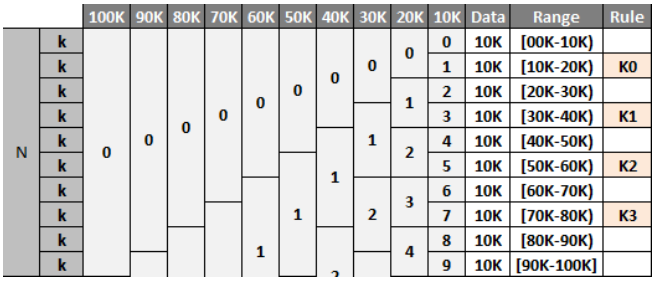


Figure 5. Sessionization using tumbling windows: window sizes, their item ranges and the rules founds (K0-K3).

The minSupport and minConfidence values can also affect the rule count results for different windows sizes. We fixed them after the first comparison of Apriori and Weka's FP-Growth (more about these in Section 5).

D. *LastFM Dataset and Preprocessing*

LastFM data contains information about ~1000 people (lastfm-1K dataset) [20] listening to songs in the LastFM databases. There are approximately 75,000 unique artists, a few hundred thousand unique songs, and millions of transactions in this ~3GB dataset from 2005-2011. Briefly, the fields include <*user-id, timestamp, artist-mbid, artist-name, song-mbid, song-title*>. In the preprocessing phase, we first cleaned the records with missing artist information and removed the time-song fields which did not contribute

to the rule extraction. This process was done offline and our ongoing work includes doing preprocessing online as well.

We used count-based sliding windows. Finally, we had two features of the dataset (<*user-id, artist-mbid*>). Since the Apriori algorithm uses high amounts of memory, just for that algorithm, we further trimmed the data to include users that listened to more than 100 songs and songs that have been listened more than 3000 times overall. This resulted in 967 unique users listening to 1105 unique artists. This work only reports the performance studies from 5.7 million rows belonging to year 2008 data, but a wider range study including 18 million rows from 2007-2008-2009 were also done (similar results were obtained and thus not reported).

V. PERFORMANCE RESULTS

The questions we tried to answer in our experiments are as follows: (1) Can we find transient rules by online rule mining that do not appear in offline rule mining? (2) Is online rule mining more process-efficient and does it generate the results faster? (3) Are the rule results generated by two strategies same, similar or different? (4) Can we learn rules (whether local or globally valid) earlier and publish them with reduced latency? The analysis results in this section answer these questions.

The results given in Table I show that Weka's FP-Growth algorithm integrated into Esper (for finding Top10 rules in 5.7M row count) runs 75x-613x faster than the Weka's Apriori algorithm over the same data stream (61/0.81 and 226/0.37). This is in line with most previous work [12], since FP-growth avoids the iterative, combinatorial candidate generations calculated by Apriori. Due to its poor performance, we later excluded Apriori from the online ARM comparisons. The results for other analysis and comparisons with Apriori are given in our previous related work [15], therefore we skip details here.

TABLE I: RESULTS OF OFFLINE ANALYSIS.

	[i:967 a:1105]	[i:1105 a:967]
Apriori	61.403s	226.502s
FPGrowth	0.811s	0.369s

(i: instances, a: attributes)

After the elimination of Apriori due to unsatisfactory performance in stream processing, we fixed and used *minSupport=0.01, minConfidence=0.9* values. As described above these parameters can be changed dynamically by the system when no rules our found, but we fixed them to be able to compare the performances and published rules of two similar FP-Growth (Weka and SPMF) algorithms.

As the window lengths increase, both the number of items (M) and the transaction counts (N, w) per window increase. Fig 5. shows the transaction sizes grouped per user. We can clearly see that longer windows will have more transactions (e.g. 100K→550) and shorter windows less transactions (e.g. 10K→300). Since the support and the confidence values use transaction counts, it will be harder items to find support in larger windows. Therefore, many local rules that find support in their respective windows do not appear when a global analysis is conducted. The minThreshold values can be decreased as window sizes increase, but in that case the computation times will also increase. Fig 6. displays the total number of rules (unique vs. non-unique) found by different tumbling window sizes at the end of the 5.7 million rows. The numbers descrease as windows sizes increase as the minSupport value was kept constant. 10K windows generate 4736 rules (3894 unique) whereas 100K windows generate 570 rules (527 unique). We can also see that the number of non-unique and unique rules found respective windows to not differe much, which is a strong indicator for the locality of rules.

