Merging data streams in operational risk management

Paolo Giudici
Department of Statistics and applied economics
University of Pavia
Strada Nuova 65, 27100 Pavia, Italy
e-mail: giudici@unipv.it

Background

The motivation of this talk is to develop efficient statistical methods aimed at measuring the performance of business controls, through the development of appropriate operational risk indicators.
A number of recent legislations and market practices are motivating such developments. For instance, the New Basel Capital Accord" (BCBS, 2001), published by Basel Committee on Banking supervision, requires financial institutions to measure operational risks, defined as "the risk of loss resulting from inadequate or failed internal processes, people and systems or from external events". In the context of information systems, the recently developed ISO7799 establishes the need of risk controls aimed at preserving the security of information systems. Finally, the publicly available specification PAS56, in setting criteria that should be met to maintain business continuity of IT-intensive companies, also calls for the development of statistical indicators aimed at monitoring the quality of business controls in place.
In this paper we shall focus on the Basel Accord, however keeping in mind that what developed here for the banking sector can be extended to the general enterprise risk management framework (on this matter see e.g. Bonafede and Giudici, 2007).

The Bank of International Settlements (BIS) is the world's oldest financial institution, whose main purpose is to encourage and facilitate cooperation among central banks (for more details see BCBS, 2001). In particular, BIS established a commission, the BaselCommittee on Banking Supervision (BCBS, in order to formulate broad supervisory standards, guidelines and recommend statements of best practice. The ultimate purpose of the Committee is the prescription of capital adequacy standards for all internationally active banks. In 1988 the BCBS issued one of the most significant international regulations impacting on the financial decision of banks: the Basel Accord. Subsequently, the BCBS worked on a revision, called the New Accord on Capital Adequacy, or Basel II (BCBS; 2001). This new framework, developed by the Committee in 2002 to ensure the stability and soundness of financial systems, was based on three 'pillars': minimum capital requirements, supervisory review and market discipline.

The crucial novelty of the new agreement was the identification of operational risk as a new category separated from the others. In fact, it was only with the new agreement that the Risk Management Group of the Basel Committee proposed the current definition of Operational Risk:
"*Operational Risk is the risk of loss resulting from inadequate or failed internal processes, people and systems or from external events*"

© Revue MODULAD, 2007

The risk management group also provided a standardized classification of operational losses into eight Business Lines (BL) and seven Event Types (ET).

The aim of operational risk measurement (for a review see e.g. Alexander, 2003; King, 2001 or Cruz, 2002) is twofold: on one hand, there is a prudential aspect, which involves setting aside an amount of capital that can cover unexpected losses. This is typically achieved estimating a loss distribution deriving functions of interest from it (such as the Value at Risk: VaR); on the other hand, there is a managerial aspects, for which the issue is to rank operational risks in an appropriate way, say from high prority to low priority, so to individuate appropriate management actions directed at improving preventive controls on such risks.
In general, the measurement of operational risks leads to the measurement of the efficacy of controls in place at a specific organisation: the higher the operational risks, the worse such controls.
The complexity of operational risks and the newness of the problem have driven international institutions, such as the Basel Committee, to define conditions that sound statistical methodologies should satisfy to build and measure operational risk indicators.

Methods

Statistical models for Operational Risk are grouped into two main categories: 'top-down' and 'bottom-up' methods. In the former, risk estimation is based on macro data without identifying the individual events or the causes of losses. Therefore, operational risks are measured and covered at a central level, so local business units are not involved in the measurement and allocation process. 'Top-down' methods include the Basic Indicator Approach (see, for example, Yasuda (2003) or Pezier (2002)) and the Standardized Approach (see Cornalba and Giudici (2004) or Pezier (2002) for more details), where risk is computed as a certain percentage of the variation of some variable, as, for example, gross income, considered as a proxy for firm performance. This first approach is suitable for small banks, that prefer a cheap methodology, easy to implement.
'Bottom-up' techniques, instead, use individual events to determine the source and amount of operational risk. Operational losses can be divided into levels corresponding to business lines and event types and the risks are measured at each level and then aggregated. These techniques are particularly appropriate for large sized banks and those operating at the international level, since they can afford the implementation of sophisticated methods, sensitive to the bank's risk profile. Methods belonging to this class are grouped into the Advanced Measurement Approaches (AMA) (BCBS, 2001). Under the AMA, the regulatory capital requirement will equal the risk measure generated by the bank's internal operational risk measurement system using the quantitative and qualitative criteria set by the Committee. It is an advanced approach as it allows banks to use external and internal loss data as well as internal expertise (Giudici and Bilotta, 2004).
Statistical methods for operational risk management in the bottom-up context have been developed in very recent years. Two main approaches have emerged based: scorecard and actuarial.
The scorecard approach is based on the so-called *Self Assessment*, which is based on the experience and the opinions of a number of internal "experts" of the company, who

usually correspond to a particular business unit. An internal procedure of control self assessment can be periodically done through questionnaires, submitted to risk managers (experts), which gives information such as the quality of internal and external control system of the organisation on the basis of their own experience in a given period. In a more sophisticated version, experts can also assess the frequency and mean severity of the losses for such operational risks (usually in a qualitative way).
Self assessment opinions can be summarised and modelled so to attain a ranking of the different risks, and a priority list of intervention in terms of improvement of the related controls.
In order to derive a summary measure of operational risk, perceived losses contained in the self-assessment questionnaire can be represented graphically (e.g. through a histogram representation) and lead to an empirical non parametrical distribution. Such a distribution can be employed to derive a functional of interest, such as the 99,9% percentile (the value at risk).
When loss data is available, it is possible to build actuarial models. These are based on the analysis of all available and relevant loss data with the aim to estimate the probability distribution of the losses. The most employed methods are actuarial models (see e.g. King, 2001; Cruz, 2002; Frachot et al., 2001; Dalla Valle et al., 2007), often based on extreme value distributions. Another line of research suggests the use of Bayesian models (see e.g. Yasuda, 2003, Cornalba and Giudici, 2004 and Fanoni, Giudici and Muratori, 2005)
The main disadvantage of actuarial methods is that they rely their estimates only on past data, thus reflecting a backward-looking perspective. Furthermore, it is often the case, especially for smaller organisations, that, for some business units, there are no loss data at all. Regulators thus recommend to develop models that can take into account different data streams, not only internal loss data (see e.g. BCBS, 2001). These streams may be: self assessment opinions, usually forward looking; external loss databases, usually gathered through consortiums of companies; data on key performance indicators.

Merging data streams

Scorecard and actuarial methods represent the methods mostly employed, in the literature as well as in professional practice. The usage of the two approached depends on the available data streams. While scorecard methods typically use self-assessment data, actuarial models do use internal loss data.
The disadvantage of these approaches is that they consider only one part of the statistical information available to estimate operational risks. Actuarial methods rely only on past loss data (backward looking) and, therefore, do not consider important information on the perspective and the evolution of the considered company; on the other hand, scorecard methods are based only on perceived data (forward looking) and, therefore, do not reflect well past experiences.
A further problem is that, especially for rare events, a third data stream may be considered: external loss data. This source of data is made up of pooled records of losses, typically higher than a certain value (e.g. 5000 €), collected by an appropriate association of banks.
It becomes thus necessary to develop a statistical methodology that is able to merge three different data streams in an appropriate way, yet maintaining simplicity of

© Revue MODULAD, 2007

interpretation and predictive power. In the talk we shall propose a flexible nonparametric approach that can reach this objective.
More precisely, we shall show how data streams can be combined and merged so to produce prudential measures of operational risks as well as scores that can be used to rank operational risks in terms of their relative priority of intervention. .
We shall compare our method with existing ones, theoretically but also with reference to an available database, according to the data mining paradigm (see e.g. Giudici, 2003).

References

Alexander, C. (2003). *Operational Risk: Regulation,Analysis and Management*, London. Financial Times, Prentice Hall.

Basel Committee on Banking Supervision (2001) *Working Paper on the Regulatory Treatment of Operational Risk*, (www.bis.org).

Bonafede, E.C. and Giudici, P. (2007). Bayesian networks for enterprise risk assessment. To appear in *Physica A*.

Cornalba, C. and Giudici, P. (2004). Statistical models for operational risk management. *Physica A*, n. 338, pp. 166-172.

Cruz M. (2002) *Modeling, Measuring and Hedging Operational Risk*, John Wiley & Sons, Chichester.

Dalla Valle, L., Fantazzini, D. and Giudici, P. (2007). Copulae and Operational Risk. To appear in *International Journal of Risk Assessment and Management.*

Fanoni, F., Giudici, P. and Muratori, M. (2005). Il rischio operativo: misurazione, monitoraggio, mitigazione. Il sole 24 ore.

Frachot, A., Georges, P. and Roncalli, T. (2001). Loss distribution approach for operational risk. Working Paper, Groupe de Recherche Opérationnelle du Crédit Lyonnais.

Giudici, P. and Bilotta, A. (2004). Modelling Operational Losses: a Bayesian Approach, *Quality and Reliability Engineering International*, n.20, pp. 407-417.

Giudici P. (2003) *Applied Data Mining: Statistical Methods for Business and Industry*. John Wiley & Sons, Chichester.

King J. (2001) *Operational Risk. Measurement and Modelling*, John Wiley & Sons, Chichester.

Pezier, J. (2002). A constructive review of the Basel proposals on operational risk. ISMA Technical report, University of Reading.

Yasuda, Y. (2003). Application of Bayesian Inference to Operational Risk Management. Technical report, University of Tsukuba.

© Revue MODULAD, 2007

Summarizing A 3 Way Relational Data Stream

Baptiste Csernel[1 2], Fabrice Clerot[2] and Georges Hébrail[1]

[1] École Nationale Supérieure des Télécommunications, 46 rue Barrault, 75013 Paris
Département Informatique et Réseaux.
[2] France Télécom R&D, 2 avenue P. Marzin, 22307 Lannion

Abstract. In this paper, we present a novel method to summarize a group of three data streams sharing a relational link between each other. That is, using the relational data base model as a reference, two entity streams and one stream of relationships. Each entity stream contains a stream of entities, each one of them referenced by a key, much in the same way as a primary key references all objects in a regular database table. The stream of relationships contains key couples identifying links between objects in the two entity streams as well as attributes characterising this particular link.
The algorithm presented here produces not only a summary of both entity streams considered independently of each other, but also gives a summary of the relations existing between the two entity streams as well as information on the join obtained joining the two entity streams through the relationship stream.
Keywords: Relational Data Mining, Data Stream Mining, Clustering, Data Stream Summaries, Data Stream Joins.

1 Introduction

The last decades have seen a huge inflation in the amount of information generated by most commercial processes and the rates at which it is produced. This has led to the development of a new field in the data analysis community devoted to the study of infinite streams of data, arriving at a rhythm so fast, and so large that they can't fit in storage and thus have to be treated in one pass. This new field, called data stream analysis has been the subject of a growing attention by different communities including but not limited to those of databases, data mining or machine learning.
This work studies the design of summaries built from such data streams. Much work has already been done to design algorithms capable of producing summaries of any given data streams. However, most real world data does not stand on its own but includes references to different data produced by different streams. That's why we have chosen to interest ourselves in the summary not of a single data stream, but of several data streams joined by relationships. To simplify the problem, we will only consider here a small example of three data streams, one relationship stream, and two entity streams.

2 Related Work

This work is related to much previous work done on data streams in the past few years [1] [2], however, it shares particular relations with two specific problems. The first one is the summary of a single data stream, and the second, the problem associated with the join of two data streams. Both of these problems have been studied separately. However, while the problem treated here might seem like the conjunction of the two previously cited, it is a new and different problem. The goal here is not to first join two streams and then summarize the resulting stream but to make a summary of the streams without having to process the join but while still taking into account the relationship information.
This particular problem has not been much considered yet to our knowledge, and that is why we have decided to propose a solution for it.

© Revue MODULAD, 2007

2 Related Work

We consider a (possibly infinite) set $T = \{t_1, t_2, \dots\}$ of unique, distinguishable *items* that are inserted into and deleted from the dataset R over time. As indicated above, we do not require R to be accessible or even materialized. In general, items that are deleted may be subsequently re-inserted. Without loss of generality, we assume throughout that R is initially empty. Thus we consider an infinite sequence of transactions, where each transaction is either an insertion or a deletion of item t_k from R. We restrict attention to "feasible" sequences such that R is a true set and items can only be deleted if they are present in R. Our goal is to ensure that, after each transaction is processed, S is a uniform sample from R.

We limit our attention to bounded-size sampling schemes which do not require access to R at any time.[3] The best known method for incrementally maintaining a sample in the presence of a stream of insertions to the dataset is the classical "reservoir sampling" algorithm [4] (RS), which maintains a simple random sample of a specified size M. The general procedure is as follows: Include the first M items into the sample. For each successive insertion into the dataset, include the inserted item into the sample with probability $M/|R|$, where $|R|$ is the size of the dataset just after the insertion; an included item replaces a randomly selected item in the sample. One deficiency of this method is that it cannot handle deletions, and the most obvious modifications for handling deletions either yield procedures for which the sample size systematically shrinks to 0 over time or which require access to R.

The only known bounded-size sampling scheme which can handle both insertions and deletions has been proposed in [3]. The idea is to include every item into the sample with probability q and to directly remove deleted items from the sample, if present. The sample is then purged every time it exceeds the upper bound, so that the algorithm is best described as Bernoulli sampling with purging (BSP). Starting with $q = 1$, we decrease q at every purge step. With q' being the new value of q, the sample is subsampled using Bernoulli sampling with sampling rate (q'/q). This procedure is repeated until the sample size has fallen below M. The choice of q' is challenging: on the one hand, if q' is chosen small with respect to q, the sample size drops significantly below the upper bound in expectation. On the other hand, a high value of q' leads to frequent purges, thereby reducing performance. Due to the difficulty of choosing q and, as discussed in the sequel, instability in the sample sizes, this algorithm can be difficult to use in practice.

