Quality of Service-Driven Stream Mining

Marcel Karnstedt, Kai-Uwe Sattler
Department of Computer Science and Automation,
TU Ilmenau, Germany
{marcel.karnstedt; kus}@tu-ilmenau.de

Dirk Habich, Wolfgang Lehner
Database Technology Group,
TU Dresden, Germany
{dirk.habich; wolfgang.lehner}@tu-dresden.de

Abstract

Scalable stream processing systems have to continuously manage changing resources efficiently, which is usually achieved by applying best-effort approaches on the level of processing operations. Thus, several authors have recently dealt with the problem of resource-aware stream processing, proposing methods and techniques capable of adapting to changing resources, both on the system and operator level. In this paper, we argue that Quality-of-Service (QoS) requirements are as mandatory as resource awareness when creating stream processing systems applicable in a wide and general range of data stream applications. We discuss QoS requirements and QoS-driven stream mining techniques as building blocks of the proposed framework for resource- and quality-aware stream processing systems. We exemplify the applicability of our approach by presenting and evaluating two QoS-driven clustering algorithms in detail.

1. Introduction

A growing number of application domains is characterized by data which is periodically or continuously produced by sensors. Among others, examples of such applications include traffic control, environmental surveillance, and manufacturing control. For the purpose of processing this data online, Data Stream Management Systems (DSMS) have been developed in the past which provide operations for filtering, aggregating and combining the data as well as query languages for specifying queries declaratively. On top of these query operators, the analysis of streaming data, i.e., the discovery of patterns and rules, by applying data mining techniques is often an important task. Here, the combination of mining and query operators provides the necessary flexibility to implement processing pipelines. Examples of such analyses are clickstream analysis, fraud and intrusion detection in telecommunications and others.

Main challenges of processing data streams lie in the (theoretical) infinity of the streams, the potential high arrival rate, and the processing restrictions regarding CPU time and memory. Thus, there are obviously two divergent objectives: the analysis results should be exact and comprehensive, but due to the stream characteristics and the resource restrictions, the analysis has to be performed on an approximation of the entire stream, i.e., a sketch / sample or a finite subset of the stream (window). However, relying on approximations affects the quality of the analysis: the derived mining model differs from the model we would get by analyzing the entire stream.

Thus, in order to get reliable instead of best-effort results, some kind of Quality-of-Service (QoS) guarantees are needed. QoS addresses two different levels:

- the system level of the DSMS with the parameters like tuple rate, CPU time, or memory consumption,
- the mining operator level referring to the quality of the resulting mining model (e.g., SSQ for clustering).

The difference between these two parameter classes is that system-level parameters are usually to be guaranteed by the system, whereas the mining level parameters either depend on the system-level parameters or they are given by the user. In the latter case, the values of system-level parameters have to be determined based on the specified QoS requirements. In this way, the DSMS either allocates the required resources or rejects the query.

Based on this model, we investigate in this paper the relationship between these two QoS levels. For this purpose, we discuss relevant parameters both on the system level and on the mining level, and we develop a mapping between them. This leads to a framework generalizing the idea of resource-aware and QoS-driven stream processing. We exemplify the mining level by selected clustering approaches, both representing an extension of the CLUStream algorithm [1]. Here, we discuss specific QoS requirements, present some more details and an extended evaluation, but focus on showing the applicability of our approach on the basis of a second, density-based, clustering method.

The remainder of this paper is structured as follows. In Section 2 we discuss QoS requirements on the system level of the DSMS. Quality measurements for mining tasks and the mapping between both introduced QoS levels are presented in Section 3. Afterwards, we propose two resource-aware clustering approaches with special focus on the QoS

requirements and challenges in Section 4. Experiments for evaluating the proposed mining approaches can be found in Section 5. We conclude our work with related work in Section 6 and a summary in Section 7.

2. QoS-Driven DSMS

A variety of current research activities are directed towards different aspects of managing data streams. Accordingly, lots of Data Stream Management Systems have been developed during the last decade. The challenges imposed by DSMS are manifold. The survey paper [2] presents a good and broad overview. Two main parts are crucial. First, DSMS operators must not be blocking, and second – even though the incoming data streams are infinite – operators are only allowed to maintain an internal state of limited size. In addition to the general DSMS requirements, real-time data stream processing is of particular interest in connection with QoS management. Therefore, the research focus of these systems has moved from the development of basic techniques for on-the-fly processing of streaming data to the (multi-query) optimization of DSMS, offering Quality-of-Service support for query evaluations and providing real-time support for time-critical application environments.

