# Adaptive Burst Detection in a Stream Engine*

Daniel Klan, Marcel Karnstedt, Christian Pölitz, Kai-Uwe Sattler
Department of Computer Science & Automation
Ilmenau University of Technology, Germany
{*first.last*}@tu-ilmenau.de

Conny Franke
Databases and Information Systems Group
University of California, Davis, USA
franke@cs.ucdavis.edu

## ABSTRACT

Detecting bursts in data streams is an important and challenging task. Due to the complexity of this task, usually burst detection cannot be formulated using standard query operators. Therefore, we show how to integrate burst detection for stationary as well as non-stationary data into query formulation and processing, from the language level to the operator level. Afterwards, we present fundamentals of threshold-based burst detection. We focus on the applicability of time series forecasting techniques in order to dynamically identify suitable thresholds for stream data containing arbitrary trends and periods. The proposed approach is evaluated with respect to quality and performance on synthetic and real-world sensor data using a full-fledged DSMS.

## 1. INTRODUCTION

Data Stream Management Systems (DSMS) are built for processing and reacting to continuous input data, which also involves certain data mining tasks. One important task is burst detection. A burst is defined as an abnormal aggregate in the stream. Usually, bursts are determined in the context of one or more time windows moving over the data stream.

Burst detection is used in a lot of application scenarios, like monitoring facility sensor data, stock trading systems, astrophysics, and traffic control systems. In all cases we have to handle continuously streaming data, which requires incremental one-pass algorithms. Additionally, all the mentioned scenarios claim for detecting bursts over a variety of dynamically chosen window sizes, rather than over predefined ones. This is called *elastic burst detection* [9].

To illustrate the problem of non-stationary stream data, imagine heat sensors, located in a single living room. A burst is detected by summing up the single temperatures from the time window of interest – any resulting sum that is above a given threshold is regarded as a burst and should result in according reactions. But, we will observe periods, because temperatures are changing with day and night hours. Second, especially on hot summer days, the room will continuously heat up, particularly in the midday hours. These facts should be involved in triggering actuations like lowering heating systems or blinds, but this should not result in an alarm just because the system thinks it detected a burst. Thus, the threshold for bursts must be adapted with occurring trends and periods. As we will show, one approach to handle this is to apply techniques from time series forecasting.

A main feature of our approach is that both, burst detection and time series forecasting, are implemented in two absolutely independent operators. Like this, they can be arbitrarily combined with other operators in a multi-functional DSMS.

*This work was in part supported by the National Science Foundation under Award No. ATM-0619139 and by the BMBF under grant 03WKBD2B.


*SAC'09* March 8-12, 2009, Honolulu, Hawaii, U.S.A.


## 2. RELATED WORK

Zhu et al. [14] characterize bursts as "abnormal aggregates in data streams". [6, 11] focus on modeling bursty behavior in data streams, but not on an efficient algorithm for detecting them. The authors of [10] mine for bursts in web search query logs. Shasha et al. [9] propose a technique for elastic burst detection in time series, where bursts can be identified over a variety of dynamically chosen window sizes. In [13], Zhang and Shasha suggest the utilization of *shifted aggregation trees* for elastic burst detection in acceptable time and space. This approach performs very well and its implementation is easy and light-weight. The authors propose the data structure, how to build and optimize it, as well as how to mine it for bursts. The approach proposed in [13] is only applicable to stationary data. This is because the threshold for identifying bursts is only adapted very slowly with evolving input data. This does not perform satisfyingly in the presence of trends or periods, because in this case the amount of false positives and wrong negatives is usually too high. The main idea of our work is therefore to combine their approach with time series forecasting, which allows to adapt the threshold to the evolving stream characteristics. To the best of our knowledge this is novel for burst detection, although forecasting methods have been used for solving similar problems. For instance, [12] utilizes auto regression models to detect outliers and change points in non-stationary data.

A closely related problem is the detection of outliers and anomalies [8, 5]. In [8], Shahabi et al. introduce a wavelet-based approach to detect outliers in time series over different window sizes. Keogh et al. [5] introduce a technique to detect anomalous patterns in time series. Yamanishi et al. [12] utilize auto regression models to detect outliers from non-stationary time series.