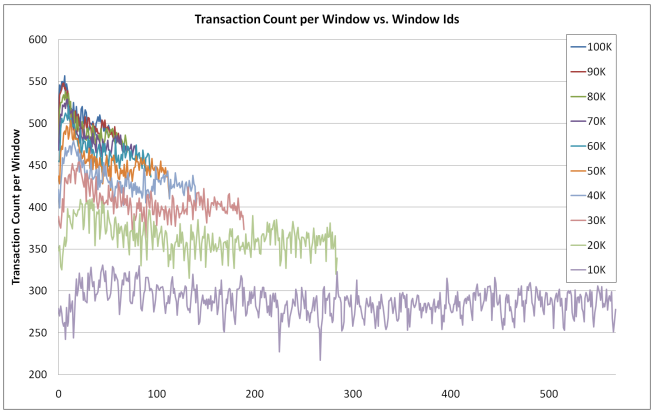


Figure 6. Transaction counts for different tumbling window sizes (Y-axis) given per window id (X-axis).

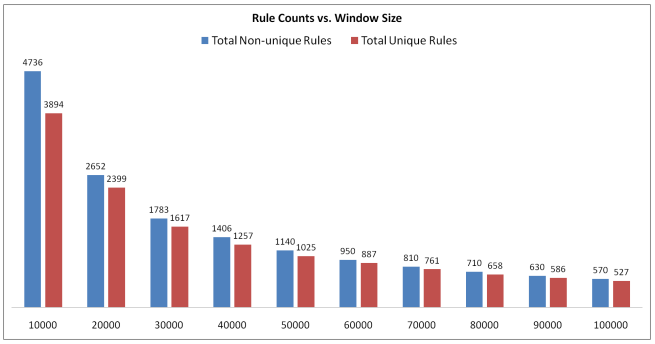


Figure 7. This figure shows the total non-unique and unique rule counts found by the analyses that use different tumbling window sizes.

To understand how the rules found in Fig 7. by different tumbling window size compare to other window sizes we conducted detailed set-based ($\cup,\cap,-,\subseteq$) similarity and difference analysis. For similarity measure we used both the Jaccard=$\cap/\cup$ and F1 measure, but we report only Jaccard in Fig. 8 as the results were quite similar. The matrix in Fig 8. is symmetric as expected with a band of 1's in the diagonals as it is the same set. Note that the Jaccard value drops sharply after this to 0.15 range for comparisons of windows with 10K size difference down to 0.02 for 90K size

difference. This finding suggests that the unique rules found by the 10K window analysis in Fig.7 are 89% different (only 11% similar) to the rules found by the 20K window analysis. We do not show that matrices for set differences and set intersections for brevity.

	10K	20K	30K	40K	50K	60K	70K	80K	90K	100K
10K	1,00	0,11	0,07	0,05	0,04	0,04	0,03	0,03	0,02	0,02
20K	0,11	1,00	0,13	0,10	0,07	0,06	0,04	0,03	0,02	0,02
30K	0,07	0,13	1,00	0,13	0,09	0,08	0,06	0,04	0,04	0,03
40K	0,05	0,10	0,13	1,00	0,14	0,10	0,07	0,05	0,04	0,04
50K	0,04	0,07	0,09	0,14	1,00	0,16	0,12	0,09	0,07	0,07
60K	0,04	0,06	0,08	0,10	0,16	1,00	0,16	0,11	0,09	0,06
70K	0,03	0,04	0,06	0,07	0,12	0,16	1,00	0,16	0,12	0,09
80K	0,03	0,03	0,04	0,05	0,09	0,11	0,16	1,00	0,16	0,11
90K	0,02	0,02	0,04	0,04	0,07	0,09	0,12	0,16	1,00	0,14
100K	0,02	0,02	0,03	0,04	0,07	0,06	0,09	0,11	0,14	1,00

