Discovering Frequent Sets from Data Streams with CPU Constraint

Submitted for Blind Review

Abstract

Data streams are usually generated in an online fashion characterized by huge volume, rapid unpredictable rates, and fast changing data characteristics. It has been hence recognized that mining over streaming data requires the problem of limited computational resources to be adequately addressed. Since the arrival rate of data streams can significantly increase and exceed the CPU capacity, the machinery must adapt to this change to guarantee the timeliness of the results. We present an online algorithm to approximate a set of frequent patterns from a sliding window over the underlying data stream – given *apriori* CPU capacity. The algorithm automatically detects overload situations and can adaptively shed unprocessed data to guarantee the timely results. We theoretically prove, using probabilistic and deterministic techniques, that the error on the output results is bounded within a pre-specified threshold. The empirical results on various datasets also confirmed the feasiblity of our proposal.

1 Introduction

Data streams have recently become a novel data type that attracted much attention from the research community, ranging from works in data stream querying (Garofalakis et al. 2002), clustering (Guha et al. 2003), classification (Hulten et al. 2001) to frequent sets mining (Manku et al. 2002). They arise naturally in a number of applications, including financial analysis & stock trading, fraud detection, and IP & sensor network (Babcock et al. 2002, Garofalakis et al. 2002), which exhibit properties of online, huge volume, high arrival rates, and fast changing behaviors. In this novel setting, conventional methods that deal with persistently stored datasets become ineffective in streaming environments. It is therefore imperative to design new techniques that have the ability to compute the answer in an online fashion with only one scan on the streaming data whilst operating under the resource limitations (e.g., CPU cycles (Tatbul et al. 2003), memory space (Jeffrey et al. 2004), and bandwidth communication (Chi et al. 2005)).

For many stream applications such as stock monitors, telecom fraud detection or sensor network surveillance , the data arrival rate is rapid and mostly unpredictable (due to the dynamic changing behavior of data sources) (Das et al. 2003). Meanwhile, analyzing results of these streams usually requires delivery under time constraints to enable optimal decisions. Unfortunately, when the rate of the data stream significantly increases and exceeds the system capacity, overloading situations happen and the system may not be able to deliver the results within the given timeframe. Consequently, the quality of service can be degraded without bound. To cope with such situations, one may add more resources to handle the increased load or distribute the computation to multiple nodes. However, such an approach is expensive and generally infeasible in practice (Tatbul et al. 2003). Therefore, it has been recognized that dropping unprocessed data (thus, only approximate results can be obtained) to reduce workload and maintain the timeliness of output results is generally preferred.

In the literature, the problem of dropping some data in order to cope with high speed data streams is often known as *load shedding*. Currently, this problem has been intensively studied in data stream querying (Babcock et al. 2004, Gedik et al. 2005, Tatbul et al. 2003, 2006, Tu et al. 2006). However, it has not been well-addressed in the context of data stream mining (Chi et al. 2005), especially for the problem of finding frequent sets from transactional data streams. For this fundamental mining task, most algorithms reported so far focused on the issue of managing limited memory space (Lin et al. 2005, Giannella et al. 2003, Chang et al. 2003*a*, Teng et al. 2003, Jeffrey et al. 2004). They ignore the fact that CPU is also a bounded resource. It is important to note that the main difficulty in frequent set mining is the large number of itemsets whose frequencies need to be tracked. Given a transaction of size m, the number of itemsets is exponentially proportional to its size; i.e., for a transaction of size $m = 10$, the number of itemsets may be up to $2^{10} - 1$. This clearly makes the frequent set mining problem susceptible to overloading.

In this paper, we present an approximate algorithm to perform online computation of a set of frequent patterns from a transactional data stream with CPU as a bounded resource. We consider the most general mining model: the *time-based sliding window* model. It is important to note that, although frequent sets are computed in the sliding data window, no revisiting of expiring transactions is needed in our approach. Incoming transaction are processed online and they will be adaptively dropped when overload situations are detected in order to guarantee the timeliness of the mining results. Furthermore, we prove that the error on the mining results is theoretically kept within a preset threshold by exploiting the Chernoff bound property (Chernoff 1952) and the deterministic technique (Manku et al. 2002). To verify the feasibility of our algorithm, experiments on various synthetic datasets have been performed where both the speed and the characteristics of data streams are changed from time to time.

The rest of the paper is organized as follows. In the following section, we formally formulate the prob-

lem of mining frequent sets from a window sliding on data streams given the limitation on CPU capacity. Section 3 describes our approach. The load detection and load shedding technique are addressed first, then the algorithm is represented and followed by the error analysis. Section 4 reports our experimental results. Related work is presented in Section 5. Finally, our conclusions are presented in Section 6.

2 Problem Definition

We consider a high speed data stream arriving as a time ordered series of transactions $\mathcal{DS} = \{t_1, t_2, ..., t_n, ...\}$. Each transaction t_i contains a set of items such that $t_i \subseteq I$, where $I = \{a_1, a_2, ..., a_m\}$ is a set of literals called items (or objects) and t_n is called the current transaction arriving on the stream.

We focus on *time-based* sliding window. In this model, let $TS_0, TS_1, \ldots, TS_{i-k+1}, \ \ldots, TS_i$ denote the time periods elapsed so far in the stream. Each time period (or time slot) contains multiple transactions arriving in that interval and thus, they form a partition of transactions in the stream. Given an integer k, the time-based sliding window is defined as the set of transactions arriving in the last k time periods and denoted by $\mathcal{SW} = \{TS_{i-k+1}, \ldots, TS_{i-1}, TS_i\}$. TS_i is called the latest time slot and TS_{i-k} is called the expiring one in the sliding window. When the time shifts to the new slot TS_i, the effect of all transactions in TS_{i-k} will be eliminated from the mining model.

Let n_w be the number of transactions arriving between TS_{i-k+1} and TS_i, the frequency of an itemset X, denoted by $freq(X)$, is defined as the number of transactions between TS_{i-k+1} and TS_i that contain X. Its support, denoted by $supp(X)$, is defined as the ratio between $freq(X)$ and n_w; i.e., $supp(X) = freq(X) \times n_w^{-1}$. Given $\sigma \in (0, 1]$ as a threshold for minimum support, X is called a frequent itemset if $supp(X) \geqslant \sigma$; it is called a maximal frequent itemset (MFI) if none of its immediate supersets are frequent.