Different approaches have been proposed for forecasting, analyzing trends and periods [3, 7, 4]. We currently use exponential smoothing following the Holt and Holt-Winters methodologies [2]. Using this, the next points of the input stream can be predicted satisfyingly accurate. Unfortunately, forecasting methods tend to become more inaccurate as more points are predicted in the future. Because of this, we recompute the forecast periodically, resulting

in only short-term predictions. Although exponential smoothing is suitable in our scenario, we investigate other methods as well.

# 3. STREAM OPERATORS FOR BURST DETECTION

Burst detection is an important component of data stream processing. Apart from monitoring applications where the identification of bursts is the primary task, burst detection is also a useful task for data cleaning. Thus, the goal of our work is a tight integration of burst detection operators into a stream processing engine. In the following, we describe this approach as part of our DSMS *StreamDB*.

## 3.1 CQL Extensions

Implementing burst detection as a stream operator means to allow arbitrary data streams as input (which could be the result of preceding processing steps like filtering or join) and producing an output stream (consisting of the burst tuples) that can be further processed by other operators. Our StreamDB engine is a complete DSMS supporting CQL as the query language. As an extension to the "standard" CQL as described e.g. in [1], StreamDB provides so-called synopsis operators which can be defined on streams similar to sliding window operators:

```
select * from stream [ synopsis(params) ]
```

In this way, burst detection is implemented as a synopsis operator, too, as shown in the following example:

```
select value, timestamp
from s1 [ burst-detection(wlen => 100,
          threshold =>'Holt Winters') ]
```

Here, burst detection is applied to the stream `s1`, which could be either a physical stream or even a view defined on other streams. The parameters in this example are the length of the sliding window `wlen` used for the aggregation pyramid (see Sect. 3.2) and the method used for determining the threshold for bursts (see Sec. 3.3). The output of the synopsis is a stream that consists of all burst tuples – in this case the pairs of value and timestamp.

In the internal physical query execution plans synopses are implemented as dedicated plan operators. In the case of burst detection even three operators are used: one for the actual burst detection, one for determining the threshold value, and the third implementing time series forecasting. In the following we describe their implementation in detail.

## 3.2 Elastic Burst Detection

Elastic burst detection is the problem of finding all windows $w_i$ of arbitrary length $|w_i|$ at time $t$, such that for a monotonic aggregation function $A$ the aggregate $A(w_i)$ exceeds a window specific threshold $f(w_i)$. For convenience, we will refer to $A(w_i)$ using $A(w_i^s, w_i^e)$, where $w_i^s$ is the first element contained in $w_i$ and $w_i^e$ is the last one. In the following section we describe the burst detection approach proposed by Zhang and Shasha and our implementation of their work.

In [13] Zhang and Shasha introduced the *aggregation pyramid*, a triangular data structure over $N$ stream elements having $N$ levels. $N$ is the maximum window size over which a burst can be detected, and in each level, the level index $h$ corresponds to the length of the analyzed sequence, i.e., the size of the regarding window. Level 0 has $N$ cells containing the original data elements $y_1, \ldots, y_N$. Level 1 has $N-1$ cells and stores aggregates over two-element sets from the original data stream. The construction of the aggregation pyramid is recursive: Level $h$ has $N-h$ cells, and each cell $c(h,t)$ contains the aggregates from a window of length $h+1$ starting at element $y_t$. The aggregate of cell $c(h,t)$ is computed by

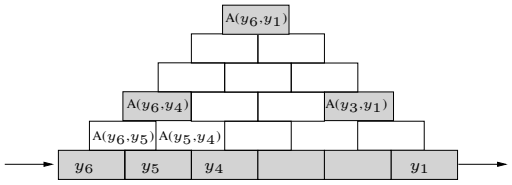



**Figure 1: An aggregation pyramid over a window of size 6 and the corresponding three level aggregation tree (shaded cells)**

$c(h,t) = A(c(h-1,t), c(0,t+h))$. Consequently, level $N-1$ stores the aggregate over the complete sequence. Figure 1 shows an aggregation pyramid over a window of size $N=6$.

The aggregation pyramid has some interesting properties. For instance, each of its cells $c(h,t)$ spans to a sub pyramid. The window starting at time $t$, which is composed of the following $h+1$ cells on level 0, is called the shadow of cell $c(h,t)$. Now it is straightforward to identify a burst. If a cell $c(h,t)$ exceeds the threshold for its shadow, a burst starting at time $t$ is discovered.