Our new RP algorithm, described in Section 3, maintains a bounded-size uniform sample in the presence of arbitrary insertions and deletions without requiring access to the base data. The RP algorithm produces samples that are significantly larger (i.e., more space efficient) and more stable than those produced by BSP, at lower cost.

3 Random Pairing

To motivate the idea behind the random-pairing scheme, we first consider an "obvious" passive algorithm for maintaining a bounded uniform sample S of a dataset R. The algorithm avoids accessing base data by making use of new insertions to "compensate" for previous deletions. Whenever an item is deleted from the data set, it is also deleted from the sample, if present. Whenever the sample size lies at its upper bound M, the algorithm handles insertions identically to RS; whenever the sample size lies below the upper bound and an item is inserted into the dataset, the item is also inserted into the sample. Although simple, this algorithm is unfortunately incorrect, because it fails to guarantee uniformity. To see this, suppose that, at some stage, $|S| = M < |R| = N$. Also suppose that an item t^- is then deleted from the dataset R, directly followed by an insertion of t^+. Denote by S' the sample after these two operations. If the sample is to be truly uniform, then the probability that $t^+ \in S'$ should equal M/N, conditional on $|S| = M$. Since $t^- \in S$ with

[3] A broader summary of available uniform sampling schemes can be found in [2].

© Revue MODULAD, 2007

probability M/N, it follows that

$$P\{t^+ \in S'\} = P\{t^- \in S,\ t^+ \text{ incl.}\} + P\{t^- \notin S,\ t^+ \text{ incl.}\} = \frac{M}{N} \cdot 1 + \left(1 - \frac{M}{N}\right) \cdot \frac{M}{N} > \frac{M}{N},$$

conditional on $|S| = M$. Thus an item inserted just after a deletion has an overly high probability of being included in the sample. The basic idea behind RP is to avoid the foregoing problem by including an inserted item into the sample with a probability less than 1 when the sample size lies below the upper bound. The key question is how to select the inclusion probability to ensure uniformity.

3.1 Algorithm Description

In the RP scheme, every deletion from the dataset is eventually compensated by a subsequent insertion. At any given time, there are 0 or more "uncompensated" deletions; the number of uncompensated deletions is simply the difference between the cumulative number of insertions and the cumulative number of deletions. The RP algorithm maintains a counter c_1 that records the number of uncompensated deletions in which the deleted item was in the sample (so that the deletion also decremented the sample size by 1). The RP algorithm also maintains a counter c_2 that records the number of uncompensated deletions in which the deleted item was not in the sample (so that the deletion did not affect the sample). Clearly, $d = c_1 + c_2$ is the total number of uncompensated deletions.

The algorithm works as follows. Deletion of an item is handled by removing the item from the sample, if present, and by incrementing the value of c_1 or c_2, as appropriate. If $d = 0$, i.e., there are no uncompensated deletions, then insertions are processed as in standard RS. If $d > 0$, then we flip a coin at each insertion step, and include the incoming insertion into the sample with probability $c_1/(c_1 + c_2)$; otherwise, we exclude the item from the sample. We then decrease either c_1 or c_2, depending on whether the insertion has been included into the sample or not.

Conceptually, whenever an item is inserted and $d > 0$, the item is paired with a randomly selected uncompensated deletion, called the "partner" deletion. The inserted item is included into the sample if its partner was in the sample at the time of its deletion, and excluded otherwise. The probability that the partner was in the sample is $c_1/(c_1 + c_2)$. For the purpose of the algorithm, it is not necessary to keep track of the identity of the random partner; it suffices to maintain the counters c_1 and c_2. Note that if we repeat the above calculation using RP, we now have $P\{t^- \notin S,\ t^+ \text{ included}\} = 0$, and we obtain the desired result $P\{t^+ \in S'\} = M/N$. A correctness proof for the random pairing algorithm is given in [2].

The RP algorithm with $M = 2$ is illustrated in Figure 1 (left). The figure shows all possible states of the sample, along with the probabilities of the various state transitions. The example starts after $i = 2$ items have been inserted into an empty dataset, i.e., the sample coincides with R. The insertion of item t_3 leads to the execution of a standard RS step since there are no uncompensated deletions. This step has three possible outcomes, each equally likely. Next, we remove items t_2 and t_3 from both the dataset and the sample. Thus, at $i = 5$, there are two uncompensated deletions. The insertion of t_4 triggers the execution of a *pairing step*. Item t_4 is conceptually paired with either t_3 or t_2 — these scenarios are denoted by a) and b) respectively — and each of these pairings is equally likely. Thus t_4 compensates its partner, and is included in the sample if and only if the partner was in the sample prior to its deletion. This pairing step amounts to including t_4 with probability $c_1/(c_1 + c_2)$ and excluding t_4 with probability $c_2/(c_1 + c_2)$, where the values of c_1 and c_2 depend on which path is taken through the tree of possibilities. A pairing step is also executed at the insertion of t_5, but this time there is only one uncompensated deletion left: t_2 in scenario a) or t_3 in scenario b). Observe that the sampling scheme is indeed uniform: at each time point, all samples of the same size are equally likely to have been materialized.

Figure 1 (right) displays the time-average sample size for a range of dataset sizes when running RP and BSP (with $q' = 0.8q$). For each dataset size, we used a sequence of insertions to create both the dataset and an initial sample ($M = 100,000$), and then measured changes in the sample size

© Revue MODULAD, 2007

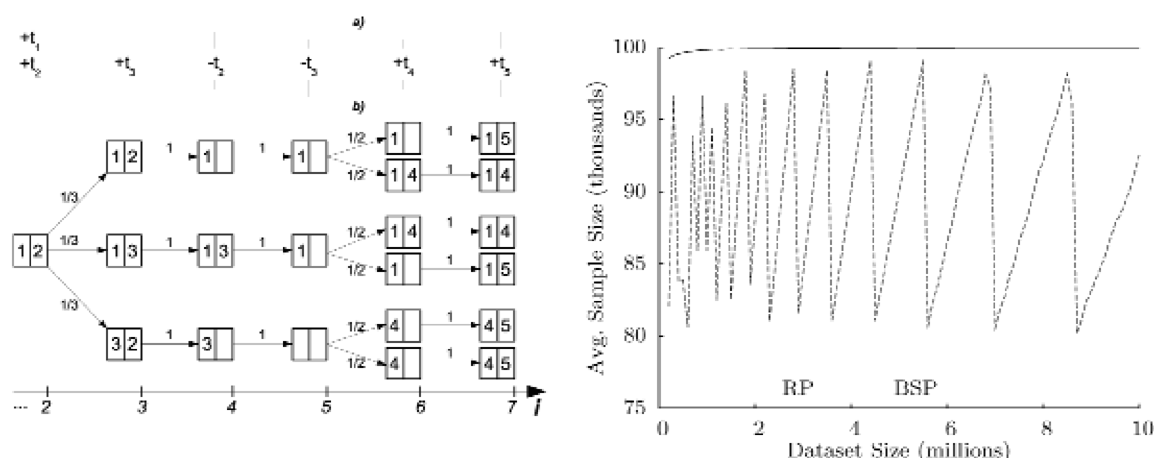


Fig. 1. Random pairing: example and average sample size

as we inserted and deleted 10,000,000 items at random. Clearly, RP produces much more stable samples than BSP, since the latter adjusts the sampling rate only at specific points of time.

3.2 A Negative Result: Resizing Samples Upwards

One might hope that there exist algorithms for resizing a sample upwards without accessing the base data. In general, we consider algorithms that start with a uniform sample S of size at most M from a dataset R and after some finite (possibly zero) number of arbitrary transactions on R — produce a uniform sample S' of size M' from the resulting modified dataset R', where $M < M' < |R|$. Unfortunately, there exists no resizing algorithm that can avoid accessing the base dataset R. To see this, suppose to the contrary that such an algorithm exists, and consider the case in which the transactions on R consist entirely of insertions. Fix a set $A \subseteq R'$ such that $|A| = M'$ and A contains $M + 1$ elements of R; such a set can always be constructed under our assumptions. Because the hypothesized algorithm produces uniform samples of size M' from R', we must have $P\{ S' = A \} > 0$. But clearly $P\{ S' = A \} = 0$, since $|S| \leq M$ and, by assumption, no further elements of R have been added to the sample. Thus we have a contradiction, and the result follows. A time-optimal resizing algorithm (which may access the base data) is given in [2],

4 Summary

Techniques for incrementally maintaining bounded samples over "datasets" — whether relational tables, views, data streams or other data collections are crucial for unlocking the full power of database sampling techniques. Our new RP algorithm is the algorithm of choice with respect to speed and sample-size stability. We have also shown that it is impossible to resize a bounded sample upwards without accessing base data.

References

1. Brian Babcock, Mayur Datar, and Rajeev Motwani. Sampling from a moving window over streaming data. In *Proc. SODA*, pages 633–634, 2002.
2. Rainer Gemulla, Wolfgang Lehner, and Peter J. Haas. A dip in the reservoir: Maintaining sample synopses of evolving datasets. In *Proc. VLDB*, pages 595–606, 2006.
3. Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *Proc. ACM SIGMOD*, pages 331–342, 1998.
4. D. E. Knuth. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Addison-Wesley, 1st edition, 1969.

© Revue MODULAD, 2007

A Practical Evaluation of Load Shedding in Data Stream Management Systems for Network Monitoring

Jarle Søberg, Kjetil H. Hernes, Matti Siekkinen, Vera Goebel, and Thomas Plagemann

University of Oslo, Department of Informatics
P.O. Box 1080, Blindern, N-0316 Oslo
{jarleso, kjetih, siekkine, goebel, plageman}@ifi.uio.no

Abstract. In network monitoring, an important issue is the number of tuples the data stream management system (DSMS) can handle for different network loads. In order to gracefully handle overload situations, some DSMSs are equipped with a tuple dropping functionality, also known as *load shedding*. These DSMSs register and relate the number of received and dropped tuples, i.e., the *relative throughput*, and perform different kinds of calculations on them. Over the past few years, several solutions and methods have been suggested to efficiently perform load shedding. The simplest approach is to keep a count of all the dropped tuples, and to report this to the end user. In our experiments, we study two DSMSs, i.e., TelegraphCQ with support for load shedding, and STREAM without this support. We use three particular network monitoring tasks to evaluate the two DSMS with respect to their ability of load shedding and performance. We demonstrate that it is important to investigate the correctness of load shedding by showing that the reported number of dropped tuples is not always correct.

Keywords: DSMS applications, data stream summaries, data stream sampling

1 Introduction

In this paper, we focus on network monitoring as the application domain for *data stream management systems* (DSMSs). More specifically, we evaluate the load shedding mechanisms of existing DSMSs in the case of *passive network monitoring* where traffic passing by an observation point is analyzed. This field of research is commonly challenged by the potentially overwhelming amounts of traffic data to be analyzed. Thus, data reduction techniques are important. Many of those techniques used by the traffic analysis research community as stand-alone tools are also utilized extensively for load shedding within the DSMSs. Examples include *sampling* [8, 9], *wavelet analysis* [11], and *histogram analysis* [23]. Monitoring tasks may also include timeliness constraints: Storing measurements and performing "off-line" analysis later is impossible in many monitoring scenarios where analysis needs to happen in near real-time. For example, an intrusion detection system (IDS) based on passive traffic measurements needs to generate alerts immediately in the case of a detected intrusion. Internet service providers (ISPs) are interested in real-time measurements of their backbones for traffic engineering purposes, e.g., to re-route some traffic in case congestion builds up in certain parts of the network. In addition, network traffic rarely generates a constant stable input stream. Instead, traffic can be bursty over a wide range of time scales [14], which implies that the monitoring system should be able to adapt to a changing load. For all these reasons, it is very interesting to study the applicability of DSMSs equipped with load shedding functionalities in this context.

The STREAM [1, 3, 4] and Gigascope [7, 6] projects have made performance evaluations on their DSMSs for network monitoring. In both projects, several useful tasks have been suggested for network monitoring. Plagemann et al. [17] have evaluated an early version of the TelegraphCQ [5] DSMS as a network monitoring tool by modeling and running queries and making a simple performance analysis.

With respect to load shedding, Golab et al. [10] sum up several of the different solutions suggested in the DSMS literature. Reiss et al. [22] evaluate sampling, multi-dimensional histograms,

© Revue MODULAD, 2007

wavelet-based histograms, and fixed-grid histograms in TelegraphCQ. They also suggest solutions for stating queries that obtain information about the shedded tuples.

We have used the two public-domain DSMSs; TelegraphCQ v2.0 and STREAM v0.6.0, for network monitoring. We have run different continuous queries and have measured system performance for different network loads. We choose to use these two systems since they are public available and have been used for network monitoring in earlier experiments. TelegraphCQ v2.0 supports load shedding, and STREAM v0.6.0 has not this functionality implemented. To our knowledge, no practical evaluations have been performed with focus on load shedding in DSMSs for network monitoring.

Our contribution is to propose a simple experiment setup for practical evaluation of load shedding in DSMSs. At this point, the experiment setup can be used for measuring *tuple dropping*, which is one of the load shedding functionalities. This is useful for systems that do not support load shedding. For measuring the tuple dropping, we suggest a metric; *relative throughput*, which is defined by the relation between the number of bits a node receives and the number of bits the node is able to compute. Our experiment setup can also evaluate the correctness of systems that support tuple dropping. Here, we show that TelegraphCQ reports less dropped tuples than what seems to be correct. We also use the experiment setup to show how TelegraphCQ behaves over time, and see that load shedding and query results reach an equilibrium. Second, we suggest a set of three network monitoring tasks and discuss how they can be modeled in TelegraphCQ and STREAM.