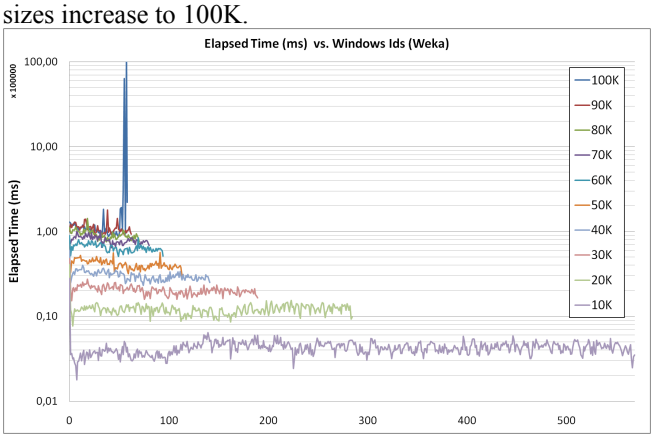
Figure 8. Jaccard similarity of rules found by different window sizes.

We found that only two popular rules were common among the analysis using different window sizes. These rules were (Rule1) **George Harrison=>The Beatles** (i.e. people who listen to George Harrison also listen to the beatles) and (Rule2) **Hilary Duff=>Britney Spears**. George Harrison, is the guitarist of The Beatles who continued to sing similar songs and apparently continued to get liked by the Beatles fans. Similarly Hilary Duff and Britney Spears have similar singing styles and do have a similar fan base. Fig. 9 shows which rule was found in which window. Rule 1 & 2 appear frequently in different window ids of analysis with all window sizes. Rule 3 and Rule 4 are also popular, but do appear in fewer many window sizes and ids. These rules are "**Iron & Wine and Coldplay=>Radiohead" (Rule3)** and **"Blackfield=>Placebo" (Rule4)**. The important thing to note is that we can pinpoint rules as they emerge and publish them without any delay.

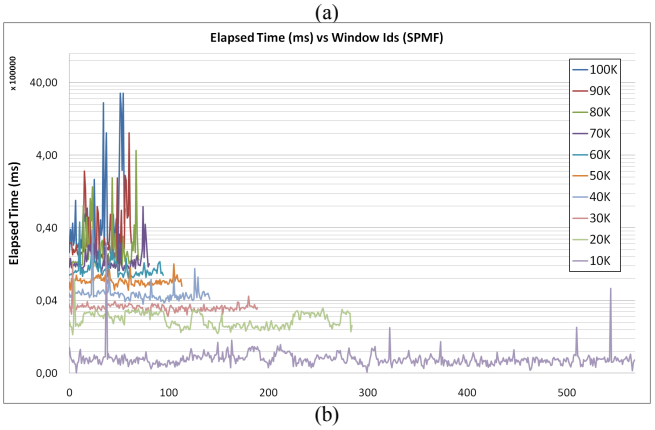
	Rule-1		Rule-2		Rule-3		Rule-4	
Window	Frequency	Window Id	Frequency	Window Id	Frequency	Window Id	Frequency	Window Id
10K	4	{274,371,499,522}	4	{50,87,337,555}	0	-	0	-
20K	1	{54}	1	{25}	0	-	0	{223}
30K	3	{91,97,123}	1	{29}	1	{55}	0	-
40K	3	{68,101,132}	2	{2,84}	1	{29}	1	{111}
50K	4	{54,58,81,106}	3	{18,21,67}	2	{33,39}	1	{22}
60K	5	{45,67,78,87,88}	1	{15}	0	-	1	{74}
70K	1	{53}	2	{1,15}	1	{28}	0	-
80K	4	{50,60,64,65}	1	{11}	1	{4}	0	-
90K	2	{41,57}	2	{10,12}	0	-	0	-
100K	4	{21,37,40,51}	1	{9}	1	{3}	1	{4}

Figure 9. Rules found by analysis in all window sizes (1-2) and other popular rules (3-4) that did not appear just as frequently as 1-2.