Our problem is defined as follows: we are given a processing capacity (CPU) C of the mining system and a data stream $\mathcal{DS}$ with arbitrarily high arrival rates (the characteristics of $\mathcal{DS}$ also may change with time). Let $Load(\mathcal{DS})$ indicate the workload of the system, then a load shedding is invoked whenever $Load(\mathcal{DS}) > C$. The objective is to find all frequent patterns together with their estimated frequencies in the sliding window while guaranteeing that $Load(\mathcal{DS}) \leqslant C$.

3 Approximate Frequent Sets under bounded CPU

3.1 Workload Estimation

Since the behavior of data streams often changes over time, detecting overload situations is an important step in our algorithm. For the frequent set mining problem, the system workload may not be simply estimated by regularly checking the rate of transactions arriving in one time unit. Rather, it is essentially dependent on the number of itemsets containing in each transaction whose frequencies must be updated. Certainly, an accurate method to evaluate the system workload is to have an exact count of this number in each transaction. Unfortunately, this task is generally impossible since the system may not be able to fully process all incoming transactions under overload situations. Therefore, it is necessary to find a technique that is able to approximately estimate this number meanwhile it is efficient to compute. We propose such an estimate based on a small set of maximal frequent itemsets (MFIs). The key intuition behind using MFIs for this task is that the number of MFIs is exponentially smaller than the number of frequent itemsets, as shown by (Guizhen Yang 2004). Meanwhile, such a compact set completely captures the entire set of frequent itemsets. To give an explicit formula for this estimate, let us denote the number of MFIs in a transaction by m and let X_i be a MFI, $1 \leqslant i \leqslant m$. We assume the following estimated time (or load coefficient) to process one transaction:

$$L = \sum_{i=1}^{m} 2^{|X_i|} - \sum_{i,j=1}^{m} 2^{|X_i \cap X_j|} \qquad (1)$$

In this equation, the first summation estimates the number of frequent itemsets within each MFI. The second one estimates the number of itemsets overlapping between MFIs. Apparently, one may suppose that computing L is expensive and eventually defeats the purpose of quickly detecting CPU overload. Nevertheless, we do not explicitly compute L by finding all MFIs and the overlapping subsets among them. Instead, the set of MFIs is maintained in a prefix tree and the equation is computed by matching transactions to this structure to derive the number of distinct frequent sets. As we shall see from the empirical results in Section 4, this approach yields very good approximation while the computational time is negligible.

Given this computed statistics for each transaction, we measure it for n transactions over one time unit and let r be the current rate of the data stream. The following inequality imposes a constraint on load shedding decisions:

$$P \times r \times \frac{\sum_{i=1}^{n} L_i}{n} \leqslant C \qquad (2)$$

In this inequality, $P \in (0, 1]$ is a parameter adjusted adaptively to make the inequality hold. It also expresses the fraction of transactions that should be discarded. The expression $r \times \frac{\sum_{i=1}^{n} L_i}{n}$ gives the estimated system workload to process transactions arriving in one time unit. L_i is described in Equation 1; C, as formulated above, is the processing capacity of the mining system.

3.2 Probabilistic Technique to Shed Load

As we have analyzed above, when the system is overloaded, an immediate approach is to drop a fraction of the stream to reduce workload. Certainly, when all incoming data is not entirely processed (and dropped transactions are lost forever), one can expect some errors in the results. Our algorithm is designed to approximate this error in a precise manner. In other words, the error is guaranteed within some specific value. To achieve this, we approach the problem by utilizing a technique from probability, the Chernoff bound (Chernoff 1952). Such a theoretically sound tool allows us to obtain a more accurate estimate on the mining error. To apply the Chernoff bound in our frequent set approximation, we clarify the following concepts. Let P be a value smaller than 1, then each coming transaction is chosen with probability P. For a set of N transactions arriving in the stream, n

transactions are chosen randomly. Given an itemset X, we want to approximate how close is its computed frequency in n sampling transactions, as compared to its actual frequency p in N transactions.

Note that event X appearing in a transaction can be seen as a Bernoulli trial and thus, we denote random variable $A_i = 1$ if X appears in the ith transaction and $A_i = 0$ otherwise. Obviously, $\Pr(A_i = 1) = p$ and $\Pr(A_i = 0) = 1 - p$.[1] Hence, n randomly drawn transactions are viewed as n independent Bernoulli trials. Let r be the number of times that $A_i = 1$ occurs in these n transactions; r is called a *binomial* random variable and thus, its expectation is np. Then, the Chernoff bound states that given an error bound ϵ, $0 < \epsilon < 1$:

$$\Pr\{|r - np| \geqslant np\epsilon\} \leqslant 2e^{-np\epsilon^2/2} \tag{3}$$

Let us call $supp_E(X) = r/n$ the estimated support of X computed from n sampling transactions, then this equation gives us the probability that the true support $supp_T(X)$ deviates from its estimated support $supp_E(X)$ by an amount $\pm\epsilon$. If we want this probability to be no more than δ, then the required number of sampling transactions is at least (by setting $\delta = 2e^{-np\epsilon^2/2}$)

$$n_0 = \frac{2p\ln(2/\delta)}{\epsilon^2} \tag{4}$$

For example, consider $p = 0.002$, $\delta = 0.05$ and $\epsilon = 0.001$, then $n_0 \approx 15,000$. This means that for itemset X, if we sample 15,000 transactions from a partition of the stream, its true support $supp_T(X)$ in this partition is beyond the range of $[supp_E(X) - 0.001, supp_E(X) + 0.001]$ with probability 0.05. In other words, $supp_T(X)$ is within $\pm\epsilon$ of $supp_E(X)$ with a high confidence of 95%.