The structure of this paper is as follows: In Section 2, we describe the tasks and the queries. While discussing the tasks, we try to introduce system specific features for the two DSMSs. Section 3 presents the implementation and setup of our experiment setup, and Section 4 shows the results from our experiments. In Section 5, we summarize our findings and conclude that the experiment setup is useful for performance evaluations of network monitoring tasks for DSMSs. Finally, we point out some future work.

2 Network Monitoring Tasks and Query Design

In this section, we discuss three network monitoring tasks. We give a short description of each task and show how to model the queries. We also argue for their applicability in the current domain. The first two tasks are used in our experiments, while the third task is discussed on a theoretical basis. For that task, we show that some tasks may give complex queries that depend on correct input and therefore can not allow load shedding, since important tuples might be dropped.

For both systems, TelegraphCQ and STREAM, the tasks are using stream definitions that consist of the full IP and TCP headers. These are called `streams.iptcp` and `S`, respectively. The definition is a one-to-one mapping of the header fields. For example, the `destination port` header field is represented by an attribute called `destPort int`. We have tried to match the header fields with available data types in the two systems. For example, TelegraphCQ supports a data type, `cidr`, which is useful for expressing IP addresses.

2.1 Task 1: Measure the Average Load of Packets and Network Load

To measure the average load of packets and the network load, we need to count the number packets obtained from the network and calculate the average length of these packets. In the IP header, we can use the `totalLength` header field for this purpose. This header field shows the number of bytes in the packet. Having the information from this task may be of great importance for the ISP's to look at tendencies and patterns in their networks over longer periods. In TelegraphCQ, we model the query as follows:

```
SELECT COUNT(*)/60, AVG(s.totalLength)*8
FROM streams.iptcp s [RANGE BY '1 minute' SLIDE BY '1 second']
```

We see that for each window calculation[1], we count the number of packets each second by dividing by 60. We multiply the **totalLength** with 8 to get bits instead of bytes. The network load is estimated by finding the average total length of the tuples presented in the window. The query returns two attributes that show the average per second over the last minute. Alternatively, this query can be stated by using sub-queries, but because of space limitations, we do not discuss this solution for TelegraphCQ here.

The STREAM query is syntactically similar. We have chosen to split up this query into two queries, to show how they express network load in bytes per second (left) and packets per minute (right):

```
Load(bytesPerSec):
RSTREAM(SELECT SUM(totalLength)
        FROM S [RANGE 1 SECOND])

RSTREAM(SELECT AVG(bytesPerSec)
FROM Load [RANGE 10 SECONDS])
```

```
Load(packetsPerMinute):
RSTREAM(SELECT COUNT(*)
        FROM S [RANGE 1 MINUTE])

RSTREAM(SELECT AVG(packetsPerMinute)
FROM Load [RANGE 10 SECONDS])
```

Both queries use a sub-query to perform a pre-calculation on the tuples before sending the results to the main query. For ISPs, it would probably be interesting only to be notified when the load exceeds certain levels. This can be added as a `WHERE`-clause, stating `AVG(bytesPerSec) > 30 Gbits`, for example.

2.2 Task 2: Number of Packets Sent to Certain Ports During the Last n Minutes

In this task, we need to find out how the destination ports are distributed on the computers in the network, and count the number of packets that head for each of these ports. The number of packets are calculated over an n minutes long window.

This is an important feature, since there might be situations where there is a need for joining stream attributes with stored information. In network monitoring, this task can be used to show the port distribution of packets on a Web-server. Additionally, it is possible to re-state the task to focus on machines in a network instead of ports, i.e., how many packets have been sent to certain *machines* during the last n minutes. To solve this, we can look at the distribution of destination IP addresses instead of ports. For instance, if one Web-server seems to be very active in the network, a proxy might need to be installed to balance the load.

We choose to extend the query by joining the data stream with a table that consists of 65536 tuples; one for each available port. We do this to show the joining possibilities in a DSMS. Thus, prior to the experiments, we create the table *ports* that contains port numbers, as integers, from 0 to 65535, in both DSMSs. We only state the TelegraphCQ query here:

```
SELECT wtime(*), streams.iptcp.destPort, COUNT(*)
FROM streams.iptcp [RANGE BY '5 minutes' SLIDE BY '1 second'], ports
WHERE ports.port = streams.iptcp.destPort
GROUP BY streams.iptcp.destPort
```

`wtime(*)` is a TelegraphCQ specific function that returns the timestamp of the current window. The STREAM query is almost equal, except that it has to project the `destPort` tuples in a sub-query before the join. This is because STREAM only allows 20 attributes for aggregations, something which is statically defined in the code, and that we choose not to change in our experiments.

2.3 Task 3: Number of Bytes Exchanged for Each Connection During the Last m Seconds

When a connection between two machines is established, we are interested in identifying the number of bytes that are sent between them. In order to manage this, we have to identify a TCP connection

[1] `RANGE BY` describes the window range while `SLIDE BY` describes the update interval.

establishment, and sum up the number of bytes for each of these connections. As stated above, this task is only making a qualitative comparison for both query results, and we outline the ideas of how to model this query.

In TCP, connections are established using a 3-way handshake [19], and it is sufficient to identify an established connection with a tuple consisting of source and destination IP addresses and ports. For expressing this task, we first need to identify all three packets that play a role in the handshake. The most important attributes to investigate are the SYN and ACK control fields, as well as the sequence number and the acknowledgment number. To achieve this, we state two sub-queries. One selects TCP headers that have the SYN option field set, while the other selects the headers having the SYN *and* ACK option fields set. In a third query, we join these headers with headers having the ACK option field set as well as joining on matching sequence and acknowledgment numbers. In the 3-way handshake, timeouts are used in order to handle lost packets and re-transmissions. We model this using sliding windows. If a tuple slides out, a timeout has been violated. To identify the number of bytes that are exchanged on a connection, we have to join all arriving tuples with one of the connection tuples. This means that the connection tuples have to be stored for a limited period of time, for example in a window. We can also construct queries that, in a similar way as the ones described above, identify the connection tear down.

A sub-task would be to identify all the connection establishments that have not joined the final ACK packet, and therefore times out. In SYN-flood attacks this timeout can be exploited by a client by not sending the final ACK to the server after sending the initial SYN packet [12]. This behavior is important to identify, and is also an important task for IDSs. In this case, when we use windows to model timeouts, we want to identify the tuples that are deleted from the window. Here, STREAM has the possibility of defining a `DSTREAM`, i.e., a *delete-stream*, which satisfies this requirement.

What is interesting in this task, is that it needs to obtain information from all the packets to function correctly. For instance, sampling of only important tuples can not be used, because the three packets depend on each other in both control fields and sequence and acknowledgment numbers. Thus, the sampler becomes a query processor itself, something that only pushes the problem one level closer to the data stream.

3 Experiment Setup

In this section, we show the design of our experiment setup for our practical evaluation of DSMSs for network monitoring (see Figure 1).

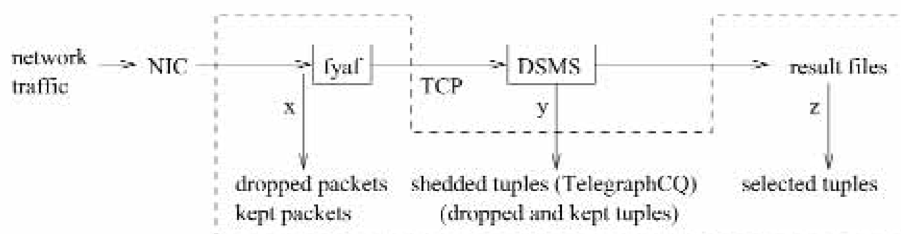


Fig. 1. The experiment setup.

We have developed a program called `fyaf`, which reads network packets from the network interface card (NIC). For packet capturing, `fyaf` uses the `pcap` library, which reports the number of kept and dropped packets after the capturing is completed. This is shown by the arrow denoted x in the figure. Kept packets are denoted x_k, while dropped packets are denoted x_d. Both TelegraphCQ and STREAM process data in a comma separated value (CSV) fashion, and `fyaf` is used to

transform the TCP/IP headers to the current DSMS's CSV stream scheme. `fyaf` also removes the payload from the packets, and sends the headers as tuples to the DSMS in the same frequency as they arrive. `fyaf` sends the tuples to the DSMS over a TCP socket, to guarantee that all tuples arrive. For this purpose, we have to change parts of STREAM so that it manages to accept socket requests. The advantage of using a local TCP socket is that possible delays in the DSMS affects the performance of `fyaf`, and thus, the packet capturing mechanism. In addition, TelegraphCQ reports the number y of kept and dropped tuples, denoted y_k and y_d respectively. In practice, TelegraphCQ's shedded tuples are reported as streams that can be queried as shadow queries of the task query [22]. The result tuples from the DSMS, z, are stored to file. An invariant is that systems that provide load shedding should have $x_d = 0$, while a system that does not support load shedding should have a varying number of dropped packets depending on queries and network load. A packet that is dropped from the NIC can not be turned into a tuple, so we consider that tuple as dropped.

The experiment setup consists of only two computers; A and B, which form their own small network. Each computer has two 3 GHz Intel Pentium 4 processors with 1 GB memory, and runs with Linux SuSE 9.2. The experiment setup and the DSMSs are located at node B. We have chosen to use two metrics: *relative throughput* and *accuracy*. We define these two metrics, as well as the term *network load*, to avoid misunderstandings.

Definition 1 Network load *is specified by the number of observed bits per second on a link.*

Definition 2 *The* relative throughput *of node N, denoted RT_N, is the relation between the network load node N receives and the network load it manages to handle. (In this paper, we use both terms "relative throughput" and "RT" for the relative throughput.)*

Definition 3 *The* accuracy *is measured as percentage of the computed result related to the correct result.*

4 Experiments and Results

In this section, we show the factors we use to verify that the experiment setup works and to run the tasks. After a practical verification, we show the results from Task 1 and Task 2. The experiments investigate the DSMSs' behavior at varying network load. The load is generated by the public domain traffic generator TG [16], which we set to send a constant rate of TCP packets with segment size of 576 bytes. We keep the rate constant to see how the DSMS behaves over time. This contradicts a real world scenario where load may change over a 24 hours period. However, we want to initiate the load shedding of TelegraphCQ, i.e., start load shedding instead of adapting to the stream, which is one of TelegraphCQ's strengths [2, 20, 15]. The traffic is generated over a time period called a *run*. Mainly, the network load is measured at 1, 2.5, 5, 7.5, and 10 Mbits/s, and for each network load, we perform 5 runs. This implies that `fyaf` has to compute data at a higher network load than the number of bits received on the NIC, to avoid becoming a bottleneck. We also assume that `fyaf` performs faster than the current DSMS since `fyaf` is less complex. This is verified by `fyaf`'s log files. All results are average values over the given number of runs. `fyaf` is set only to send the TCP/IP packets to the DSMS as tuples. Other packets, e.g. control packets in the network, are written to file for further analysis and identification.

The STREAM prototype does not support load shedding, and we use this system to investigate if it is sufficient to use the experiment setup for addressing the number of dropped tuples.

We define one query for this test, $Q1$ – `SELECT * FROM S`. $Q1$ simply projects all the tuples that are sent to the DSMS. This makes it easy to verify that the number of tuples sent from `fyaf` and number of tuples returned from $Q1$ are equal, i.e., $x_k = z$ in Figure 1. If this is the case, we can say that the relative throughput of STREAM is equal to the relation between dropped and kept packets from the NIC, i.e., $RT_{STREAM} = \frac{x_k}{x_d + x_k}$.

In TelegraphCQ, a separate wrapper process, the *wrapper clearing house* (WCH), is responsible for transforming the arriving tuples so that they are understood by the continuous query handling process; the *back end* (BE) process. The WCH aims to receive and obtain tuples from one or more external sources [13], which is a necessity for TelegraphCQ to handle real-time streams from several sources. The information about the shedded tuples can be accessed by using *shadow queries*. Thus, the accuracy metric can be used to evaluate TelegraphCQ's load shedding. TelegraphCQ provides several load shedding techniques [21, 22], but for simplicity, we only look at the number of dropped and kept tuples. By adding these two numbers together, we get a good estimate of the total number of tuples TelegraphCQ receives each second.

Since TelegraphCQ uses load shedding, we have the possibility of using the experiment setup to investigate the correctness of the load shedding functionality. The idea is simple: We know the number of tuples sent to TelegraphCQ, x_k, and the number it returns, z. The difference between those two results should be equal to the number of tuples dropped by the shedder, i.e., $y_d = |x_k - z|$. TelegraphCQ's relative throughput is denoted RT_{tcq}, which means that $RT_{tcq} = \frac{y_k}{y_d + y_k}$.

For investigating RT_{tcq}, we have chosen to use a query that is equal to $Q1$. By using such a query, we can investigate the correctness of the load shedding, since all tuples are selected:

```
SELECT <X>
FROM streams.iptcp
```

We vary the number of attributes, to see if this gives different results. Thus, in this query, `<X>` can be either "`*`", "`sourceip, destip, sourceport, destport`", or "`destip`". We can also observe how many tuples TelegraphCQ drops while the queries run. We execute the queries over streams with duration of 60 seconds and vary the network load.

Following are the results from these experiments. For STREAM, we investigate the result files from $Q1$, where we observe a small deviation from the expected results. We observe that $z < x_k$. `fyaf`'s log files reveal that ARP [18] packets compose a small part of the traffic. `fyaf` only sends TCP/IP headers to the DSMS and filters off other packets, like ARP. Since we use these statistics to evaluate the results from our experiments, we need to investigate the margin of error composed by the ARP packets. We calculate the margin of error by comparing the number of ARP packets to the total number of packets captured by `fyaf`. We observe that the margin of error is negligible, as it is always less than 0.025 %. Consequently, we consider this margin of error as insignificant. Thus, we conclude that the experiment setup can be used to measure the number of dropped packets.