Note that different ARM algorithms would find the same set of rules, therefore the results of the analyses in Figures 6→9 would be the same. But how do their performances compare. We have seen before that Apriori was inapplicable due to its exponential time complexity. Therefore, we implemented and integrated a second, more-efficient version of the FP-Growth algorithm called SPMF (Sequential Pattern Mining Framework) into ReCEPtor and compared it with the FP-Growth implementation in Weka. The results are given in Fig.10. All results were obtained on an IBM server with Intel Xeon processors, but only a single core was used. We hope to report performance results of parallel execution with Esper's HA (high availability) mode in the future. In Figure 10, we can see that SPMF is 6-7x faster than Weka's FP-Growth (e.g. compare 10K window size), but the performance results of Weka's implementation are more stable. SPMF can vary widely, especially as window sizes increase to 100K.



(a)



(b)

Figure 10. Per window executions times of the (a) FP-Growth implementation in Weka and (b) FP-Growth implementation in SPMF.

Fig. 11 shows the total time for different windows sizes and the results are quite critical. First of all, it shows that the offline rule mining operation that becomes impossible with thousands of users and millions (5.7M) of rows becomes possible and feasible with an online analysis as we have done in ReCEPtor. Note that a year's worth of LastFM data was mined in a few minutes. We see that smaller window sizes can be processed much faster as window sizes decrease and yet lots of unique rules will still be found as we have demonstrated above. SPMF is ~*7x* faster than Weka's FP-Growth for the 10K windows, but the difference reduces to ~*2x* for 90K windows and almost equals for the 100K. The fastest analysis in Fig. 11 shows us that it is possible to process 5.7 Million rows in 6.42 minutes in a single core giving us an immense rate of ~887K rows/min per core or ~15K rows/sec per core. Note that this is *not* a simple stream filtering operation, but a complex algorithmic calculation. This is the rate at which we can sustain ARM over fast streams. Many streaming applications do not

generate data this fast. Yet, as the support and confidence calculations are embarrasingly parallel we can easily sustain much higher rates with multi-core servers. We estimate that we can potentially mine rules over a fast stream of ~600K rows/sec using a 40 core IBM symmetric multi-processor (SMP) server, but the claim bears a proof which we leave as future work. However, our performance findings here render the strong assumptions about the need for "one-pass stream mining" algorithms over fast data questionable (whether they are really needed or not).

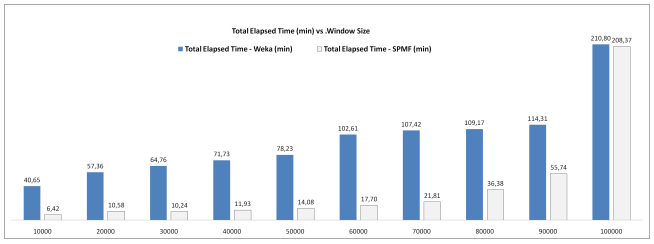


Figure 11. Total processing times of two FP-Growth algorithms (Weka's and SPMF) for different window sizes.

Fig. 12 show the memory usage of the SPMF FP-Growth rule mining algorithm, which was more efficient than Weka's both in time and space. First, even the largest memory used (3.17GB) is easily obtained on any commodity server – even laptops- today. All the stream processing is done in-memory therefore this analysis was required and useful. Larger window sizes require more memory for holding the data and creating the candidate sets as can be expected. The increased memory usage towards smallest window sizes are due to the increased number of rules generated as shown in Fig. 7.

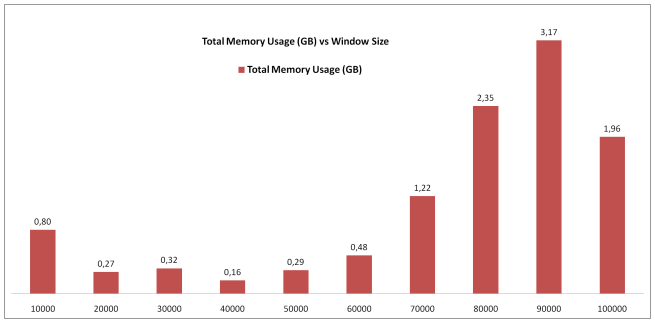


Figure 12. Memory usage of SPMF FP-Growth algorithm for different tumbling window sizes.