3.3 The Algorithm

In our framework, frequent patterns are discovered from a time-based sliding window where the window of interest is the set of transactions arriving in the last k time slots. We further divide the data stream into *con*ceptual win*dows* (condows) having the same size of Δ transactions and assume that each time slot consists of multiple condows. Accordingly, these condows form the basic units of each time slot (rather than individual transactions) in our model. The value of Δ is chosen to be equal to n_0 as computed in Equation 4.

When the window slides to a new time slot TS_i, the effect of all condows in the expired one, i.e., TS_{i-k} will be eliminated and thus, the current window now consists of condows arriving only in time slots $\mathcal{SW} = \{TS_{i-k+1}, ..., TS_i\}$. It is also important to note that since the rate of the data stream can change with time, the number of condows appearing in each time slot may not be the same.

In essence, not all itemsets appearing in the stream should be recorded and counted due to system resource limitations. By conceptually dividing the stream into condows, we can employ a deterministic threshold to filter itemsets whose frequencies are insignificant. In our algorithm, the threshold $\gamma(< \sigma)$ is used for this task. If a pattern has a support no more than γ, it is unlikely to be frequent in the near future and therefore, should be removed early from the results. Furthermore, our goal is not only to produce an approximate set of frequent patterns but also wants to deliver it within a precise error limit. This guarantee is a challenging task since our algorithm is designed to scan transactions online to find frequent sets; a potential candidate is generated only after all its subsets are found to be significantly frequent. In other words, longer patterns will suffer from a larger margin of error and as a result, it is not able to guarantee the same error for every pattern of different lengths. However, when partitioning the data stream into equal-sized condows, we can approximate the patterns' error based on their lengths and more importantly, it is able to tighten their upper error bound and the condition to generate potential candidates.

In our design, an array storing minimum frequency thresholds is utilized for this purpose. If m is the maximal size of frequent sets that the user wants to explore from the stream, then the only condition this array needs to satisfy is: $a[i] < a[j]$ for $1 \leqslant i < j \leqslant m$ and $a[m] \leqslant \gamma \times \Delta$. Therefore, an itemset of size j will be generated if its immediate subset frequencies in the current condow are above the threshold specified in

[1] We use Pr(.) to denote the probability of a condition being met.

Algorithm 1. Find frequent sets from a time-based sliding window.

Input:
(1) A processing capacity (CPU) C of the mining system;
(2) A window sized of k time slots sliding on data stream $\mathcal{DS}$;
(3) A minimum support threshold $\sigma \in (0, 1]$;
(4) A significant threshold $\gamma \in (0, \sigma)$ and condow size Δ;
Output: At anytime on demand, return all estimated frequent sets computed in the sliding window

1: $TS = 0$; $w_o = 0$; $w_c = 1$; $W[mod(TS, k)] = w_c$;
2: Periodically identify sampling rate P;
3: **for** each $t_n \in \mathcal{DS}$, sampling it with probability of P and if t_n is chosen **do**
4: **for** all $X \subseteq t_n$ and $X \in \mathcal{S}$ **do** $Ccnt(X)++$
5: **for** all $X \subseteq t_n$ and $X \notin S$ **do**
6: **if** X is 1-itemset **then**
7: Insert X to $\mathcal{S}$ with $X.Wid = w_c$; $Ccnt(X) = 1$
8: **else**
9: **if** (for all $Y \subset X$, $\nexists Y \in S$ such that $(Y.Wid < w_c$ and $Ccnt(Y) \leq arr[|X|])$ **or** $(Y.Wid = w_c$ and $Ccnt(Y) \leq arr[|X|] - arr[|Y|]))$ **then**
10: Insert X to $\mathcal{S}$ with $X.Wid = w_c$; $Ccnt(X) = 1$;
11: **end if**
12: **end if**
13: **if** X cannot be inserted into $\mathcal{S}$ **then**
14: $t_n = t_n - \{Z \subseteq t_n \mid Z$ is a superset of $X\}$
15: **end if**
16: **end for**
17: **for** every Δ transactions sampled, scan $\mathcal{S}$ and **do**
18: $X.Freq[w_c] = Ccnt(X) \times P^{-1}$
19: $S = S - \{X \mid X.Wid < w_o$ and $\sum X.Freq[i] \leqslant (w_c - w_o + 1) \times a[|X|]\}$
20: $S = S - \{X \mid X.Wid \geqslant w_o$ and $\sum X.Freq[i] \leqslant (w_c - X.Wid + 1) \times a[|X|]\}$
21: $w_c = \lceil n/\Delta \rceil + 1$
22: **end for**
23: **if** A new time slot arrives and $TS \geqslant k$ **then**
24: $TS{+}{+}$; $w_o = W[mod(TS, k)]$; $W[mod(TS, k)] = w_c$
25: Remove $X.Freq[i]$ such that $i \leqslant w_o$
26: **end if**
27: **end for**
28: **if** mining results are requested **then**
29: **return** $\{X \mid X.Wid < w_o$ and $\sum X.Freq[i] \geqslant (w_c - w_o + 1) \times \Delta \times \sigma\}$
30: **return** $\{X \mid X.Wid \geqslant w_o$ and $\sum X.Freq[i] + (X.Wid - w_o + 1) \times a[|X|] \geqslant (w_c - w_o + 1) \times \Delta \times \sigma\}$
31: **end if**

$a[j]$; i.e., its subsets are significant in the condow. On the other hand, by choosing the size of each condow to be equal to Δ transactions, we can approximate (in terms of probability) the true support of patterns within each time slot to be no more than ϵ when load shedding is involved. As will be shown in Section 3.4, the error on frequent sets is guaranteed to be within the preset error thresholds γ and ϵ.

Given the above analysis, we describe our algorithm as follows. We name the algorithm *L*oad *S*hedding for mining frequent sets from *S*liding *W*indows (LSSW) and its pseudo code is outlined in Algorithm 1. LSSW uses a prefix tree $\mathcal{S}$ to maintain frequent sets discovered from the sliding window. Each node in $\mathcal{S}$ corresponds to an itemset X having the following fields:

- $Item$: The last item of itemset X. Thus, X is represented by the set of items on the path from the root to the node.
- Wid: The index of the condow at which X is inserted into $\mathcal{S}$.
- $Freq[1..max]$: A circular queue storing frequencies counted at each condow in the sliding window.