In order to efficiently identify the exact length and time of the burst, Zhang et. al additionally introduce a hierarchical tree structure called shifted aggregation tree. The nodes of this tree are a proper subset of the cells in the aggregation pyramid. There are at least two nodes at each level of the tree, except for the root level. The leaf nodes of the shifted aggregation tree correspond to the nodes at level 0 of the aggregation pyramid. A node on level $i$ of the aggregation tree represents the aggregate of its child nodes on levels $\leq i-1$. A three level aggregation tree is depicted in Figure 1 (shaded cells). Each node of the aggregation tree corresponds to one node of the aggregation pyramid.

A central property of burst detection using aggregation trees is that the aggregate of each subsequence of length $l$, with $l \leq h_i - s_i + 1$, is covered by the shadow of a node at level $i$. $h_i$ denotes the size of the window at level $i$ and $s_i$ is the shift at level $i$. That means, each window having size $\leq h_i - s_i$ is covered by at least one node on level $i$, and consequently each burst occurring in a window of length $h_i - s_i$ can be discovered.

For the detection of a burst within the aggregation tree it is sufficient to check if the aggregate in a node of level $i$ exceeds any of the thresholds for window sizes between $h_{i-1} - s_{i-1} + 2$ and $h_i - s_i + 1$. If a burst is discovered, a detailed search on the shadow of the node that exceeds the threshold is necessary.

Using the above method, we implemented a burst detection operator as part of our DSMS StreamDB. For efficiency, we only store the rightmost branch of the aggregation tree as a list. This is sufficient, as the nodes of the aggregation tree contain the aggregates for all possible subsequences of the sliding window over time as the stream progresses and the sliding window moves. The $N$ most recent stream elements, which are contained in the current sliding window, are maintained in a ring buffer. The height of the aggregation tree is $N/minSize$, where $N$ is the maximum window size and $minSize$ is the minimum length of a burst that can be detected.

## 3.3 Threshold Detection

A main problem of the introduced burst detection is the determination of a suitable threshold. If the threshold is too high, bursts are potentially not detected. Conversely, a threshold that is too low results in a high number of (maybe wrong) alerts. In addition, this increases the workload, as each time an alert is triggered all cells within the shadow of the regarding cell have to be investigated. And of course, a suitable threshold depends on the user and his actual requirements.

In [13] the authors suggest to use a heuristic in order to identify a static threshold. This method is based on the mean of the data seen so far. The drawback of this approach is that it is only appropriate

for stationary data with known distribution. Consequently, an algorithm using the static threshold cannot react correctly to trends or periods. One idea is to continuously forecast the stream in order to adapt the threshold. A popular and easy to implement method for achieving this is exponential smoothing.

In single (also called simple) exponential smoothing the forecast is the linear combination of the weighted average of the current observation and the previous smoothed observations. With this method, recent observations are more relevant for forecasting than older observations. A forecast value $S_t$ is predicted using the elements $y_i, i < t$, seen so far. Single smoothing is sufficient for short-range forecasting and static time series. For non-static time series showing a trend the double exponential smoothing (also known as Holt exponential smoothing) is appropriate. For example, if a room is continuously warming up due to a running radiator an alert is undesirable. In double smoothing, a forecast value $S_t$ and a trend $b_t$ are estimated. The trend can be used to predict the $k$th value in the future by $S_{t+k} = S_t + k \cdot b_t$. Triple exponential smoothing is suited to also handle seasonalities. [2] presents more details of this technique used for smoothing as well as forecasting. A main issue with them is to choose appropriate control parameters ($\alpha$ for single, $\alpha$ and $\gamma$ for double smoothing).

The idea is to use the techniques in order to determine a forecast value for each time $t$. The current stream element $y_t$ is compared to this value. If it is above this threshold it is regarded as an element of a burst. Single exponential smoothing can be updated incrementally in every time step, just like the mean. The problem is that both methods also include elements from bursts into the calculations, which reflects in too high thresholds. This in turn results in missing (parts of) bursts as we will show in Section 4. Identifying these elements and exclude them is possible, but rather expensive. This is because at the time a burst is detected some or all of its elements did already pass (depending on the size of the aggregation tree). Thus, they are already included into the calculations.