After running the TelegraphCQ queries, we observe that the relative throughput decreases as the network load increases above 5 Mbits/s, and we observe - as expected - that the task projecting "`*`" has a lower relative throughput than the other two. At 20 Mbits/s we observe that the relative throughput is down to 0.1 for all three queries. Up to 20 Mbits/s, we see that $\frac{x_k}{x_d + x_k} = 0.999$. On higher network loads, the experiment setup starts to drop a significant amount of tuples. Consequentially, we have a maximum of 20 Mbits/s to run the TelegraphCQ queries on. The remaining 0.001 of dropped tuples from the experiment setup are interesting and will be subject for future work.

Following, we discuss the correctness of TelegraphCQ's load shedding functionality. We set RT_{es} to be the relative throughput that is expected from TelegraphCQ based on the data we obtain from the experiment setup, i.e., $RT_{es} = \frac{z}{x_k}$. For correct results, we have that $RT_{es} = RT_{tcq}$. The results from running $Q1$ shows that TelegraphCQ seems to report a higher relative throughput than what seems to be correct, which means that $RT_{tcq} > RT_{es}$. In Figure 2, we show the relative error by calculating $\frac{RT_{es}}{RT_{tcq}}$. What is interesting, is that the relative error does not decrease as the network load increases. Instead, we have a minimum at 10 Mbits/s before all three query results seem to converge to 1. At the current time, we can not explain this behavior, as all known sources of errors in our experiment setup have been investigated. To investigate this even further is topic of ongoing work. Thus, we see that TelegraphCQ seems to report a more optimistic load shedding than what is true when it comes to tuple dropping. Based on these results, we conclude that the experiment setup can be used for reporting dropped packets for DSMSs that do not have this functionality

© Revue MODULAD, 2007

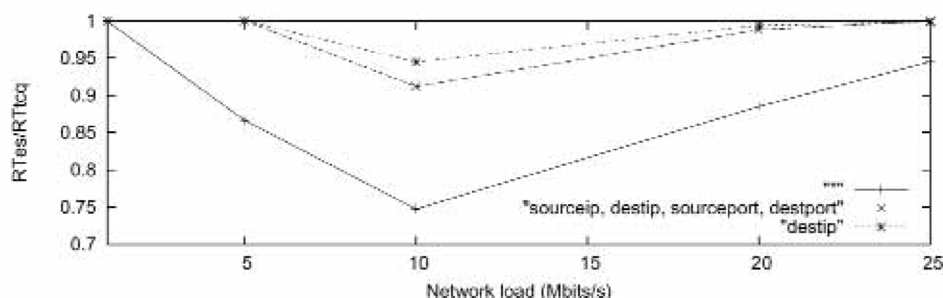


Fig. 2. Relative error in RT_{tcq}.

implemented. Since this is possible, we use this experiment setup to investigate the correctness of the reported load shedding, and we assume that TelegraphCQ does not report a correct number.

4.1 Task 1

For Task 1, we continue investigating TelegraphCQ's accuracy in the query results. For Task 2, we show the difference between STREAM and TelegraphCQ, as well as discussing the behavior of TelegraphCQ in a one hour run. For Task 1 and Task 2, each run lasts for 15 minutes, and the upper limit for the network is 10 Mbits/s.

Based on the findings from the prior experiments, we wish to continue investigating TelegraphCQ's accuracy. Instead of using RT_{cs}, as we did above, we now use the real network load for correcting. We have verified that TG's network load is correct.

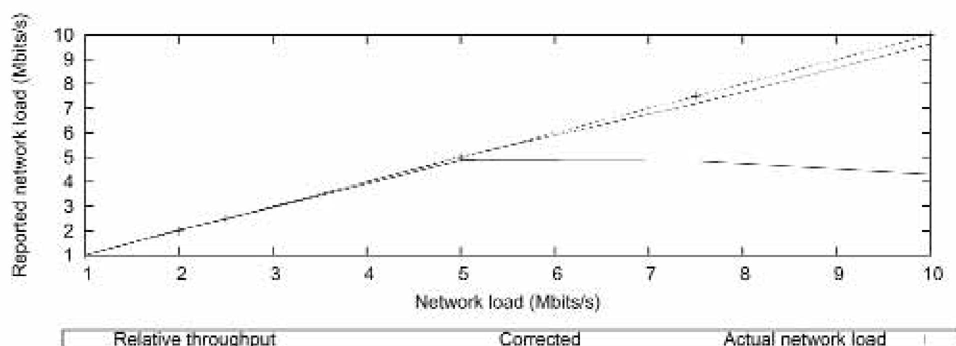


Fig. 3. Relative throughput as reported and corrected.

As Task 1 aims to measure the number of packets and network load, it is easy to estimate the correctness of the results. Figure 3 shows the correct network load as the cross dashed diagonal line. The relative throughput is decreasing just after 2 Mbits/s and is reduced considerably when the network load is 10 Mbits/s. We use the information from the query to correct the output by adding the shedded tuples and multiply with the average packet size. We see that after 5 Mbits/s, the corrected results start to deviate from the actual network load. At 10 Mbits/s, the correctness is 95.1 %. This is expected and coincide with what we observed in the prior experiment. Since the maximum network load is 10 Mbits/s, we do not know how accurate these results are at higher speeds, but we should observe that the deviation is reduced as the network load increases.

© Revue MODULAD, 2007

4.2 Task 2

For Task 2, we start by comparing the relative throughput from the two DSMSs. RT_{tcq} is computed by calculating the shedded tuples from TelegraphCQ, while RT_{STREAM} is computed by counting the number of dropped packets from the NIC. Since one packet is equal to one tuple, and that we have shown that it is possible to use our experiment setup for this purpose, it is interesting to look at the differences between the two systems. Figure 4 shows the relative throughput for the two systems.

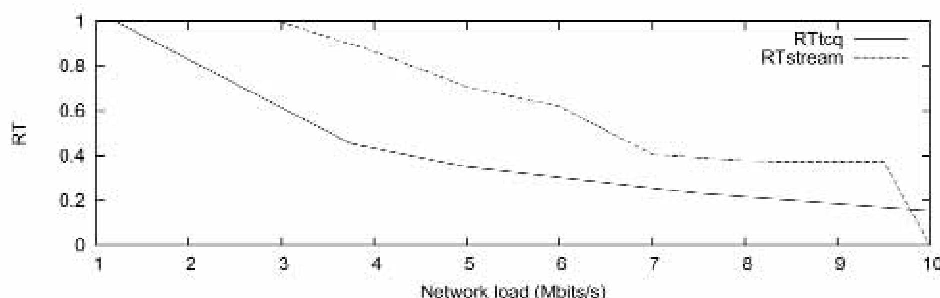


Fig. 4. A comparison between TelegraphCQ and STREAM.

We see that STREAM manages to compute more tuples than TelegraphCQ does, but that both systems have a significantly lower relative throughput when the network load increases to 10 Mbits/s. In our experiments, STREAM did not have any correct runs at 10 Mbits/s, thus, the relative throughput is set to 0. An alternative way of comparing between the two systems, is to turn off TelegraphCQ's load shedding, something that is possible, but we have chosen not to do this in our experiments.

Because of space limitation, we do not discuss the comparison between the two systems any deeper. Instead, we focus on looking at the relative throughput and results from TelegraphCQ's in Task 2.

One interesting issue with investigating streams of data, is to see how the systems behave over a time interval. In Figure 5, we observe TelegraphCQ's average relative throughput over 5 Mbits/s runs that lasted for one hour. The upper graph shows the plot of the relative throughput, while the lower graph shows a plot of the number of packets that are destined for the open port at machine B. We see that the output does not start until after the window is filled. What is interesting to see is that the relative throughput drops significantly after the first five minutes. This also implicate that TelegraphCQ waits until the first window is filled up before starting the aggregations and outputs. Note that five minutes of TCP/IP headers is much data to compute. We also see that the relative throughput is only approximately 0.8 to begin with, meaning that TelegraphCQ already is under overload before starting the calculations.

A consequence of the sudden drop at five minutes is that the result starts to decrease dramatically. For example, after approximately ten minutes of low relative throughput, the number of result tuples reaches zero. But after one hour, we see that the two processes cooperate well, and the plot shows a behavior that looks like a harmonic reduction. One possible solution is that we observe that since the BE has few tuples to process, the WCH is allowed to drop fewer tuples, something which results in more tuples to process, and thus, lower relative throughput. This mutual fight for resources converges to a final relative throughput. As future work, we will look at the possibilities of even longer runs, to see if this equilibrium is stable.

© Revue MODULAD, 2007

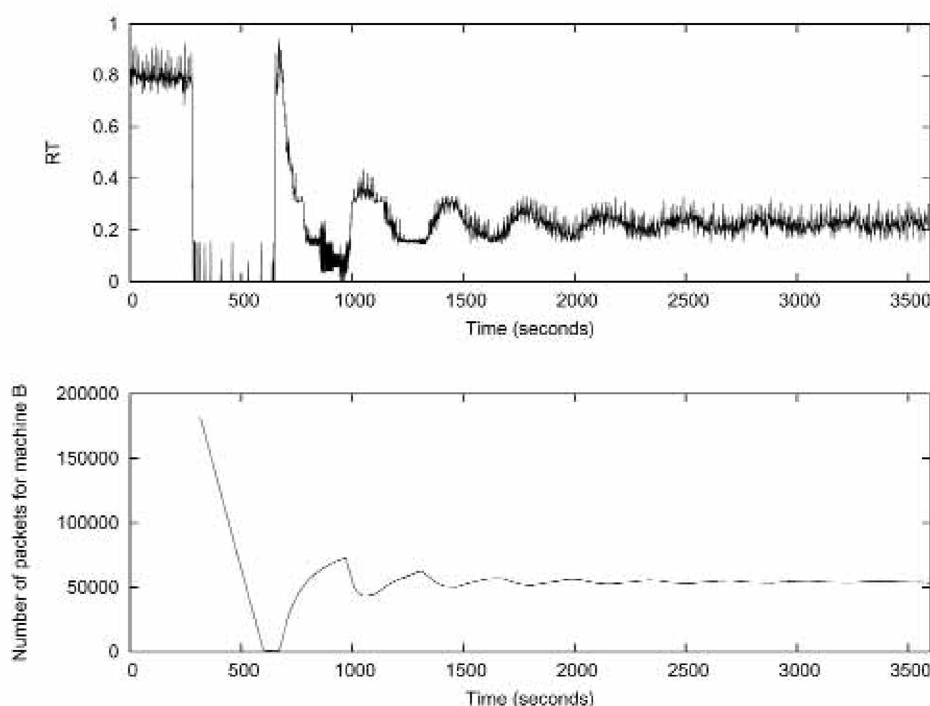


Fig. 5. The relative throughput and number of packets from a query lasting for an hour.

5 Conclusions

In this paper, we perform a practical evaluation of load shedding in DSMSs for network monitoring. We argue that DSMSs can be used for this application by referring to related work, as well as stating a set of network monitoring tasks and suggesting queries that solve these tasks. We also investigate and outline a solution for a more complex task, which aims for identifying TCP connection establishments.

For the practical evaluation, we suggest a simple experiment setup that can be used to add tuple dropping in DSMSs that do not support load shedding, as well as evaluating the tuple dropping accuracy in DSMSs that support this functionality. We also suggest a metric; *relative throughput*, which indicates the relation between the number of bits a node receives and number of bits the node is able to compute. By running simple projection tasks on two DSMSs - STREAM and TelegraphCQ - we verify that the experiment setup works as intended. We also observe that TelegraphCQ seems to report higher relative throughput than what is actually reached. Then, we evaluate the two first network monitoring tasks, and show TelegraphCQ's correctness and a comparison between STREAM and TelegraphCQ's relative throughput. Finally, we discuss the results from an one-hour run of TelegraphCQ.

Our conclusion is that load shedding is a very important functionality in DSMSs, as the network load often is higher than what the system manages to handle. We also conclude that network monitoring is an important application for DSMSs and that our experiment setup is useful for evaluating the DSMSs in network monitoring tasks. We argue for this by showing some results from our experiments. We see that our simple experiment setup can be used to investigate the correctness of tuple dropping in DSMSs.

Since network monitoring is a promising application for DSMSs, our future work is to investigate the expressiveness of the Borealis stream processing engine as a network monitoring tool, as well as evaluating its performance in our experiment setup. We will also extend our experiment setup to evaluate the correctness of load shedding techniques such as *sampling* and *wavelets*.