We denote the latest (or current) condow in the sliding window by w_c, and the oldest with w_o. To efficiently eliminate expired time periods, the indexes of the first condows in each time slot are tracked globally and stored in a circular array W of k elements corresponding to the last k time slots. Accordingly, the value of w_o is always cleared each time the window slides. The time slot index TS is initialized to 0 when the algorithm starts and is incremented at each new time period. Let t_n be the new arriving transaction. LSSW consists of the following steps:

1. The system workload is periodically estimated to identify overloading. If such a situation occurs, the maximal value of P is identified via Equation 2. Otherwise, P is set to 1.
2. For each incoming transaction t_n, it is sampled with probability P. If t_n is chosen, the following steps are performed:
   - Increment: If an itemset X appearing in t_n is also currently maintained in $\mathcal{S}$, then its frequency in the current condow, denoted by $Ccnt(X)$, is increased by 1.
   - Insert: For each $X \subseteq t_n$ and $X \notin \mathcal{S}$, insert X into $\mathcal{S}$ with $X.Wid = w_c$ and $Ccnt(X) = 1$ if X is a singleton [2]; Otherwise, let Y be any immediate subset of X, then X is inserted into $\mathcal{S}$ if the following three conditions hold:
     - All immediate subsets of X are in $\mathcal{S}$;
     - $\nexists Y$ such that $Y.Wid < w_c$ and $Ccnt(Y) \leqslant arr[|X|]$; i.e., there is no Y being inserted into $\mathcal{S}$ from previous condows whose frequency in the current condow is insufficiently significant;
     - $\nexists Y$ such that $Y.Wid = w_c$ and $Ccnt(Y) \leqslant (arr[|X|] - arr[|Y|])$; i.e., there is no Y that has just been inserted into $\mathcal{S}$ in the current condow after which its frequency is no more than $(arr[|X|] - arr[|Y|])$.

     In cases where X is not inserted into $\mathcal{S}$, all its supersets in t_n need not be further checked.
   - Update frequency: After each Δ transactions have been sampled, the algorithm updates $Freq[w_c] = Ccnt(X) \times P^{-1}$. Note that to compensate for dropping transactions using sampling rate P, X's frequency is scaled up appropriately by P^{-1} to approximate its true frequency in the sampling part of the stream. At this step, LSSW also scans $\mathcal{S}$ to remove all but 1-itemsets that satisfy either of the two conditions:
     - If $X.Wid < w_o$ and $\sum X.Freq[i] \leqslant (w_c - w_o + 1) \times a[|X|]$
     - If $X.Wid \geqslant w_o$ and $\sum X.Freq[i] \leqslant (w_c - X.Wid + 1) \times a[|X|]$

     Certainly if an itemset is removed, all its supersets are also removed. Those itemsets recently inserted into $\mathcal{S}$ in the current condow will not be removed since they are generated after their immediate subsets became sufficiently frequent. After that, LSSW updates the index for the next condow by $w_c = \lceil n/\Delta \rceil + 1$.
   - Remove expiring time slot: In case the window slides to a new time slot, TS will be incremented by 1 and if $TS \geqslant k$, the algorithm further removes frequency counts in queue $X.Freq$ where indexes are smaller than value w_o stored in $W[mod(TS, k)]$. Then LSSW updates $W[mod(TS, k)] = w_c$ to register the first condow index of the new time slot.
3. At any instant upon the user's request, the algorithm scans $\mathcal{S}$ to output all itemsets satisfying either of the conditions below:
   - If $X.Wid < w_o$ and $\sum X.Freq[i] \geqslant (w_c - w_o + 1) \times \Delta \times \sigma$
   - If $X.Wid \geqslant w_o$ and $\sum X.Freq[i] + (X.Wid - w_o + 1) \times a[|X|] \geqslant (w_c - w_o + 1) \times \Delta \times \sigma$

   For the second condition, it is noted that $(X.Wid - w_o + 1) \times a[|X|]$ is X's maximal frequency error from w_o to $X.Wid$ (as will be proven in the following section).

3.4 Error Analysis

We prove the correctness of our algorithm in this section. For simplicity, we shall omit the notation X and re-denote its *true* frequency between two condows w_α and w_β (where $\alpha \leqslant \beta$) by $f_T(w_\alpha, w_\beta)$ and its *estimated* one by $f_E(w_\alpha, w_\beta) = \sum_{i=\alpha}^{\beta} X.Freq[i]$. Respectively, $s_T(w_\alpha, w_\beta)$ and $s_E(w_\alpha, w_\beta)$ denote its *true* and *estimated* supports in this period. In the proof of Lemmas 1 and 2, we ignore 1-itemsets and itemsets those have $X.Wid < w_o$ since their frequencies have been precisely counted in the sliding window.

[2] Note that the immediate subsets of a 1-itemset is $\{\varnothing\}$ which appears in every transaction, all 1-itemsets are therefore inserted into $\mathcal{S}$ without conditions. For the same reason, they are also not pruned from $\mathcal{S}$.

Lemma 1 *Under no load shedding, if an itemset X is deleted at condow w_d, then*

1. *$f_T(w_s, w_d) \leqslant (w_d - w_s + 1) \times a[|X|]$; i.e., the true frequency of X between w_d and w_s – any condow at which X was previously inserted – is no more than $(w_d - w_s + 1) \times a[|X|]$; and*

2. *$f_T(w_d^p + 1, w_d) \leqslant (w_d - w_d^p) \times a[|X|]$, where w_d^p is any condow (previous w_d) at which X was also deleted.*

Proof. Let us denote condow indexes of the two latest periods at which X is inserted and deleted by $w_s^{\ell-1}$, $w_d^{\ell-1}$ and w_s^ℓ, w_d^ℓ respectively.