One approach is to use double exponential smoothing in order to implement a multiple step forecast. Once the smoothing variables are determined, they can be used for forecasting the following elements. This works fine for stationary data as well as data showing some trend.

The introduced methods take all elements of the stream seen so far into consideration. We refer to this as the *static* approach. As evolving data streams may change during time, distribution and trends may change as well. To handle this we also implemented *dynamic* methods, which are based on jumping windows. This means, at regular intervals all old elements are ignored and further threshold prediction is based only on new arriving elements.

Threshold determination and exponential smoothing are implemented as separate operators in StreamDB. These operators can be combined with the burst detection operator to continuously pass on accurate thresholds for burst detection.

## 3.4 Non-Stationary Burst Detection Algorithm

In the following we describe the general burst detection algorithm (Algorithm 1) that uses the exponential smoothing. Although there exist three separated operators, for clarity we present only one all-encompassing algorithm. As presented here, the algorithm works for both, single and double exponential smoothing. Thus, it is capable of handling linear trends. Which method is actually used is controlled by the input variables. For static double smoothing we use $smoothcont = false$. Like this, we determine the smoothing values $S$ and $b$ only once in the beginning using the first $cnt$ elements of the stream. Dynamic exponential smoothing refers to a re-computation of the smoothing values based on jumping windows. Thus, only in this case we set $smoothcont = true$ and $dynamic = true$. Single exponential smoothing is always computed incrementally in each time step (assured by setting $cnt = 1$ and $smoothcont = true$) and can by static ($dynamic = false$) or dynamic ($dynamic = true$).

**Input**: new stream element $y_t$, time $t$, number $cnt$, bool $smoothcont$, bool $dynamic$
**Output**: elements of a detected burst, $\emptyset$ otherwise
*/* collect first cnt points */*
**if** $t = 0$ **then** $Y = \emptyset$; **end**
**if** $t \leq cnt$ **then**
  $Y = Y \cup \{y_t\}$;
  */* only update tree */*
  $burstdetection(-1, y_t)$;
  **if** $t = cnt$ **then**
    */* initial exponential smoothing on first points */*
    */* $S$ is the predicted $y_{t+1}$ */*
    $S, b = expsmoothing(Y)$;
    $k = 0$;
    **if** $dynamic$ **then** $Y = \emptyset$; **end**
  **end**
  **return** $\emptyset$;
**end**
*/* collect next cnt points */*
$Y = Y \cup \{y_t\}$;
*/* predict current value ($k$ time steps since computing $S$) */*
$\mu = S + k \cdot b$;
$f = \mu + 2 \cdot \sqrt{\sigma^2}$;
$k = k + 1$;
*/* update tree and check for bursts */*
$burst = burstdetection(f, y_t)$;
*/* recompute exponential smoothing */*
**if** $k \geq cnt \wedge smoothcont = true$ **then**
  $S, b = expsmoothing(Y)$;
  $k = 0$;
  **if** $dynamic$ **then** $Y = \emptyset$; **end**
**end**
**return** $burst$;

**Algorithm 1**: General algorithm

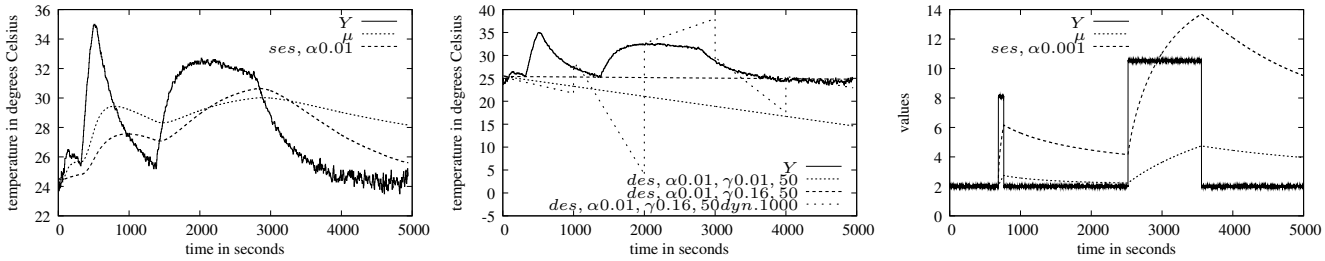
We start with an initial learning phase in order to get first parameters $S$ and $b$ from the exponential smoothing (implemented in method $expsmoothing$ – if single smoothing is used, $b$ can be an arbitrary value, because it is not used later on). The size $cnt$ of the window used for this should depend on the input data and the actual application scenario. How to determine an appropriate value for $cnt$ is one of the open issues we currently investigate. In parallel, we build the aggregation tree (included in method $burstdetection$). After the initial learning, we can start burst detection. This is also done by the method $burstdetection$, which runs the detection algorithm from [13].