© Revue MODULAD, 2007

References

1. A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom. Stream: The stanford data stream management system. *Department of Computer Science, Stanford University*, 2004.
2. R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, 2000.
3. S. Babu, L. Subramanian, and J. Widom. A data stream management system for network traffic management. In *Proceedings of the 2001 Workshop on Network-Related Data Management (NRDM 2001)*, May 2001.
4. S. Babu and J. Widom. Continuous queries over data streams. *SIGMOD Rec.*, 30(3):109–120, 2001.
5. S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. *Proceedings of the 2003 CIDR Conference*, 2003.
6. C. Cranor, Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spataschek. Gigascope: High performance network monitoring with an sql interface. In *Proceedings of the 21st ACM SIGMOD International Conference on Management of Data / Principles of Database Systems*, June 2002.
7. C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk. Gigascope: a stream database for network applications. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 647–651, New York, NY, USA, 2003. ACM Press.
8. N. Duffield. Sampling for passive internet measurement: A review. *Statistical Science*, 19(3):472–498, 2004.
9. N. Duffield, C. Lund, and M. Thorup. Learn more, sample less: control of volume and variance in network measurement. *IEEE Transactions in Information Theory*, 51(5):1756–1775, 2005.
10. L. Golab and M. T. zsu. Issues in data stream management. *SIGMOD Rec.*, 32(2):5–14, 2003.
11. P. Huang, A. Feldmann, and W. Willinger. A non-instrusive, wavelet-based approach to detecting network performance problems. In *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 213–227, New York, NY, USA, 2001. ACM Press.
12. T. Johnson, S. Muthukrishnan, O. Spatscheck, and D. Srivastava. Streams, security and scalability. In *Proceeding of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec 2005)*, August 2005.
13. S. Krishnamurthy, S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. R. Madden, F. Reiss, and M. A. Shah. Telegraphcq: An architectual status report. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 2003.
14. W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Trans. Netw.*, 2(1):1–15, 1994.
15. S. Madden, M. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive contiuous queries over streams. *ACM SIGMOND*, June 2002.
16. P. E. McKenney, D. Y. Lee, and B. A. Denny. *Traffic generator release notes*, 2002.
17. T. Plagemann, V. Goebel, A. Bergamini, G. Tolu, G. Urvoy-Keller, and E. W. Biersack. Using data stream management systems for traffic analysis - a case study. April 2004.
18. D. Plummer. Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware. RFC 826 (Standard), November 1982.
19. J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFC 3168.
20. V. Raman, A. Deshpande, and J. M. Hellerstein. Using state modules for adaptive query processing. *Report No. UCB/CSD-03-1231*, February 2003.
21. F. Reiss and J. M. Hellerstein. Data triage: An adaptive architecture for load shedding in telegraphcq. Technical report, February 2004.
22. F. Reiss and J. M. Hellerstein. Declarative network monitoring with an underprovisioned query processor. In *The 22nd International Conference on Data Engineering*, April 2006.
23. A. Soule, K. Salamatia, N. Taft, R. Emilion, and K. Papagiannaki. Flow classification by histograms: or how to go on safari in the internet. In *SIGMETRICS '04/Performance '04: Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 49–60, New York, NY, USA, 2004. ACM Press.

© Revue MODULAD, 2007

Using Data Stream Management Systems to analyze Electric Power Consumption Data

Talel Abdessalem[1], Raja Chiky[1], Georges Hébrail[1,2], Jean Louis Vitti[1]

[1] ENST Paris - CNRS UMR 5141, 46, Rue Barrault, 75013 Paris Cedex 13, France
Email: {abdessalem, chiky, hebrail, vitti}@enst.fr
[2] EDF R&D - Département ICAME, 1, Av. du Général de Gaulle, 92140 Clamart, France
Email: georges.hebrail@edf.fr

Abstract. With the development of AMM (Automatic Metering Management), it will be possible for electric power suppliers to acquire from the customers their electric power consumption up to every second. This will generate data arriving in multiple, continuous, rapid, and time-varying data streams. Data Stream Management Systems (DSMS) - currently available as prototypes - aim at facilitating the management of such data streams. This paper describes an experimental study which analyzes the advantages and limitations of using a DSMS for the management of electric power consumption data.

Keywords: Electric power consumption, Data Stream Management System, electric load curve, continuous queries.

1 Introduction and motivation

The forthcoming deployment of AMM (Automatic Metering Management) infrastructures in Europe will enable a more accurate observation of a large number of customers. Electric power consumption will be possibly measured at a rate up to one index per second. These measures are useful for operations such as billing, data aggregation and consumption control. A traditional database system (DBMS Data Base Management System) could be used to manage this information. However, methods used currently in DBMS are not adapted to data generated from AMM in a streaming way. AMM generates an overwhelming amount of data arriving in multiple, continuous, rapid, and time-varying data streams. Several Data Stream Management Systems (DSMS) have been developed these last years to meet these needs. The role of these systems is to process in real time one or more data streams using *continuous* queries.
We performed an experimental study, on two public domain and general purpose DSMS prototypes (STREAM and TelegraphCQ), to analyze the advantages and limits of using such a system. Some typical queries of electric power consumption analysis were defined: experiments and results are reported in this paper.
The paper is organized as follows. Section 2 gives a brief introduction to DSMS's in general and presents in particular STREAM and TelegraphCQ. Section 3 presents the experimental study, and results are reported in Section 4. In Section 5, we conclude this paper and give an outlook to our ongoing and future research in this area.

2 Data Stream management Systems

Data Stream Management Systems [3] are designed to perform continuous queries over data stream. Data elements arrive on-line and stay only for a limited time period in memory. In a DSMS, continuous queries evaluate continuously and incrementally arriving data elements. DSMS use windowing technics to handle some operations like aggregation as only an excerpt of a stream (window) is of interest at any given time. A window may be physically defined in terms of a time interval (for instance the last week), or logically defined in terms of the number of tuples (for example the last 20 elements).

Several DSMS prototypes have been developed within the last five years. Some of them are specialized in a particular domain (sensor monitoring, web application, ...), some others are for general use (as STREAM [2] and TelegraphCQ [4]).

STREAM

STREAM (STanford stREam datA Management) is a prototype implementation of a DSMS developed at Stanford University [2]. This system allows the application of a great number of declarative and continuous queries to static data (tables) and dynamic data (streams). A declarative query language CQL (Continuous Query Language), derived from SQL, is implemented to process continuous queries. STREAM uses the concept of sliding windows over streams. These windows are of three types: time based window ([range T] where T is a temporal interval), tuple based window ([Row N] where N is a number of tuples), and partitioned sliding window ([Partition By $A_1, ..., A_k$ Rows N]) similar to Group BY in SQL.

TelegraphCQ

TelegraphCQ is a DSMS developed at University of Berkeley [4]. TelegraphCQ is built as an extension to the PostGreSQL relational DBMS to support continuous queries over data streams. The format of a data stream is defined as any PostGreSQL table in the PostgreSQL's Data Definition Language, and created using CREATE STREAM. TelegraphCQ is a mode of PostGreSQL execution. It supports the creation of archived and unarchived streams that are fed with external sources using stream specific wrapper. Queries that contain streams support the SELECT syntax and include extra predicates RANGE BY, SLIDE BY and START AT to specify the window size for stream operations. Each stream has a special time attribute that TelegraphCQ uses as the tuples timestamp for windowed operations.

3 Experiments

We installed STREAM and TelegraphCQ V2.1, and tested them on a data file of indices of electric power consumption measured during 150 days. These indices were obtained from three electric meters in different geographical cities, and our experiments can be extended to a larger number of meters. Each meter sends a tuple every 2 seconds. Each tuple is composed of a set of attributes such as meter identifier, time, date, index shown by the meter, and some additional information. The difference between two indices at two different instants gives the electric power consumption between these two instants.
For all queries, we assume a unique input stream Stream_index merging the streams of the three meters and defined as follows:

```
Stream_index (year INT, month INT, day INT, h INT, m INT, sec INT, meter char(12), index INT)
```

Stream_index is the name identifying the data stream. The date is specified with three attributes: 'month','day','year' ; and time by three other attributes: 'h', 'm', 'sec' (hour, minute, second). The 'meter' attribute identifies the meter and 'index' indicates the measured index (in kWh).
Electric power consumption analysis requires continuous queries over stream. Among the possible queries, we study here three queries which are representative of this kind of application.

Q1. Consumption of the last 5 minutes -minute by minute- grouped by meter, or by city.

- STREAM: a CQL query can be assigned a name to allow its result to be referenced by other queries. This feature (similar to *view* definition in SQL) allows us to express subqueries which are not allowed in CQL. We solve query *Q1* using the following steps:

© Revue MODULAD, 2007

1. Min_index: compute the minimum index grouped by meter and minute
```
SELECT      year, month, day, h, m, meter, min(index) as minindex
FROM        Stream_index
GROUP BY year, month, day, h, m, meter;
```

2. Cons_Minute_60 : compute the consumption from hh:59 to hh+1:00 (For example: from 07:59 to 08:00).
```
SELECT c1.year as year_beg, c1.month as month_beg, c1.day as day_beg, c1.h as h_beg,
       c1.m as m_beg, c1.meter, c1.minindex as minindex_beg, c2.year as year_end,
       c2.month as month_end,c2.day as day_end, c2.h as h_end, c2.m as m_end,
       c2.minindex as minindex_end, c2.minindex-c1.minindex as Cons
FROM   Min_index as c1, Min_index as c2
WHERE  c1.month=c2.month AND c1.day=c2.day AND c1.year=c2.year
AND    c1.h=(c2.h-1) AND c1.m=59 AND c1.meter=c2.meter AND c2.m=0
```

3. Cons_Minute_01_59: compute the consumption from the first minute and minute 59 of each hour (For example: from 07:00 to 07:59 minute by minute)
```
SELECT (similar to Cons_Minute_60 )
FROM   Min_index as c1, Min_index as c2
WHERE  c1.month=c2.month AND c1.day=c2.day AND c1.year=c2.year
AND    c1.h=c2.h AND c1.m=(c2.m-1) AND c2.m>0 AND c1.meter=c2.meter
```

4. Stream_Cons: set the union of the consumptions calculated by the two preceding queries as a single stream.
```
ISTREAM(Cons_Minute_60 UNION Cons_Minute_01_59);
```

5. Cons_per_M: compute the consumption for the last 5 minutes -minute by minute- grouped by meter.
```
SELECT year_beg, month_beg, day_beg, h_beg, m_beg, Stream_Cons.meter, Tablecity.city,
       minindex_beg, year_end, month_end, day_end, h_end, m_end, minindex_end, Cons
FROM   Stream_Cons[range 150 seconds], Tablecity
WHERE  Stream_Cons.meter = Tablecity.meter;
```

   We join here Stream_Cons with TableCity (containing meters' identifiers and the cities where they are) to get the meter's city. We parameter a window by 150 second because a tuple arrives every 2 seconds, whereas it is indexed by a timestamp at each second.

6. The sum of consumptions grouped by city.
```
SELECT      year_beg, month_beg, day_beg, h_beg, m_beg, city, sum(Cons)
FROM        Cons_per_M
GROUP BY year_beg, month_beg, day_beg, h_beg, m_beg, city;
```

- TelegraphCQ: We created a stream Elec.stream and assigned a wrapper to this stream. Each tuple is indexed by a timestamp 'tcqtime' specifying the creation time of the tuple, which is assumed to be monotonically increasing with a format like 'YYYY-MM-DD hh:mm:ss'.
  To solve this query, we compute the minimum of the indices for each minute. Then, we make a selfjoin of the result obtained with a delay of one minute. Thanks to PostGreSQL's operators for date management, we can easily compute the consumption between two instants delayed by one minute.
  Nested queries are not supported by TelegraphCQ, but we can use the WITH construction as an alternative. Selfjoins are not allowed either, we were constrained to create two identical streams to join them. We created the first stream Elec.minstream1 with the minimum of indices by minute using a sliding window parameterized by an interval of 6 minutes, and a second identical stream Elec.minstream2 but windowed on 5 minutes. This enables us to make the difference

© Revue MODULAD, 2007

between these two streams with a delay of one minute to get the consumption during the 5 last minutes minute by minute.

```
CREATE STREAM Elec.minstream1 ( meter varchar(12), minindex INTEGER,
                                tcqtime TIMESTAMP TIMESTAMPCOLUMN )
                                TYPE UNARCHIVED;
CREATE STREAM Elec.minstream2 ( as Elec.minstream1 ) ;

WITH
Elec.minstream1 AS ( SELECT   meter,min(index),DATE_TRUNC('minute', tcqtime)
                     FROM     Elec.stream [RANGE BY '6 minutes' SLIDE BY '1 minute'
                              START AT '2003-12-04 07:50:00']
                     GROUP BY meter, DATE_TRUNC('minute', tcqtime) )
Elec.minstream2 AS ( SELECT ...  as Elec.minstream1 with RANGE BY 5 minutes )
(
SELECT f1.meter, f1.minindex, f1.tcqtime, f2.minindex, f2.minindex-f1.minindex, f2.tcqtime
FROM   Elec.minstream1 as f1 [RANGE BY '1 minute' SLIDE BY '1 minute'
       START AT '2003-12-04 07:50:00'],
       Elec.minstream2 as f2 [RANGE BY '1 minutes' SLIDE BY '1 minute'
       START AT '2003-12-04 07:50:00']
WHERE f1.meter = f2.meter AND f1.tcqtime = (f2.tcqtime - interval '1 minute'));
```

To get the sum of consumption by city, we transform the last query into a stream, we call it Elec.streamcons. Streams Elec.minstream1 and Elec.minstream2 are computed with sliding windows of 1 minute. We apply a join operation between Elec.streamcons and table TableCity windowed to 5 minutes.

Q2 - Historical consumption -minute by minute- grouped by meter, or by city, starting from a fixed point.

- STREAM: We apply the same queries as previously without windowing. At the last query (Cons_Per_M), we filter to keep only the dates and time exceeding the fixed starting point.
- TelegraphCQ: This query was easy to solve thanks to START AT construction. It filters the tuples starting from a fixed point. We followed the same steps as the preceding query.

Q3 - Alarm -hour by hour- at exceeding a 'normal' consumption depending on the temperature.

This query gives an alarm when a consumption exceeds 10% of a 'normal' consumption over a period from midnight to the current hour. The alarm is given after midday. 'Normal' consumption is given hour by hour in a table consNormal for a temperature of 20°C.