When X is inserted at w_s^ℓ, its maximum error at this condow is $a[|X|]$, and $f_E(w_s^\ell, w_d^\ell)$ is its frequency count since then to w_d^ℓ. Therefore $f_T(w_s^\ell, w_d^\ell) \leqslant f_E(w_s^\ell, w_d^\ell) + a[|X|]$. Together with the deletion rule at w_d^ℓ, we have $f_T(w_s^\ell, w_d^\ell) \leqslant (w_d^\ell - w_s^\ell + 1) \times a[|X|]$. On the other hand, X is not inserted anytime between $w_d^{\ell-1} + 1$ and $w_s^\ell - 1$, its true frequency in each of these condows is no more than $a[|X|]$. This means that $f_T(w_d^{\ell-1} + 1, w_s^\ell - 1) \leqslant (w_s^\ell - 1 - w_d^{\ell-1}) \times a[|X|]$. Summing up these 2 periods, $f_T(w_d^{\ell-1}, w_d^\ell) \leqslant (w_d^\ell - w_d^{\ell-1}) \times a[|X|]$ is derived.

Similar to above when X was inserted at $w_s^{\ell-1}$ and deleted at $w_d^{\ell-1}$, we have $f_T(w_s^{\ell-1}, w_d^{\ell-1}) \leqslant (w_d^{\ell-1} - w_s^{\ell-1} + 1) \times a[|X|]$. Together with the above result, $f_T(w_s^{\ell-1}, w_d^\ell) \leqslant (w_d^\ell - w_s^{\ell-1} + 1) \times a[|X|]$ is satisfied. By repeating the same calculation for every previous insertion and deletion periods, the lemma follows. □

Lemma 2 *Under no load shedding, if an itemset X is deleted at condow w_d, then its true frequency between w_d and any previous window w_i is no more than $(w_d - w_i + 1) \times a[|X|]$; i.e., $f_T(w_i, w_d) \leqslant (w_d - w_i + 1) \times a[|X|]$.*

Proof. We distinguish two cases. For the first case, where X's frequency is not counted at w_i. Then, from w_i to its next insertion at w_s^i (but not include w_s^i itself), its true frequency at each condow is no more than $a[|X|]$. Thus, $f_T(w_i, w_s^i - 1) \leqslant (w_s^i - w_i) \times a[|X|]$. On the other hand, by Lemma 1, its true frequency since w_s^i to w_d is bounded by $f_T(w_s^i, w_d) \leqslant (w_d - w_s^i + 1) \times a[|X|]$. Therefore, $f_T(w_i, w_d) \leqslant (w_d - w_i + 1) \times a[|X|]$.

For the second case where X's frequency is counted at w_i, we prove by contradiction. Assume that $f_T(w_i, w_d) > (w_d - w_i + 1) \times a[|X|]$. Let us denote the closest condows of w_i, at which X is inserted and deleted by w_s^i and w_d^i respectively (i.e., $w_s^i \leqslant w_i \leqslant w_d^i$). By Lemma 1, $f_T(w_d^i + 1, w_d) \leqslant (w_d - w_d^i) \times a[|X|]$. Meanwhile, we have assumed that $f_T(w_i, w_d) > (w_d - w_i + 1) \times a[|X|]$. Accordingly, its true frequency between w_i and w_d^i must be greater than $(w_d^i - w_i + 1) \times a[|X|]$. Together with the fact that X has been counting since w_s^i, we have $f_E(w_s^i, w_i - 1) > (w_i - 1 - w_s^i) \times a[|X|]$. Consequently $f_E(w_s^i, w_d^i) > (w_d^i - w_s^i) \times a[|X|]$, making X not be deleted at w_d^i. This contradicts the fact that X was deleted at w_d^i. The lemma is proved. □

Lemma 2 is important since it allows approximating the maximal error of every itemset discovering within the sliding window regardless of the new index value of w_o. Note that the value of w_o is arbitrary since the number of condows within each time period is variable.

Theorem 1 *Let w_o and w_c be the oldest and current condows of the sliding window respectively. Under no load shedding, $s_T(w_o, w_c) \leqslant s_E(w_o, w_c) + \gamma$.*

Proof. If X is inserted at w_o, the first condow of the current sliding window, its maximum error at this condow is at most $a[|X|]$. Therefore, $f_T(w_o, w_c) \leqslant f_E(w_o, w_c) + a[|X|]$. Otherwise, X is possibly deleted some time earlier, as late as, at condow $w_i - 1$, and then inserted into $\mathcal{S}$ at condow w_i. By Lemma 2, $f_T(w_o, w_i - 1)$ is at most $(w_i - w_o) \times a[|X|]$ when such a deletion took place. Since $a[|X|]$ is the maximal error when X is inserted at w_i and $f_E(w_i, w_c)$ is its frequency count since then, it follows that $f_T(w_o, w_c) \leqslant f_E(w_i, w_c) + (w_i - w_o + 1) \times a[|X|]$.

On the other hand, let n_w be the number of transactions in the sliding window. Certainly, $\Delta \times (w_i - w_o + 1) \leqslant n_w$ (since $w_i \leqslant w_c$), we have $(w_i - w_o + 1) \times a[|X|] \leqslant \gamma \times n_w$. Therefore, dividing the inequality above for n_w, we derive $s_T(w_o, w_c) \leqslant s_E(w_o, w_c) + \gamma$. □

Theorem 2 *Under load shedding, $s_T(w_o, w_c) \leqslant s_E(w_o, w_c) + \gamma + \epsilon$ with a probability of at least $1 - \delta$.*

Proof. This theorem can be directly derived from the Chernoff bound and Theorem 1 above. At time slots where the load shedding happens, the true support of X is guaranteed within $\pm\epsilon$ of the counting support with probability $1 - \delta$ when the Chernoff bound is applied for sampling. Meanwhile, by Theorem 1, this counting support is limited by $s_T(w_o, w_c) \leqslant s_E(w_o, w_c) + \gamma$. Therefore, getting the upper bound (i.e., ϵ) in the Chernoff bound, we derive the true support of X to be no more then its estimated support by $\gamma + \epsilon$ with a confidence of $1 - \delta$. □

4 Experimental Results

We implemented our algorithm in C++ and performed experiments on a 1.9GHz Pentium machine with 1GB of memory running Windows XP. Three synthetic data streams are generated with a size of 3 million transactions each by using the IBM data generator (Agrawal et al. 1994) (we are not able to obtain any real data set that is large enough to simulate data streams). We obtained dataset T10.I6.D3000K by setting average transaction size T=10, average maximal frequent set size I=6. With similar setting, we obtained T8.I4.D3000K and T5.I3.D3000K. For all three datasets, the number of maximal potentially frequent itemsets L=2000 and the number of unique items N=10,000 (for more detail on dataset generation, we refer the reader to (Agrawal et al. 1994)). Since our approach is probabilistic, the precision and recall measures will be used to evaluate the effectiveness of our algorithm. For the same reason, each experiment is repeated 10 times for each parameter combination and the average results are reported.