The important point is the computation of the current threshold $f$. For this, we use the parameters from the exponential smoothing in order to predict the value $\mu$ expected for the current stream element $y_t$. The standard deviation $\sigma^2$ used for computing $f$ is determined from the $N$ elements in the regarded time window.

If we use dynamic smoothing the threshold computation is based on only short-term predictions, which can help to identify changes in the stream. Currently, we use a fixed interval of $cnt$ elements. We are investigating if this should be replaced by a more dynamic and flexible approach. Integrating triple exponential smoothing is straight forward and omitted here.

# 4. EVALUATION

The primary purpose of the following evaluation is to show the applicability of the different forecasting techniques in different situations. In addition, we measured the performance of the method




(a) Mean vs. single smoothing (b) Double smoothing (c) Mean vs. single smoothing stationary

**Figure 2: Heat sensor and stationary synthetic data**

running in our DSMS StreamDB. In the following figures, we plot the values of an input stream referred to by $Y$. The streams of real and synthetic data have an incoming rate of one element per second. To evaluate the different methods introduced before, we plot the thresholds detected by each of them. To identify the applied methods, we abbreviate single exponential smoothing by $ses, \alpha$ and double exponential smoothing by $des, \alpha, \gamma, cnt$. Determining the applied threshold on the basis of the simple mean, as proposed by [9], is identified by $\mu$. If we apply a dynamic method based on jumping windows we write `dyn` and indicate the used window size by a succeeding number. The issued queries are of the form

```
select * from Y
  [ burst-detection( wlen => 512,
    threshold => 'Holt Winters') ]
```

while altering the used method for threshold computation and window length (and optionally providing further method-specific parameters, where needed). The tests were run on a AMD Athlon 2 GHz PC with 512 MB main memory running a Linux operating system. In each figure all elements from the signal $Y$ positioned above the plotted threshold are detected as elements of a burst.

In Figures 2(a) and 2(b) we present results on real data from a heat sensor in our institute. To create two bursts, we placed a cup of hot water on top of the sensor. The first time, we removed the cup, causing a rapid cooling. The second time, we did not remove the cup und thus caused a rather moderate cooling.

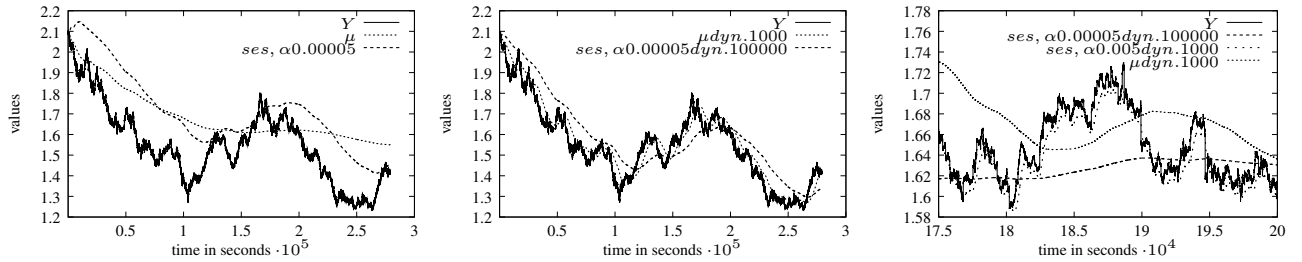
First, we compare the mean method with single exponential smoothing, both run in static mode (Figure 2(a)). As expected, the determined threshold follows the development of $Y$. Nevertheless, the bursts are identified correctly. Using static double exponential smoothing we can detect the length of each burst more accurately, see Figure 2(b). This is due to determining the smoothing variables $S$ and $b$ in a period of the regular signal and not changing it later on. Using the mean and single smoothing methods the burst periods influence the determined threshold accordingly. Figure 2(b) additionally highlights the issues of choosing appropriate parameter values: whereas the static smoothing with $\gamma = 0.16$ fits very well, using $\gamma = 0.01$ is a very bad choice. The threshold of the dynamic double smoothing with a window size of 1000 illustrates the problem of choosing a suitable window size and shows what can happen if smoothing is initialized during a period of burst. Then, the predicted trend results in totally wrong forecast values. On the heat sensor data, the static double smoothing with a suitable $\gamma$ is the best choice. This is expected, as the signal is rather stationary in regular times.