- STREAM: There is a basic deficiency in the STREAM prototype that makes impossible to solve this query. There are no operators for standard date format management. Indeed, to perform this query, we would have had to treat several cases: the passage from a day to the next (example: from 12/02/2004 23:00 to 13/02/2004 00:00), the passage from a month to the next (example: from 31/01/2004 23:00 to 01/02/2004 00:00), the passage from a year to the next,...
- TelegraphCQ: We use a stream Elec.temperature that gives the temperature recorded each hour during 150 days. We apply a join operator between table consNormal and stream Elec.temperature in order to build a stream of normal consumption Elec.streamconsnormal which depends on temperature (a gradient approach is used).

  To solve this query, we apply a sliding window parametered by a time interval of 24 hours to the Elec.stream consumption stream , we calculate the sum of consumption hour by hour during 24h, and we filter the consumption taken after midday. We follow the same steps for the stream of normal consumption to get a stream of normal consumption cumulated hour by hour from midday. Finally, we compare the two streams to filter consumption which exceeds of 10 % normal consumption. The query is not reported in this paper by lack of room.

© Revue MODULAD, 2007

4 Synthesis

As for the clarity and easiness of writing queries, TelegraphCQ is much better than STREAM, thanks to its ability of defining landmark physical windows combined with the date management facilities provided by the PostGreSQL embedded system. Moreover, TelegraphCQ allows to reuse results of queries as a stream. It allows also redirecting the output of queries into a file to be stored in a specified format. Queries can be added dynamically when others are being executed, which is not possible with STREAM.
STREAM is characterized by its graphical interface with a graphical query and system visualizer. In fact, it's important for users (administrators) to have the ability to inspect the system while it is running and to understand how the continuous queries are managed. For instance, it was possible to see in STREAM that the aggregation operations of query 1. were performed incrementally without keeping the whole stream in memory.
Comparing our expectations and requirements, we can state that the recent TelegraphCQ prototype is quite useful for online analysis of electrical power consumption data, mainly because we solved easily all our defined queries. Though the performance of TelegraphCQ appears to be much better on our queries than in STREAM, we would like to have an idea of the system performance and control the amount of memory being used to process our queries.

5 Conclusion and Future Work

We have shown that two DSMS's allows to treat individual electric power consumption data arriving in a continuous and fast rate stream. According to our experiments, it appears that TelegraphCQ is better adapted to our needs than STREAM. However, TelegraphCQ does not offer tools to view the structure of query plans, nor monitor system resources and inspect the amount of memory being used. The comparison between these two DSMS's is mainly functional and does not take into account system performance. A follow-up study is to analyze the performance of TelegraphCQ when scaling up to larger streams figuring a larger number of meters.
Some research teams work on distributed DSMS's, like the Borealis [1] proposal. Such distributed DSMS's should be more appropriate to deal with AMM data which are distributed in nature. We are going to test the Borealis system in the close future, in addition to a following of future releases of TelegraphCQ or commercial follow-up systems.

References

1. D.J.Abadi, Y.Ahmad, M.Balazinska, U.Cetintemel, M.Cherniack, J.H.Hwang, W.Lindner, A.S.Maskey, A.Rasin, E.Ryvkina, N.Tatbul, Y.Xing, and S.Zdonik (2005). The Design of the Borealis Stream Processing Engine. 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05), Asilomar, CA, January 2005.
2. Arasu A., Babcock B., Babu S., Cieslewicz J., Datar M., Ito K., Motwani R. , Srivastava U., Widom J.(2004). STREAM: The Stanford Data Stream Management System, Book chapter.
3. Babcock B., Babu S., Datar M., Motwani R., and Widom J. (2002). Models and issues in data stream systems. In Symposium on Principles of Database Systems, pages 116. ACM SIGACT-SIGMOD.
4. S.Chandrasekaran, O.Cooper, A.Deshpande, M.J.Franklin,J.M.Hellerstein, W.Hong, S.Krishnamurthy, S.R.Madden, V.Raman, F.Reiss, M.A.Shah (2003). TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. CIDR 2003.

© Revue MODULAD, 2007

Johannes Gehrke abstract

TUTORIAL

Processing Data Streams: You Only Get One Look

Continuous data streams arise naturally, for example, in the network installations of large Telecom and Internet service providers where detailed usage information from different parts of the network needs to be continuously collected and analyzed for interesting trends. This talk will provide an overview of the key research results surrounding the systems issues in data stream processing. I will overview recent results on data stream system architectures, query processing, languages for querying data streams, and complex event processing. I will conclude with some open research questions in these areas.

© Revue MODULAD, 2007

Rule Learning from Data Streams: an overview

Jesús S. Aguilar–Ruiz

School of Engineering
Pablo de Olavide University
Seville, Spain

Abstract. Classification is a very well studied task in data mining. In the last years, important works have been published to scale up classification algorithms in order to handle large datasets. However, due to the high rate of streams of data, a number of emerging applications are demanding new approaches. Rule learning is an efficient alternative to address non–stationary environments. The talk presents an overview of rule–based learning algorithms for data streams and emphasizes some important aspects of these techniques.

Keywords: Data Streams, Rule–based learning.

1 Introduction

The advances in hardware technology have paved the way for the development of algorithms that can process the real–time information at a rapid rate. Streams of data grow at an unlimited rate and traditional data mining algorithms cannot process them efficiently. In spite of the great increase of storage capacity, it is not even enough for hundreds or thousands of instances arriving per second. Nowadays, typical problems such as clustering, classification, frequent pattern mining, change detection or dimensionality reduction are being reconsidered in the realm of data streams. What was initially finite data is now infinite data, thus giving rise to many challenges in machine learning, data mining and statistics.

Classification and rule learning are important, well studied tasks in machine learning and data mining. Classification methods represent the set of supervised learning techniques where a target categorical variable is predicted based on a set of numerical or categorical input variables. A variety of methods such as decision trees, rule based methods, and neural networks are used for the classification problem. Most of these techniques have been designed to build classification models from static data sets, where several passes over the stored data are possible.

In order to classify and model large–scale databases, important works have been recently addressed to scale up inductive classifiers and learning algorithms. In environments where high–rate streams of detailed data are constantly generated, memory and time limitations make multi pass scalable algorithms unfeasible. Also, real–world data streams are not generated in stationary environments, requiring incremental learning approaches to track trends and adapt to changes in the target concept.

Furthermore, the classification process may require simultaneous model construction and testing in an environment which constantly evolves over time. However, within incremental learning, a whole training set is not available a priori as examples arrives over time, normally one at a time t and not time dependent necessarily (e.g., time series). Despite online learning systems continuously review, update, and improve the model, not every online system is based on an incremental approach.

2 Some aspects of learning from data streams

Formally, a data stream D can be defined as a sequence of examples (also called transactions or instances), $D=(e_1,e_2,\ldots,e_i,\ldots)$, where e_i is the i-th arrived example. To process and mine data streams, different window models are often used. A window is a subsequence between the i-th and j-th arrived examples, denoted as W[i,j]=$(e_i,e_{i+1},\ldots,e_j)$, $i < j$. There are three common models:

© Revue MODULAD, 2007

- Landmark window. The model computes examples from a starting e_i to the current e_t. If $i = 1$, then the model processes the entire data streams.
- Sliding window. This model is based on the size of the windows w, and then computes all the examples of the subsequence W[t-w+1,t], where e_t is the current example.
- Damped window. This is a variant that can be applied to the aforementioned models, as it assigns weights to examples in order to give more importance to more recent examples.

Ideally, a rule based system should not use *sampling*, and the window size should be one. Some important aspects of the design of incremental algorithms for data streams are:

- **Influence of the arriving:** both the order of examples, the arriving rate and the possible time–dependency are critical.
- **Adaptation to change:** knowledge obtained may not be useful in the future, so any change of tendency must be detected.
- **Learning curve:** at initial stages, the incremental model is not accurate enough.

3 Rule–based systems

In general, rule based classification algorithms for data streams are quite difficult to build, particularly when the window size is one, as the necessary statistics to maintain the performance are hard to keep and update. This factor is even more critical when concept drift is present in data.

3.1 AQ11 Algorithm

The first incremental algorithm based on set of rules, which are described in DNF. AQ11 does not store examples in memory, and relies on the learning process, i.e., on the decision rules obtained. For any new example covered by a rule with a different label from the one of the example, AQ11 refines such rules iteratively, until the wrongly classified example is not covered. Rules are newly generalized from new examples.

3.2 GEM Algorithm

The learning scheme is very similar to AQ11, although each process of generalization or specialization takes into account every example arrived until that moment. In practice, it is not an useful technique.

3.3 STAGGER Algorithm

The first incremental algorithm that provides decision rules, and designed to be robust in the presence of noise and able to deal with concept drift. STAGGER does not update the rules constantly, but only when there is a high level of inconsistency. STAGGER saves statistics instead of examples, and adopts a conservative policy in order to detect changes in the long time.

3.4 SCALLOP Algorithm

SCALLOP is a scalable classification algorithm for numerical data streams. The algorithm produces a set of decision rules that is constantly verified during the learning process, in order to maintain the tendency of data. The verification is a process that involves modification, updating and deleting of statistics to handle concept drift properly. One of the main features of this system is the use of several sliding windows.

© Revue MODULAD, 2007

4 Conclusions

The development of learning or classification systems for data streams is a very difficult task. There are many temporal and spatial constraints and the algorithm must provide accurate response at any time. Furthermore, if it is required the system to be descriptive, as decision rules for instance, then the level of complexity increases, since the updating of descriptive models takes longer.

References

1. Maloof, M., Michalski, R.: Incremental learning Artificial Intelligence 154 (2004).
2. Muthukrishnan, S.: Data streams: algorithms and applications. In: Proc. of The 14th annual ACMSIAM Symposium on Discrete Algorithms. (2003).
3. Jin, R., Agrawal, G.: Efficient decision tree construction on streaming data. In Proc. of The 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining KDD03, ACM Press (2003).
4. Wang, H., Fan, W., Yu, P., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In Proc. of The 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining KDD03, ACM Press (2003).

© Revue MODULAD, 2007

A New Support Measure for Items in Streams *

Toon Calders
Eindhoven University of Technology

Abstract

Mining streams is a challenging problem, because the data can only be looked at once, and only small summaries of the data can be stored. We present a new frequency measure for items in streams that does not rely on a fixed window length or a time-decaying factor. Based on the properties of the measure, an algorithm to compute it is shown. Experimental evaluation supports the claim that the new measure can be computed from a summary with very small memory requirements, that can be maintained and updated efficiently. In this extended abstract, the main points of the presentation are discussed.

1 Motivation

Mining frequent items over streams received recently a lot of attention. It presents interesting new challenges over traditional mining in static databases. It is assumed that the stream can only be scanned once, and hence if an item is passed, it can not be revisited, unless it is stored in main memory. Storing large parts of the stream, however, is not possible because the amount of data passing by is typically huge.

Different models have already been proposed in literature. The main characteristic is: how must the frequency of an item be measured? There are different types of models. (1) the sliding window model, (2) the time-fading model, or (3) the landmark model. In the sliding window model [1, 3, 6, 8, 10], only the most recent events are used to determine the frequency of an item. In order to avoid having to count the supports on this window all over again in every time point, the algorithm in fact updates the frequency of the items based on the deletion of some transactions and the insertion of other. In the time-fading model, the past is still considered important, but not as important as the present. This is modelled by gradually fading away the past [9]. That is, there is, e.g., a decay factor $d < 1$, and a timepoint that lies n timepoints in the past, only contributes d^n to the count. In addition to this mechanism, a tilted-time window can be introduced [4, 5]. In the *landmark* model, a particular time period is

*The presentation is based on material presented in the ECML/PKDD'06 workshop International Workshop on Knowledge Discovery from Data Streams (IWKDDS) [2] and is joint work with Nele Dexters and Bart Goethals of the University of Antwerp.

© Revue MODULAD, 2007

fixed, from the landmark designating the start of the system up till the current time [7, 8, 11]. The analysis of the stream is performed for only the part of the stream between the landmark and the current time instance.

Obviously, the landmark model is not very well suited to find current trends. The sliding window and the time-decaying models are more suitable; the sliding window method focusses solely on the present, while the time-decaying model still takes the past into account, although the effect fades away. For both the sliding window and the fading window approach, it is hard to determine the right parameter settings. Especially for the sliding window method, if the window length is set too high, interesting phenomena might get smoothed out. For example, suppose that the occurrence of an item a is cyclic; every month, in the beginning of the month, the frequency of a increases. If the length of the sliding window, however, is set to 1 month, this phenomenon will never be captured. In many applications it is not possible to fix a window length or a decay factor that is most appropriate for every item at every timepoint in an evolving stream.

2 Maximal Frequency Measure

Therefore, we propose a new frequency measure. We assume that on every timestamp, a new itemset arrives in the stream. We denote a stream as a sequence of itemsets; e.g., $\langle ab\ bc\ abc\rangle$ denotes the stream where at timestamp 1, the itemset ab arrives, at timestamp 2, bc, and at timestamp 3, abc. The *maximal frequency* measure for an itemset I in a stream S of length t, denoted $mfreq(a, S)$, is defined as the maximum of the frequency of the itemset over the time intervals $[s, t]$, with s any timepoint before t.

Example *We focus on target item a.*

$$\begin{aligned} mfreq(a, \langle a\ b\ a\ a\ a\ b\rangle) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{4}{6}\right) = \frac{3}{4}\ . \\ mfreq(a, \langle b\ c\ d\ a\ b\ c\ d\ a\rangle) &= \max\left(\frac{1}{1}, \cdots\right) = 1\ . \\ mfreq(a, \langle x\ a\ a\ x\ a\ a\ x\rangle) &= \max\left(\frac{0}{1}, \frac{1}{2}, \frac{2}{3}, \frac{2}{4}, \frac{3}{5}, \frac{4}{6}, \frac{4}{7}\right) = \frac{2}{3}\ . \end{aligned}$$

3 Algorithm

For this frequency measure, we present an incremental algorithm that maintains a small summary of relevant information of the history of the stream that allows to produce the current frequency of a specific item in the stream immediately at any time. That is, when a new itemset arrives, the summary is updated, and when at a certain point in time, the current frequency of an item is required, the result can be obtained instantly from the summary. The structure of the summary is based on some critical observations about the windows with the maximal frequency. In short, many points in the stream can never become the

© Revue MODULAD, 2007

starting point of a maximal window, no matter what the continuation of the stream will be.