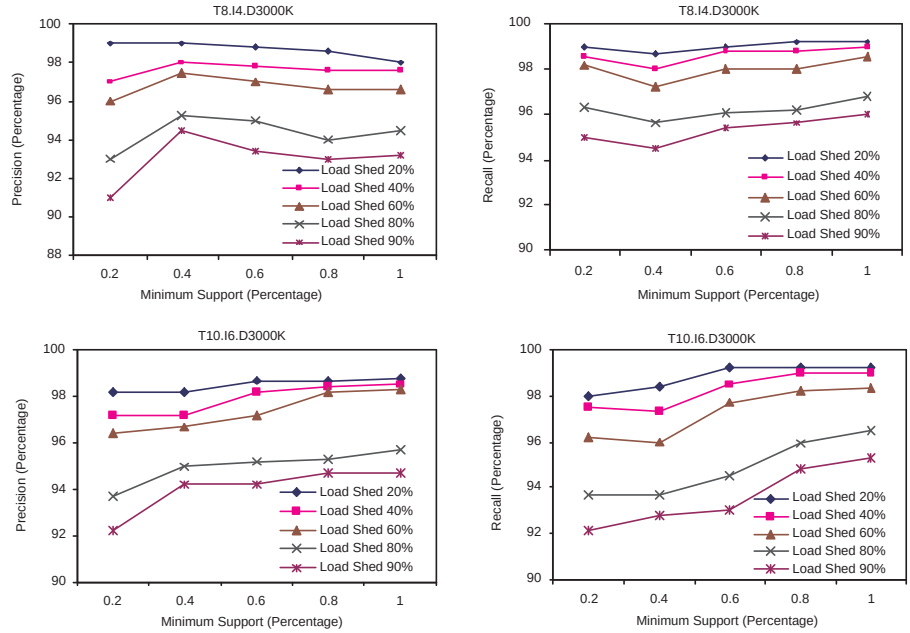

Figure 1: Accuracy on T8.I4.D3000K and T10.I6.D3000K

4.1 Accuracy measurements

In our experiments, we fix $\epsilon = 0.001, \delta = 0.01$ and $p = 0.002$. Accordingly, the number of transactions for each condow as determined by the Chernoff bound is $\Delta \approx 20K$. Since $p = 0.002$, the minimum support threshold σ will be chosen to be no less than this value. Specifically, σ will be varied between 0.002 and 0.01. We also fix $\gamma = 0.1\sigma$ (except $\gamma = 0.25\sigma$ for $\sigma = 0.002$) and the maximal length of frequent patterns $m = 10$ (interval between $a[i]$ is $\gamma\Delta/m$). The sliding window consists of 10 time slots and each time slot receives $200K$ transactions (i.e., 10 condows). For simplicity, instead of varying loads, we fix the load and change the number of condows that the system can handle at each time period. For example, if only 2 of 10 condows can be processed in each time slot, this translates to an input stream rate that is 5 times higher than the CPU capacity and thus, the percentage of load shedding is 80%.

Figure 1 shows our experiment results on two datasets T8.I4.D3000K and T10.I6.D3000K where the minimum support threshold is varied from 0.2% to 1% and the load shed is increased from 20% to 90%. We observed that at lower levels of system workload, the algorithm returns a higher number of true frequent itemsets as indicated by the higher value of recall, and a smaller number of false frequent itemsets from the precision measure. A larger gap in accuracy occurs when the load shed increases from 60% to 80% and 90%. At these levels, only 2 and 1 condows are processed (respectively) at each time slot. Nevertheless, the algorithm continues to find more than 92% of all true frequent itemsets (while keeping the false frequent itemsets at below 10%) even though the system workload has increased 10 times that of the CPU capacity, i.e., a load shed of 90%. And this is maintained across all levels of support thresholds.

More details are shown in Figure 2 when we plot the precision and recall for each itemset length. We report the results on T10.I6.D3000K with $\sigma = 0.2\%$ since at this threshold, it is possible to find frequent sets that has a length of up to 10. It can be observed that the precision decreases as the length of a frequent set increases. This happens because the longer patterns are generated only after *all* its subsets are found to be significantly frequent. Hence, the precision on longer itemsets is expected to be lower as a consequence of the larger margin error. Fortunately, distinguishing the error by its pattern length allows the algorithm to closely approximate the maximum lost for each pattern. As seen in Figure 2, the precision on 10-itemsets is maintained above 80%. Note that the recall is not affected by this approximation as the true frequency of itemsets is guaranteed by the Chernoff bound, which is generally dependent on the number of sampling transactions. We observed that other than very large load shedding (more than 80%), the recall is always found to be higher than 92% for every itemset length.