During our research we discussed the use of incremental updates among windows and even briefly tested the outcomes. However, the disadvantages were two-fold: (1) We were keeping more state around, thus increasing the memory usage and (2) The additional states kept turned a real-time analytics apporach into the imitation of an offline rule mining algorithm, which we did not desire.

VI. CONCLUSIONS

Conducting on-the-fly analytics over big and fast data is the newest IT challenge. In this paper, we presented implementation details and performance results of *ReCEPtor*, our system for "online" Association Rule Mining (ARM) over big and fast data streams. We made critical comparisons of "online" vs. "offline" data mining. Specifically, we implemented and tested Apriori and two different FP-Growth algorithms (from Weka and SPMF) inside Esper CEP engine and compared their performances using LastFM social music site data at different tumbling window sizes.

We found that with online ARM we can generate many unique rules, with a sustained throughput up of ~15K rows/sec, and lowest possible latency for rule publications (as they emerge) compared to offline rule mining. We have found many interesting and realistic musical preference rules such as "George Harrison→Beatles" and reported some of them here. We hope that our findings can shed light on the design and implementation of other fast data analytics systems in the future.

ACKNOWLEDGMENT

This work was supported in part by European Union FP7 Marie Curie Program BI4MASSES Grant, Turkish National Institute of Science and Technology (TUBITAK) Grant 190E194, IBM Shared University Research program, Turkish Telecomm and Avea Labs. We thank our sponsors for their continued support.

REFERENCES

[1] What is Big Data? http://www-01.ibm.com/software/data/ bigdata/

[2] D. Hansen, Big Data Gets Real-time: Oracle Fast Data, Oracle White Paper, March 28th, 2013.

[3] Apache Hadoop Project, http://hadoop.apache.org/

[4] Sergey Melnik, et al, Dremel: interactive analysis of web-scale datasets, Proc. of the VLDB Endowment, v.3 n.1-2, 2010

[5] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, ACM PODS, 2002, June, pp 1-16.

[6] D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik. "Aurora: A new model and architecture for data stream management", *VLDB Journal*, 12(2):120–139, August 2003.

[7] IBM Infosphere Stream Mining Toolkit, http://www-01.ibm.com/software/data/infosphere/streams/

[8] EsperTech Inc. Event Stream Intelligence, http://www.espertech.com/

[9] R Agrawal and R Srikant, Fast algorithms for mining association rules, Proc. 20th Int. Conf. on Very Large Data Bases (VLDB), 1994

[10] C. Borgelt, An Implementation of the FP-growth Algorithm, ACM Workshop of Open Source Data Mining Software, (OSDM), pages 1-5, 2005.

[11] C. Giannella, J. Han, J. Pei, X. Yan, P. S. Yu; Mining Frequent Patterns in Data Streams at Multiple Time Granularities; Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT; 2003.

[12] N. Jiang, L. Gruenwald, Research issues in data stream association rule mining, In SIGMOD Record, Vol 35, No 1, March 2006.

[13] P. S. Yu, Y. Chi: Association Rule Mining on Streams. Encyclopedia of Database Systems (2009)

[14] M. M. Gaber, A. Zaslavsky, S. Krishnaswamy; Mining Data Streams: A Review; ACM SIGMOD Record Vol. 34, No. 2; June 2005.

[15] I. Ari, E. Olmezogullari, O. F. Celebi, Data Stream Analytics and Mining in the Cloud, IEEE DaMiC 2012

[16] Y. Zhu, D. Shasha, StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, Proc. of VLDB 2002.

[17] I. Ari, O. F. Celebi, Finding Event Correlations in Federated Wireless Sensor Networks, In Federated Wireless Sensors and Systems Workshop (FedSenS), In Conj. with IWCMC, 2011

[18] Pang-Ning Tan, Introduction to Data Mining, Ch 6. Association Analysis: Basic Concepts and Algorithms, 25 March 2006

[19] Weka ML, http://www.cs.waikato.ac.nz/~ml/weka/

[20] LastFM, http://www.dtic.upf.edu/~ocelma/MusicRecommendation-Dataset/ lastfm-1K.html