Example In the following stream, the only positions that can ever become the starting point of a maximal interval for the singleton itemset a are indicated by vertical bars.

$$\langle |a\ a\ a\ b\ b\ b\ a\ b\ b\ a\ b\ a\ b\ a\ b\ a\ b\ b\ b\ b\ |a\ a\ b\ a\ b\ b\ |a\rangle$$

The summary will thus consist of some statistics about the few points in the stream that are still candidate starting points of a maximal window. These important points in the stream will be called the *borders*. More details can be found in [2].

4 Experimental Evaluation

Critical for the usefulness of the technique are the memory requirements of the summary that needs to be maintained in memory. We show experimentally that, even though in worst case the summary depends on the length of the stream, for realistic data distributions its size is extremely small. Obviously, this property is highly desirable as it allows for an efficient and effective computation of our new measure. Also note that our approach allows exact information as compared to many approximations considered in other works. In Figure 1, for some distributions, the maximal number of borders in synthetically generated random streams have been given. The number of borders corresponds linearly to the memory requirements of the algorithm.

5 Summary

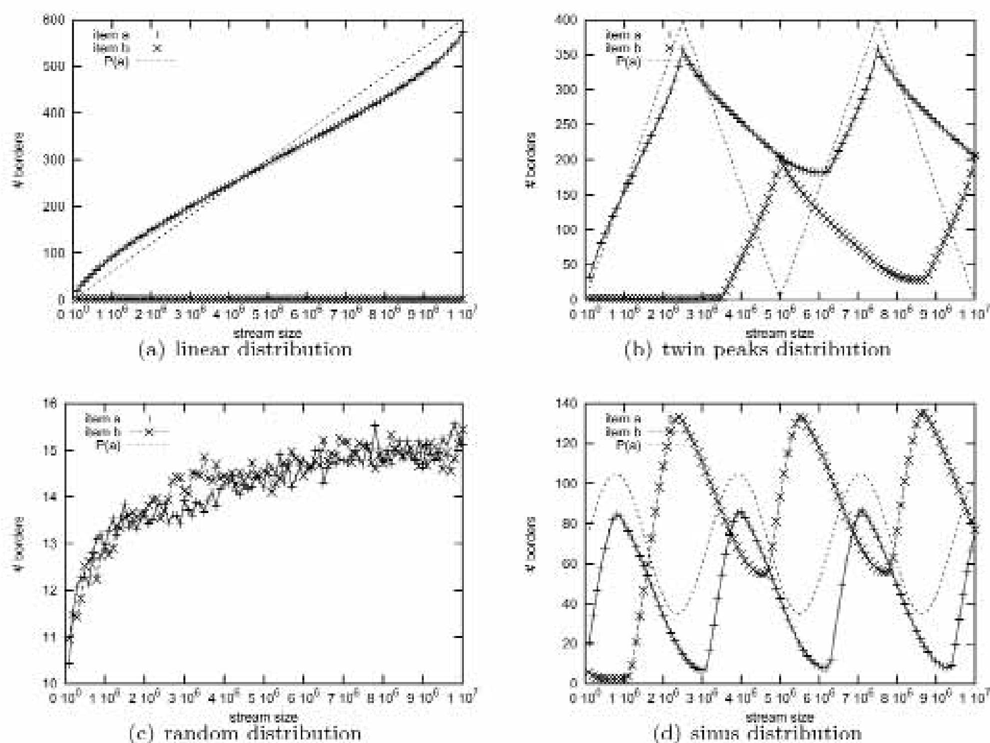
In the presentation, a new support measure for itemsets in streams is introduced and motivated. An algorithm to maintain a small summary based on which the support can immediately be produced is presented. Experimental evaluation shows that the memory requirements of this summary are very low.

References

[1] Ruoming J. and Agrawal G.: An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. in Proc. 5th IEEE Int. Conf. on Data Mining (ICDM'05), pp 210–217.

[2] Calders, T., Dexters, N., and Goethals, B.: Mining Frequent Items in a Stream Using Flexible Windows. In *Proc. of the ECML/PKDD-2006 International Workshop on Knowledge Discovery from Data Streams (IWKDDS)*; (2006)

© Revue MODULAD, 2007



(a) linear distribution (b) twin peaks distribution

(c) random distribution (d) sinus distribution

Figure 1: Size of the summaries for two items a and b

[3] Demaine E.D., Lopez-Ortiz A. and Munro, J.I.: Frequency Estimation of Internet Packet Streams with Limited Space. In *Proc. of the 10th Annual European Symposium on Algorithms*, pp 348–360. (2002)

[4] Giannella C., Han J., Robertson E. and Liu C.: Mining Frequent Itemsets Over Arbitrary Time Intervals in Data Streams. In *Technical Report TR587*, Indiana University, Bloomington. (2003)

[5] Giannella C., Han J., Pei J., Yan X. and Yu P.S.: Mining Frequent Patterns In Data Streams at Multiple Time Granularities. In H. Kargupta, A. Joshi, K. Sivakumar and Y. Yesha (eds), *Next Generation Data Mining*, pp 191 212. (2003)

[6] Golab L., DeHaan D.,Demaine E.D., Lopez-Ortiz A. and Munro J.I.: Identifying Frequent Items in Sliding Windows over On-Line Packet Streams. In *Proc. of the 1st ACM SIGCOMM Internet Measurement Conference*, pp 173 178. (2003)

© Revue MODULAD, 2007

[7] Jin R. and Agrawal G.: An Algorithm for In-Core Frequent Itemset Mining on Streaming Data. In *Proc. of the 5th IEEE International Conference on Data Mining (ICDM)*, pp 210–217. (2005)

[8] Karp, R. M., Papadimitriou, C. H. and Shenker, S.: A Simple Algorithm for Finding Frequent Elements in Streams and Bags. In *ACM Trans. on Database Systems* 28, pp 51–55. (2003)

[9] Lee, D. and Lee, W.: Finding Maximal Frequent Itemsets over Online Data Streams Adaptively. In *Proc. of the 5th IEEE International Conference on Data Mining (ICDM)*, pp 266–273. (2005)

[10] Lin C.-H., Chiu D.-Y., Wu Y.-H. and Chen A.L.P.: Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. In *Proc. SIAM International Conference on Data Mining*. (2005)

[11] Yu J.X., Chong Z., Lu H. and Zhou A.: False Positive or False Negative: Mining Frequent Items from High Speed Transactional Data Streams. In *Proc. of the 30th International Conference on Very Large Databases*, pp 204–215. (2004)

© Revue MODULAD, 2007

Mining Sequential Patterns from Data Streams: a Centroid Approach

Alice Marascu Florent Masseglia

INRIA Sophia Antipolis
2004 route des Lucioles - BP 93
06902 Sophia Antipolis, France
E-mail: {Alice.Marascu,Florent.Masseglia}@sophia.inria.fr

Abstract

In recent years, emerging applications introduced new constraints for data mining methods. These constraints are typical of a new kind of data: the *data streams*. In data stream processing, memory usage is restricted, new elements are generated continuously and have to be considered as fast as possible, no blocking operator can be performed and the data can be examined only once. At this time only a few methods has been proposed for mining sequential patterns in data streams. We argue that the main reason is the combinatory phenomenon related to sequential pattern mining. In this paper, we propose an algorithm based on sequences alignment for mining approximate sequential patterns in Web usage data streams. To meet the constraint of one scan, a greedy clustering algorithm associated to an alignment method is proposed. We will show that our proposal is able to extract relevant sequences with very low thresholds.

Keywords: data streams, sequential patterns, web usage mining, clustering, sequences alignment.

1 Introduction

The problem of mining sequential patterns from a large static database has been widely addressed [2, 11, 14, 18, 10]. The extracted relationship is known to be useful for various applications such as decision analysis, marketing, usage analysis, etc. In recent years, emerging applications such as network traffic analysis, intrusion and fraud detection, web clickstream mining or analysis of sensor data (to name a few), introduced new constraints for data mining methods. These constraints are typical of a new kind of data: the *data streams*. A data stream processing has to satisfy the following constraints: memory usage is restricted, new elements are generated continuously and have to be considered in a linear time, no blocking operator can be performed and the data can be examined only once. Hence, many methods have been proposed for mining items or patterns from data streams [6, 3, 5]. At first, the main

© Revue MODULAD, 2007

problem was to satisfy the constraints of the data stream environment and provide efficient methods for extracting patterns as fast as possible. For this purpose, approximation has been recognized as a key feature for mining data streams [7]. Then, recent methods [4, 8, 17] introduced different principles for managing the history of frequencies for the extracted patterns. The main idea is that people are often more interested in recent changes. [8] introduced the *logarithmic tilted time window* for storing patterns frequencies with a fine granularity for recent changes and a coarse granularity for long term changes. In [17] the frequencies are represented by a regression-based scheme and a particular technique is proposed for segment tuning and relaxation (merging old segments for saving main memory).
However, at this time only a few methods has been proposed for mining sequential patterns in data streams. We argue that the main reason is the combinatory phenomenon related to sequential pattern mining. Actually, if itemset mining relies on a finite set of possible results (the set of combinations between items recorded in the data) this is not the case of sequential patterns where the set of results is infinite. In fact, due to the temporal aspect of sequential patterns, an item can be repeated without limitation leading to an infinite number of potential frequent sequences. In a web usage pattern, for instance, numerous repetitions of requests for pdf or php files are usual.
In this paper, we propose the SMDS (Sequence Mining in Data Streams) algorithm, which is based on sequences alignment (such as [10, 9] have already proposed for static databases) for mining approximate sequential patterns in data streams. The goal of this paper is first to show that classic sequential pattern mining methods cannot be included in a data stream environment because of their complexity and then to propose a solution.
Our method, proposed in this paper, is able to perform several operations on the sequences of a batch, in only one scan. Those operations include a clustering of the sequences and an alignment of the sequences of each cluster.
The proposed algorithm is implemented and tested over a real dataset. Our data comes from the access log files of Inria Sophia-Antipolis. We will thus show the efficiency of our mining scheme for Web usage data streams, though our method might be applied to any kind of sequential data. Analyzing the behaviour of a Web site's users, also known as Web Usage Mining, is a research field, which consists of adapting the data mining methods to the records of access log files. These files collect data such as the IP address of the connected host, the requested URL, the date and other information regarding the navigation of the user. Web Usage Mining techniques provide knowledge about the behaviour of the users in order to extract relationships in the recorded data. Among available techniques, the sequential patterns are particularly well adapted to the log study. Extracting sequential patterns on a log file is supposed to provide this kind of relationship: "*On the Inria's Web Site, 10% of users visited consecutively the homepage, the available positions page, the ET*[1] *offers, the ET missions and finally the past ET competitive selection*". We

[1] ET: Engineers, Technicians

© Revue MODULAD, 2007

want to extract typical behaviours from clickstream data and show that our algorithm meets the time constraints in a data stream environment and can be included in a data stream process at a negligible cost.

The rest of this paper is organized as follows. The definitions of Sequential Pattern Mining and Web Usage Mining are given in Section 2. Section 3 gives an overview of two recent methods for extracting frequent patterns in data streams. The framework proposed in this paper is presented in Section 4 and empirical studies are conducted in Section 5.

2 Definitions

In this section we define the sequential pattern mining problem in large databases and give an illustration. Then we explain the goals and techniques of Web Usage Mining with sequential patterns.

2.1 Sequential Pattern Mining

The problem of mining sequential patterns from a static database DB is defined as follows [2]:

Definition 1 *Let $I = \{i_1, i_2, ..., i_k\}$, be a set of k literals (items). I is a k-itemset where k is the number of items in I. A sequence is an ordered list of itemsets denoted by $< s_1 s_2 \ldots s_n >$ where s_j is an itemset. The data-sequence of a customer c is the sequence in DB corresponding to customer c. A sequence $< a_1 a_2 \ldots a_n >$ is a subsequence of another sequence $< b_1 b_2 \ldots b_m >$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$.*

Example 1 *Let C be a client and S=$<$ (c) $(d\ e)$ (h) $>$ be that client's purchases. S means that "C bought item c, then he bought d and e at the same moment (i.e. in the same transaction) and finally bought item h".*

Definition 2 *The support of a sequence s, also called supp(s), is defined as the fraction of total data-sequences that contain s. If supp$(s) \geq minsupp$, with a minimum support value $minsupp$ given by the user, s is considered as a frequent sequential pattern.*

The problem of sequential pattern mining is thus to find all the frequent sequential patterns as stated in definition 2.

2.2 From Web Usage Mining to Data Stream Mining

For classic Web usage mining methods, the general idea is similar to the principle proposed in [12]. Raw data is collected in access log files by Web servers. Each input in the log file illustrates a request from a client machine to the server (*http daemon*).