4.2 Adaptability

To test the adaptability of our algorithm, we use T5-8-10.D3000K where its first, second and third parts are made up of each 1000K transactions from T5.I3.D3000K, T8.I4.D3000K, and T10.I6.D3000K. To evaluate the workload using Equation 2, we first set the CPU capacity to be infinite and let the algorithm process the entire dataset. Figure 3(a) shows the relationship between the processing time and the statistics measured within each condow, where the range of support threshold varies between 0.2% and 0.6%. In the figure, σT and σT(w/o MFI) respectively indicates the time to process a condow when we enable and disable the function detecting workload using MFI tree. σS refers to the statistics computed in Equation 2 for each condow. As seen from the figure, σT and σT(w/o MFI) are very close to each other, indicating that the time to compute the statistics is very small compared to the time to find all frequent sets from the stream. When using MFI tree

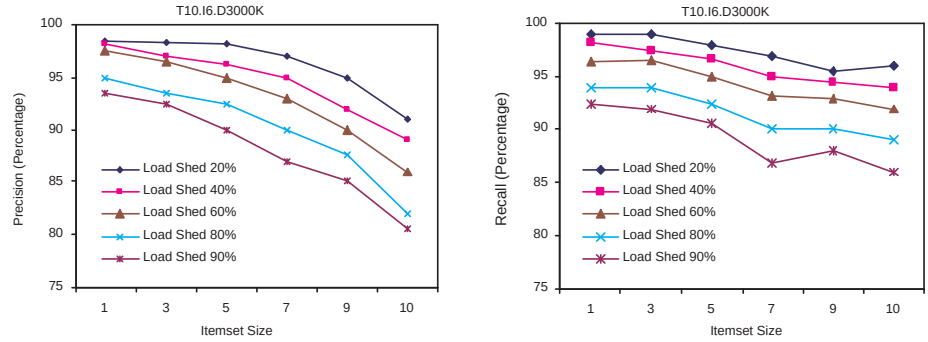

Figure 2: Accuracy vs. Itemset Length for T10.I6.D3000K

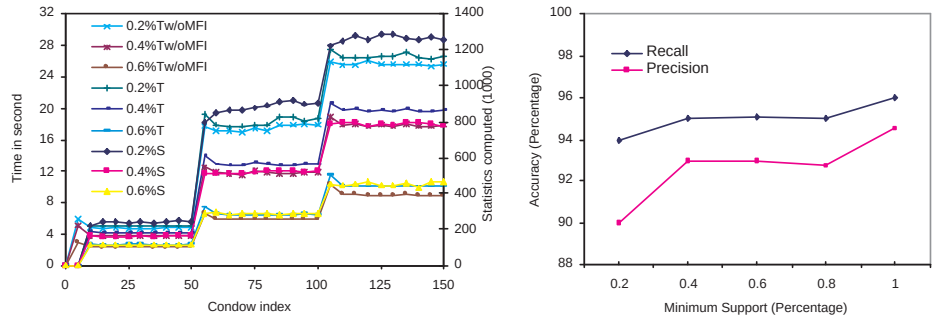

Figure 3: Adaptability on T5-8-10.D3000K

to evaluate workload, it only incurs more CPU usage when the algorithm updates this structure. However, this operation is only performed when a significant change in workload is detected in the stream or it is periodically performed for a set of condows. In the experiments presented, the MFI tree is updated at every time slot (or every 10 condows).

Also from Figure 3(a) we observe that the statistics computed is almost linearly proportional to the processing time across the support thresholds experimented. For instance, at $\sigma = 0.4\%$, the average statistics computed for one condow for transactions sized T5 is 160.7K, T8 is 514.3K and T10 is 792.5K; and the average time to process them are respectively 4.1s, 13.0s and 19.8s. What these statistics mean is that if we limit the CPU capacity and the data stream is initially sent to the system at a rate just below its CPU capacity, then the algorithm will have to shed load by sampling at a rate of 1/3 ($\approx \frac{160.7}{514.3} \approx \frac{4.1}{13}$) and 1/5 ($\approx \frac{160.7}{792.5} \approx \frac{4.1}{19.8}$) in the second and third parts of the stream. In other words, it is effective to use the computed statistics to identify the appropriate amount of data for shedding.

In Figure 3(b), we report the accuracy of our algorithm for this experiment. As we can observe, the recall at all minimum support thresholds is still very high ($\geqslant 94\%$) and is likely the same. Nevertheless, we find that the precision is slightly lower than that in the pure dataset T10.I6.D3000K. This happens since we note that the last sliding window also includes 1000K transactions sized T8 and when the algorithm processes T10.I6.D1000K, more frequent itemsets are found in this part of the stream. According to our approximation (when estimating $(X.Wid - w_o) \times a[|X|]$ to be X's maximal frequency lost), a small fraction of those itemsets discovered in T10.I6.D1000K have over-estimated frequencies. This means that they were locally frequent in the third part of the stream, but not in the second part. However, it is interesting to note that at all support thresholds examined, the percentage of false frequent itemsets is consistently within 10%.

5 Related Work

The issue of dropping a fraction of data sets when the system is overloaded has been intensively studied in real-time databases (Wolfgang & Alexander 1994). Recently, this problem has been extended to data stream querying (Tatbul et al. 2003, Babcock et al. 2004, Tu et al. 2006, Tatbul et al. 2006). In this context, the problem of load shedding is defined as the process of finding an optimal plan for inserting dropping operations in a query network. The Aurora system is one of the first stream querying projects addressing this issue. This work utilizes various Quality of Service (QoS) graphs to represent different important levels of querying objects. Accordingly, when the system is overloaded, arriving transactions will be shed progressively starting from those containing less important objects. This project has recently developed to Borealis, a distributed stream processing system, where the load shedding problem is extended for sliding window model (Tatbul et al. 2006). The STREAM project (Babcock et al. 2004) is another work in this field. In STREAM, a load shedding scheme based on sampling is proposed for sliding window aggregate queries. This approach modifies the query network by inserting load shedder operators together with their sampling rates in such a way that

the total sampling rate eliminates sufficient amount of dropping data. This work is similar to ours in that the random sampling is used as a means of load shedding. However, it addresses the problem of querying objects with optimized sampling rates in a query network while our work addresses the issue in the context of frequent set mining. In data stream mining, Loadstar (Chi et al. 2005) is recently reported as the first work focused on the load shedding problem for classification. In this work, various Quality of Decision (QoD) metrics is introduced to measure uncertainty levels in classification when exact values of the data are not available because of load shedding. A Markov model is utilized to predict the distribution of feature values and then classification decisions are made based on predicted values and QoD metrics.