In Figure 2(c) the behavior of the mean and single smoothing methods on stationary data is illustrated again. The figure shows that bursts may be detected incorrectly in their length and that the thresholds need some time to decrease to the constant level again. Here, double smoothing would again fit perfectly, because the stationary character of the data is considered when determining the smoothing variables. Another possible solution is to exclude elements that belong to a burst from the computation of mean and forecast – with the problems of identifying these elements as discussed before.

The input data from Figure 3 represents the Swiss Franc-U.S. Dollar tickwise exchange rate data from the years 1985-1991[1]. The three figures perfectly show the suitability of the different methods for different needs, depending on the kind of bursts that shall be detected. Using static methods (Figure 3(a)) we get a rather high threshold, identifying only a subset of all peaks as bursts. This is suited to get a more coarse view on the data. Using dynamic methods (Figure 3(b)) results in a threshold that follows the original signal more closely. Like this, we detect more bursts and thus get a finer granulated view. This can even be more precise: in Figure 3(c) we zoom into a part of the stream and additionally plot the threshold determined with single smoothing and a very small window size of 1000. As illustrated, the threshold plot follows the $Y$ plot almost identically, which results in a lot of smaller bursts – providing a very detailed view on the input data. Using double exponential smoothing is unsuited on the tickwise data. The problem is that this data is neither stationary nor shows a clear trend. Here, it might help to incrementally update the smoothing parameters with each input element – which is exactly the same as using single exponential smoothing.

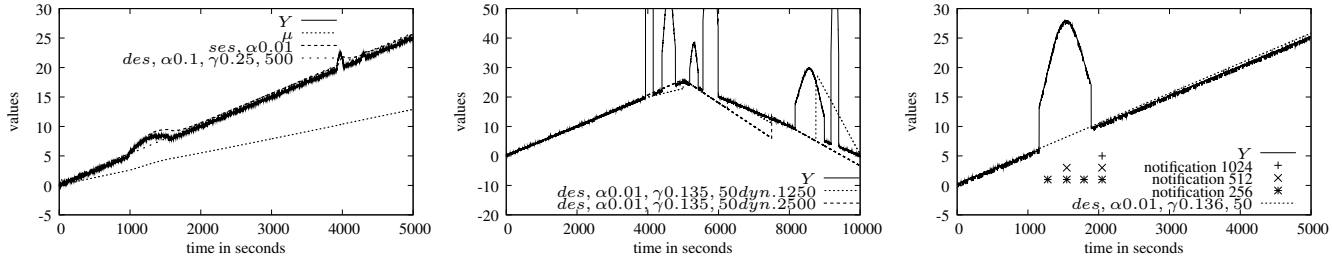
In Figure 4 we analyze the suitability of the double smoothing method in the presence of trends. We generated some synthetic data with a trend and some normally distributed bursts. Figure 4(a) shows that the mean method is absolutely unqualified for this, because we would detect the whole stream as one big burst. The single smoothing method fits better, because it follows the trend. But, in a burst the forecast values are influenced by that burst – resulting in detecting a wrong length of a burst. Here, static double smoothing helps. Due to the initial determination of the smoothing variables the threshold exactly follows the trend, resulting in detecting all bursts with their correct lengths. Of course this cannot work if the trend changes over time. In this case, the smoothing should be reinitialized when such a change occurs, which is another challenging problem. In Figure 4(b) we illustrate the suitability of different window sizes in combination with dynamic double smoothing for this task. This prevents from detecting the actual point of change, but results in the problem of choosing an appropriate window size. Both plotted methods are able to identify the current trend – but for each of them there are times where the threshold is rather inaccurate due to the inclusion of burst elements into its computation. In Figure 4(c) we illustrate the times of burst detection for different sizes of the aggregation pyramid. We used $N = 256$, 512, and 1024 as the size of the window comprising the level 0 of this pyramid. The points show only the time of notification, their value is chosen only conforming to the needs of clarity. As expected, with