© Revue MODULAD, 2007

Client	d1	d2	d3	d4	d5
1	a	c	d	b	c
2	a	c	b	f	c
3	a	g	c	b	c

Table 1: File obtained after a pre-processing step

Definition 3 *Let Log be a set of server access log entries. An entry g, $g \in Log$, is a tuple $g =< ip_g, ([l_1^g.URL, l_1^g.time] \ldots [l_m^g.URL, l_m^g.time]) >$ such that for $1 \leq k \leq m$, $l_k^g.URL$ is the item requested by the user g at time $l_k^g.time$ and for all $1 \leq j < k$, $l_k^g.time > l_j^g.time$.*

The structure of a log file, as described in definition 3, is close to the "`Client -Time-Item`" structure used by sequential pattern algorithms. In order to extract frequent behaviours from a log file, for each g in the log file, we first have to transform ip_g into a client number and for each record k in g, $l_k^g.time$ is transformed into a time number and $l_k^g.URL$ is transformed into an item number. Table 1 gives a file example obtained after that pre-processing. To each client corresponds a series of times and the URL requested by the client at each time. For instance, the client 2 requested the URL "f" at time $d4$. The goal is thus, according to definition 2 and by means of a data mining step, to find the sequential patterns in the file that can be considered as frequent. The result may be, for instance, <(a)(c)(b)(c)> (with the file illustrated in table 1 and a minimum support given by the user: 100%). Such a result, once mapped back into URLs, strengthens the discovery of a frequent behaviour, common to n users (with n the threshold given for the data mining process) and also gives the sequence of events composing that behaviour.
Nevertheless, most methods that were designed for mining patterns from access log files cannot be applied to a data stream coming from web usage data (such as clickstreams). In our context, we consider that large volumes of usage data are arriving at a rapid rate. Sequences of data elements are continuously generated and we aim at identifying representative behaviours. We assume that the mapping of URLs and clients as well as the data stream management are performed simultaneously. Furthermore, as stated by [13], sequential pattern extraction, when applied to Web access data, is effective only if the support is very low. A low support means long response time and the authors proposed a divisive approach to extract sequential patterns on similar navigations (in order to get highly significant patterns). Our goal with this work is close to that of [13] since we will provide a navigation clustering scheme designed to facilitate the discovery of interesting sequences, while meeting the needs for rapid execution times involved in data stream processing.

© Revue MODULAD, 2007

3 Related Work

In recent years, many contributions have been proposed for mining patterns in data streams [6, 3, 5, 8, 17, 19, 20]. [1, 15] also consider the problem of mining sequences in streaming data. In this section, we give an overview of [8] and [17].

3.1 FP-Streaming: Frequent Itemset Mining

The authors of [8] describe an approach based on a batch environment and introduce the FP-stream structure for storing frequent patterns and the evolution of their frequency. The authors propose to consider batches of transactions (the update is done only when enough incoming transactions have arrived to form a new batch). For each batch, the frequent patterns are extracted by means of the FP-Growth algorithm applied on a FP-tree structure representing the sequences of the batch. Once the frequent patterns are extracted, the FP-stream structure stores the frequent patterns and their tilted time windows. The tilted time windows give a logarithmic overview on the frequency history of each frequent pattern.

3.2 FTP-DS: Temporal Pattern Mining

In [17] a regression based scheme is given for temporal pattern mining from data streams. The authors propose to record and monitor the frequent temporal patterns extracted. The frequent patterns are represented by a regression-based method. The FTP-DS method introduced in [17] processes the transactions time slot by time slot. When a new slot has been reached, FTP-DS scans the data of the new slot with the previous candidates, proposes a set of new candidates and will scan the data in the next slot with those new candidates. This process is repeated while the data stream is active. FTP-DS is designed for mining inter-transaction patterns. The patterns extracted in this framework are itemsets and this work do not address the extraction of sequences as we propose to do. The authors claim that any type of temporal pattern (causality rules, episodes, sequential patterns) can be handled with proper revisions. However, we discuss the limits of mining sequential patterns from data streams in Section 4.1.

3.3 SPEED: Mining Sequential Streaming Data

In [16] the authors propose to extract sequential patterns from a data stream thanks to an original and efficient tree structure. Their problem is the most similar to ours, since their goal is to extract sequences with a specific threshold and to manage the history of frequencies in a tilted time window manner. The proposed tree structure takes into account the inclusion of sequences embedded in each batch in order to optimize their storage. Actually, this tree offers a "region" technique in order to group sub-sequences of a sequence.

© Revue MODULAD, 2007

4 The SMDS Algorithm: Motivation and Principle

Our method relies on a batch environment (widely inspired from [8]) and the prefix tree structure of PSP [11] (for managing frequent sequences). We first study the limitations of a sequential pattern mining algorithm that would be integrated in a data stream context. Then, we propose our framework, based on a sequences alignment principle.

4.1 Sequential Pattern Mining in a Batch Environment

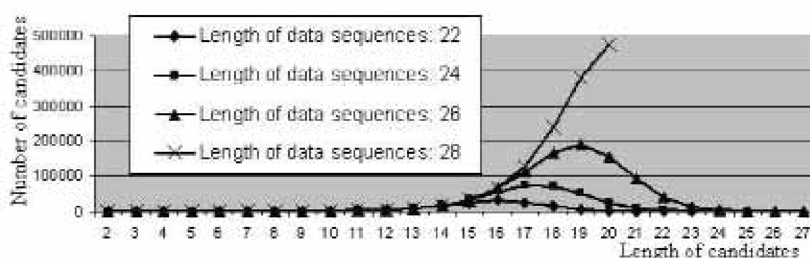


Figure 1: Limits of a batch environment involving PSP

Our method will process the data stream as batches of fixed size. Let $B_1, B_2, ...B_n$ be the batches, where B_n is the most recent batch of transactions. The principle of SMDS will be to extract frequent sequential patterns from each batch b in $[B_1..B_n]$ and to store the frequent approximate sequences in a prefix tree structure (inspired from [11]). Let us consider that the frequent sequences are extracted with a classic exhaustive method (designed for a static transaction database). We argue that such a method will have at least one drawback leading to a blocking operator. Let us consider the example of the PSP [11] algorithm. We have tested this algorithm on databases containing only two sequences (s_1 and s_2). Both sequences are equal and contain itemsets having length one. The first database contains 11 repetitions of the itemsets (1)(2) (*i.e.* $s_1 = <(1)(2)(1)(2)...(1)(2)>$, lentgh$(s_1)=22$ and $s_2 = s_1$). The number of candidates generated at each scan is reported in figure 1. Figure 1 also reports the number of candidates for databases of sequences having length 24, 26 and 28. For the base of sequences having length 28, the memory was exceeded and the process could not succeed. We made the same observation for PrefixSpan[2] [14] where the number of intermediate sequences was similar to that of PSP with the same mere databases. If this phenomenon is not blocking for methods extracting the whole exact result (one can select the appropriate

[2]Downloaded from http://www-sal.cs.uiuc.edu/~hanj/software/prefixspan.htm

© Revue MODULAD, 2007

method depending on the dataset), the integration of such a method in a data stream process is impossible because the worst case may appear in any batch[3].

4.2 Principle

The outline of our method is the following: for each batch of transactions, discovering clusters of users (grouped by behaviour) and then analyzing their navigations by means of a sequences alignment process. This allows us to obtain clusters of behaviours representing the current usage of the Web site. For each cluster having size greater than $minSize$ (specified by the user) we store only the summary of the cluster. This summary is given by the aligned sequence obtained on the sequences of that cluster.

For each batch, our clustering algorithm is initialized with only one cluster, which contains the first navigation (first sequence of the batch). SMDS is then able to process a batch of sequences in only one scan. During this scan, the following operations are performed:

1. For each navigation n in the batch, n is compared to each existing cluster. Let c be the cluster such that its centroid ς_c is the most similar to n, then n is inserted into c. If no such cluster has been found then a new cluster is created and n is inserted in this new cluster. The comparison of n (the navigation sequence) with a cluster c is explained in section 4.4.
2. For each cluster c, SMDS computes the centroid ς_c of c incrementally. This step is detailed in section 4.3. This step is very important, since each sequence s to be inserted in a cluster will be compared to the centroid sequence of each cluster.
3. At the end of the batch, the centroid of each cluster c will stand for the extracted knowledge, since it can be considered as a summary of c.
4. Each centroid is inserted into a prefix tree designed to capture the history of the extracted sequential patterns.

4.3 Centroid of a Cluster

The centroid of a cluster is found thanks to the alignment technique of [10] applied to the cluster. This alignment technique uses the dynamic programming. When the first sequence is inserted in the cluster, the centroid is equal to this unique sequence.
The alignment of sequences leads to a weighted sequence represented as follows: $SA = < I_1 : n_1, I_2 : n_2, ..., I_r : n_r >: m$. In this representation, m stands for the total number of sequences involved in the alignment. I_p $(1 \leq p \leq r)$ is an itemset represented as $(x_{i_1} : m_{i_1}, ...x_{i_t} : m_{i_t})$,where m_i is the number of

[3] In a web usage pattern, for instance, numerous repetitions of requests for pdf or php files are usual

Step 1 :				
S_1:	<(a,c)	(e)	()	(m,n)>
S_2:	<(a,d)	(e)	(h)	(m,n)>
SA_{12}:	(a:2, c:1, d:1):2	(e:2):2	(h:1):1	(m:2, n:2):2
Step 2 :				
SA_{12}:	(a:2, c:1, d:1):2	(e:2):2	(h:1):1	(m:2, n:2):2
S_3:	<(a,b)	(e)	(i,j)	(m)>
SA_{13}:	(a:3, b:1, c:1, d:1):3	(e:3):3	(h:1, i:1, j:1):2	(m:3, n:2):3
Step 3 :				
SA_{13}:	(a:3, b:1, c:1, d:1):3	(e:3):3	(h:1, i:1, j:1):2	(m:3, n:2):3
S_4:	<(b)	(e)	(h,i)	(m)>
SA_{14}:	(a:3, b:2, c:1, d:1):4	(e:4):4	(h:2, i:2, j:1):3	(m:4, n:2):4

Figure 2: Different steps of the alignment method with sequences from example 2

sequences containing the item x_i at the pth position in the aligned sequences. Finally, n_p is the number of occurrences of itemset I_p in the alignment. Example 2 describes the alignment process on 4 sequences. Starting from two sequences, the alignment begins with the insertion of empty items (at the beginning, the end or inside the sequence) until both sequences contain the same number of itemsets.

Example 2 *Let us consider the following sequences:* $S_1 = <(a,c)\ (e)\ (m,n)>$, $S_2 = <(a,d)\ (e)\ (h)\ (m,n)>$, $S_3 = <(a,b)\ (e)\ (i,j)\ (m)>$, $S_4 = <(b)\ (e)\ (h,i)\ (m)>$. *The steps leading to the alignment of these sequences are detailed in Figure 2. At first, an empty itemset is inserted in* S_1. *Then* S_1 *and* S_2 *are aligned in order to provide* SA_{12}. *The alignment process is then applied to* SA_{12} *and* S_3. *The alignment method goes on processing two sequences at each step.*

At the end of the alignment process, the aligned sequence (SA_{14} in figure 2) is a summary of the corresponding cluster. This aligned sequence, which will be the centroid of the cluster from the above example, may not be a true pattern. The approximate sequential pattern can be obtained by specifying k: the number of occurrences of an item in order for it to be displayed. For instance, with the sequence SA_{14} from figure 2 and $k = 2$, the filtered aligned sequence will be: <(a,b)(e)(h,i)(m,n)> (corresponding to items having a number of occurrences greater or equal to k).

In SMDS, the alignment is updated in a incremental way, for each sequence added to the cluster. For that purpose, we maintain a matrix, which contains the number of items for each sequence and a table of distances between each sequence and the other ones. This is illustrated in figure 3. The matrix (left) stores for each sequence the number of occurrences of each item in this sequence.

© Revue MODULAD, 2007

Seq	a	b	c
s_1	2	0	1
s_2	1	0	1
$\vdots$			

Seq	$\sum_{i=1}^{n} similMatrix(s, s_i)$
s_1	16
s_2	14
s_n	13
s_3	11
$\vdots$	
s_{n-1}	1

Figure 3: Distances between sequences

For instance, s_1 is a sequence containing the item a twice. The table of distances stores the sum of similarities ($similMatrix$) between each sequence and the other ones. Let s_{1_i} be the number of occurrences of item i in sequence s_1 and let m be the total number of items. $similMatrix$ is found thanks to the matrix in the following way :
$similMatrix(s_1, s_2) = \sum_{i=1}^{m} min(s_{1_i}, s_{2_i})$.

For instance, with two sequences s_1 and s_2 in the matrix given in figure 3 this sum is: $s_{1_a} + s_{2_b} + s_{2_c} = 1 + 0 + 1 = 2$.
Sometimes, the alignment has to be refreshed and cannot be updated incrementally. Let us consider a sequence s_n. First of all, s_n is inserted in the matrix and its distance to the other sequences is computed ($\sum_{i=1}^{n} similMatrix(s_n, s_i)$). s_n is then inserted in the table of distances, with respect to the decreasing order of distances values. For instance, in figure 3, s_n is inserted after s_2. Let r be the rank where s_n is inserted (in our current example, $r = 2$) in c. 0.5 is a parameter specified by the user. There are two possibilities after having inserted s_n:

1. $r > 0.5 \times |c|$. In this case, the alignment is updated incrementally and $\varsigma_c = alignment(\varsigma_c, s_n)$.
2. $r \leq 0.5 \times |c|$. In this case, the centroid has to be refreshed and the alignment is computed again for all the sequences in this cluster.

4.4 Comparing Sequences and Centroids

Let s be the current sequence and C the set of all clusters. SMDS scans C and for each cluster $c \in C$, performs a comparison between s and ς_c (the centroid of c, which is an aligned sequence). This comparison is based on the longest common sequence (LCS) between s and ς_c. The length of the sequence is also taken into account, since it has to be no more than 120% and no less than 80% of the original sequence (first sequence inserted in c).

Definition 4 Let s_1 and s_2 be two sequential patterns. Let $LCS(s_1, s_2)$ be the length of the longest common subsequences between s_1 and

© Revue MODULAD, 2007