On the other hand, recent work on mining frequent itemsets over data streams can be classified into three models. The first model is the *landmark* model where frequent sets are discovered between a particular point of time (called landmark) and the current time. Lossy Counting (Manku et al. 2002) and FDPM (Jeffrey et al. 2004) are typical algorithms in this group. The second model is the *time-fading* model where transactions are weighted on the time they arrive. This model gives more attention (fine granularity) to the recently arrived data and relaxes for the earlier ones (coarse granularity). Works that focus on this model include the estDec (Chang et al. 2003*a*) and FP-Streaming (Giannella et al. 2003) algorithms. The third model is the *sliding-window* model. Compared to the two previous ones, this model further considers the elimination of transactions. Frequent itemsets are found within a fixed portion of the stream which is pre-specified by a period of time or a number of transactions. FTP-DS (Teng et al. 2003) (Frequent Temporal Patterns of Data Streams), estWin (Chang et al. 2003*b*), and Time-sensitive sliding window (Lin et al. 2005) are algorithms focusing on this model. A core idea underlying all these works is that they focus on designing methods that can efficiently summarize within limits of main memory. Our work (classified to the sliding window model) further addresses the load shedding problem when CPU capacity is not sufficient to manage all streaming transactions. More importantly, the mining results are approximated and guaranteed within a precise error threshold.

6 Conclusions

In this paper, we visit the issue of applications with data streams that had a variable arrival rate and data characteristics that could inadvertently push the workload above the system's capacity. To avoid such situations, it is important to design algorithms that can adapt to the underlying changes in the data characteristics to guarantee the timely delivery of results. Finding frequent patterns is an exemplar of a difficult problem in such a situation, and our paper presents the solution to this problem. An algorithm to approximate the set of frequent patterns from the data stream while taking the limitations of CPU capacity is proposed. Not only does our proposed algorithm performed well in overloaded situations, the results that it delivers are well guaranteed to be within the error bounds specified across patterns of different lengths. In addition to the theoretical proofs, our empirical evidences provided the verification that our proposal is effective in dealing with such data streams.

References

Agrawal, R. & Srikant, R. (1994), Fast Algorithms for Mining Association Rules in Large Databases, *in* 'VLDB International Conference on Very Large Data Bases', pp. 489–499.

Babcock, B., Datar, M. & Motwani, R. (2004), Load Shedding for Aggregation Queries over Data Streams, *in* 'VLDB International Conference on Data Engineering', pp. 350–361.

Babcock, B., Babu, S., Datar, M., Motwani, R. & Widom, J. (2002), Models and Issues in Data Stream Systems, *in* 'ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', pp. 1–16.

Lin, C., Chiu, D., Wu, Y. & Chen, A. (2005), Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window, *in* 'SIAM International Conference on Data Mining', pp. 68–79.

Das, A., Gehrke, J. & Riedewald, M. (2003), Approximate join processing over data streams, *in* 'ACM SIGMOD International Conference on Management of Data', pp. 40–51.

Garofalakis, M., Gehrke, J. & Rastogi, R. (2002), Querying and mining data streams: you only get one look a tutorial, *in* 'ACM SIGMOD International Conference on Management of Data'.

Gedik, B., Wu, K.L., Yu, P.S. & Liu, L. (2005), Adaptive load shedding for windowed stream joins, *in* 'ACM CIKM International Conference on Information and Knowledge Management', pp. 171–178.

Giannella, C., Han, J., Pei, J., Yan, X. & Yu, P.S. (2003), *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*, Next Generation Data Mining AAAI/MIT.

Guha, S., Meyerson, A., Mishra, N., Motwani, R. & O'Callaghan, L. (2003), Clustering Data Streams: Theory and Practice, *in* 'IEEE Transactions on Knowledge and Data Engineering', pp. 515–528.

Chernoff, H. (1952), A measure of asymptotic efficiency for thests of a hypothesis based on the sum of observations, *in* 'Annals of Mathematical Statistics', pp. 493–509.

Hulten, G., Spencer, L. & Domingos, P. (2001), Mining Time-Changing Data Streams, *in* 'ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 97–106.

Wolfgang, H. & Alexander, S. (1994), *Real time computing*, Springer-Verlag.

Chang, J.H. & Lee, W.S. (2003*a*), Finding recent frequent itemsets adaptively over online data streams, *in* 'ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 487–492.

Chang, J.H. & Lee, W.S. (2003*b*), EstWin: Adaptively monitoring recent change of frequent itemsets over online data streams, *in* 'ACM CIKM International Conference on Information and Knowledge Management', pp. 536–539.

Manku, G.S. & Motwani, R. (2002), Approximate Frequency Counts over Data Streams, *in* 'VLDB International Conference on Very Large Data Bases', pp. 346-357.

Tatbul, N. & Zdonik, S.B. (2006), Window-Aware Load Shedding for Aggregation Queries over Data Streams, *in* 'VLDB International Conference on Very Large Data Bases', pp. 799-810.

Tatbul, N., Çetintemel, U., Zdonik, S.B., Cherniack, M. & Stonebraker, M. (2003), Load Shedding in a Data Stream Manager, *in* 'VLDB International Conference on Very Large Data Bases', pp. 309-320.

Teng, W.G., Chen, M.S., & Yu, P.S. (2003), A Regression-Based Temporal Pattern Mining Scheme for Data Streams, *in* 'VLDB International Conference on Very Large Data Bases', pp. 93-104.

Tu, Y.C., Liu, S., Prabhakar, S. & Yao, B. (2006), Load Shedding in Stream Databases: A Control-Based Approach, *in* 'VLDB International Conference on Very Large Data Bases', pp. 787-798.

Guizhen Yang (2004), The complexity of mining maximal frequent itemsets and maximal frequent patterns, *in* 'ACM SIGKDD International Conference on Knowledge Discovery and Data Mining', pp. 344-353.

Chi, Y., Yu, P.S., Wang, H. & Muntz, R.R. (2005), Loadstar: A Load Shedding Scheme for Classifying Data Streams, *in* 'SIAM International Conference on Data Mining', pp. 346–357.

Yu, J.X., Chong, Z., Lu, H. & Zhou, A. (2004), False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams, *in* 'VLDB International Conference on Very Large Data Bases', pp. 204-215.