[1] available at http://www.cs.ucr.edu/∼eamonn/iSAX/iSAX.html




(a) Mean vs. single smoothing (b) Dynamic mean vs. single smoothing (c) Dynamic mean vs. single smoothing /2

**Figure 3: Tickwise data**




(a) Mean vs. single vs. double smoothing (b) Changed trend (c) Notification times

**Figure 4: Synthetic data with trend**

a size of 1024 we detect only one big burst a short time after it occurred. With 512 we have two notification times and with 256 there are four, each time signalizing only a fraction of the whole burst. Summarizing the detected elements, the whole burst is detected completely by all three methods.

Our tests showed that the approach scales linearly in time and space with respect to the number of processed data streams. As expected, the used main memory scales absolutely linear. The used cpu time scales even less than linearly. The actual time needed for burst detection is negligible as well: we processed up to 10000 data streams with an incoming rate of two elements per second and did not experience any slow downs or delays.

# 5. CONCLUSION

In this paper, we showed how to combine threshold-based burst detection with time series forceasting methods in order to improve the accuracy of burst detection. The evaluation results presented above clearly show that there is no ultimate method. Rather, choosing the right method depends on the kind of input data and the bursts we want to detect (and in which granularity). In most cases, the mean and single smoothing techniques are appropriate. If the data contains a trend, the double smoothing technique should be used. If this trend is going to change, we have to switch to a window-based technique or combine it with some method for change detection. In any case the input data should be analyzed a priori and one should try to characterize it (and its potential future behavior) before choosing the method to apply. Open issues are the determination of suitable parameters for the smoothing techniques and the development of efficient methods for ignoring elements from bursts when determining forecasts.

# 6. REFERENCES


[1] ARASU, A., BABU, S., AND WIDOM, J. The CQL Continuous Query Language: Semantic Foundations and Query Execution. Tech. rep., University of Stanford, 2003.

[2] CHATFIELD, C., AND YAR, M. Holt-Winters Forecasting: Some Practical Issues. *The Statistician, Special Issue: Statistical Forecasting and Decision-Making 37*, 2 (1988), 129–140.

[3] ERGÜN, F., MUTHUKRISHNAN, S., AND SAHINALP, S. C. Sublinear methods for detecting periodic trends in data streams. In *LATIN* (2004).

[4] HINNEBURG, A., HABICH, D., AND KARNSTEDT, M. Analyzing Data Streams by Online DFT. In *IWKDDS-2006* (2006), pp. 67–76.

[5] KEOGH, E., LONARDI, S., AND CHI' CHIU, B. Y. Finding surprising patterns in a time series database in linear time and space. In *KDD'02* (2002), pp. 550–556.

[6] KLEINBERG, J. M. Bursty and hierarchical structure in streams. *Data Min. Knowl. Discov. 7*, 4 (2003), 373–397.

[7] PAPADIMITRIOU, S., BROCKWELL, A., AND FALOUTSOS, C. AWSOM: Adaptive, Hands-Off Stream Mining. In *VLDB 2003* (2003), pp. 560–571.

[8] SHAHABI, C., TIAN, X., AND ZHAO, W. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. In *SSDBM'00* (2000), p. 55.

[9] SHASHA, D., AND ZHU, Y. *High Performance Discovery in Time Series: Techniques and Case Studies*. Springer, 2004.

[10] VLACHOS, M., MEEK, C., VAGENA, Z., AND GUNOPULOS, D. Identifying Similarities, Periodicities and Bursts for Online Search Queries. In *SIGMOD 2004* (2004), pp. 131–142.

[11] WANG, M., MADHYASTHA, T. M., CHAN, N. H., PAPADIMITRIOU, S., AND FALOUTSOS, C. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *ICDE* (2002).

[12] YAMANISHI, K., AND TAKEUCHI, J.-I. A Unifying Framework for Detecting Outliers and Change Points from Non-stationary Time Series Data. In *SIGKDD 2002* (2002), pp. 676–681.

[13] ZHANG, X., AND SHASHA, D. Better burst detection. In *ICDE '06* (2006), pp. 146–149.

[14] ZHU, Y., AND SHASHA, D. Efficient Elastic Burst Detection in Data Streams. In *SIGKDD 2003, Washington, D.C., USA* (2003), pp. 336–345.