If quality-aware data stream processing is required, different aspects ranging from the QoS specification to the implementation of operator execution have been considered. Some DSMS like Aurora [3] and QStream [13] already provide useful solutions. Due to the widespread interpretation of the term 'quality,' a variety of QoS metrics have been introduced by existing DSMS. Some QoS metrics are dedicated to specific operators; others are more generic and thus applicable to the whole operator network. In general, the proposed QoS metrics can be classified into time-dependent and data-dependent metrics. The throughput (or data rate) is a typical time-dependent metric indicating the amount of stream input a DSMS is able to handle with bounded resources. The larger the value, the higher the QoS. The sampling rate, for example, is a data-dependent QoS metric describing the fraction of the input data stream that has to be sampled out.

The proposed DSMS are different regarding the negotiation and the guarantee of Quality-of-Service requirements. Based on the QoS specification, best-effort DSMS try to meet the user-given QoS requirements as closely as possible but they are unable to give any quality guarantees on time-dependent QoS metrics. In contrast, a QoS-guaranteeing DSMS is able to fulfill or guarantee a negotiated QoS requirement during the complete lifetime of the standing query. The only precondition is a successful QoS negotiation process. In this case, the required DSMS elements are resource management, QoS negotiation, resource reservation, resource monitoring, and an optimization with an appropriate strategy.

Figure 1 illustrates the process of mapping between resource constraints and quality measures. The focus of the work presented here lies on the QoS metrics dedicated to operators of the DSMS. These metrics and according QoS-driven operators are basic building blocks of DSMS in order to be able to guarantee a system-wide Quality of Service. Thus, in the next section, we discuss quality measures and QoS requirements for stream mining operators as an example for advanced operators of scalable and generally applicable DSMS.

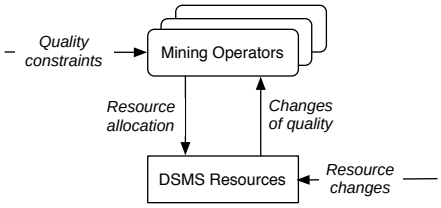


Figure 1. Quality Mapping

3. QoS-Driven Mining Operators

Operators for data stream mining represent a good example for complex operators utilized in DSMS. By implementing such complex tasks as operators, we allow for the creation of arbitrary query plans. These plans can combine several different operators, even more traditional ones (e.g., join, filter), with sophisticated techniques like clustering. Moreover, parts of multiple queries can be shared easily, which constitutes a prerequisite for building scalable and non-monolithic DSMS.

All resources available in a streaming environment, as well as all queries running in parallel, are managed solely by the DSMS. As resources and queries may vary over time, operators have to be resource-aware and they should be able to dynamically adapt to changing situations, always aiming for the most efficient utilization of available resources. In the context of QoS, the opposite direction becomes equipollently important as well. Operators have to provide quality guarantees to the DSMS under a certain amount of provided resources. The general goal is to provide both directions: *(i)* determine quality guarantees under provided resources, and *(ii)* calculate needed resources in order to achieve a provided minimum quality.

Depending on *(i)* specific QoS and resource requirements on the system level of a DSMS (memory, time, ...), *(ii)* the kind of adaptation process applied to actual operators, and *(iii)* the user's quality preferences (exact approximations, time granularity, ...), we have to handle different affected quality measures. In the next sections, we classify the most important of these quality measures (particularly those relevant for stream mining tasks) and discuss a concrete mapping from resources provided by the DSMS to qualities guaranteed by single mining operators, and vice versa. For the remainder of this paper, we focus on the main memory

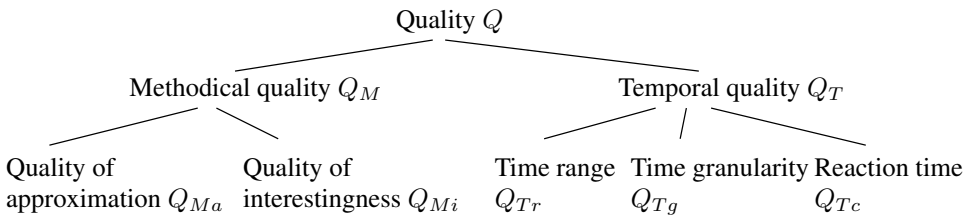


Figure 2. Different Quality Measures

as the most important resource in a scalable DSMS.

3.1. Quality Measures

In [6], we introduced several quality measures that apply to stream mining techniques in general. The resulting classification is pictured in Figure 2.

We identified two main classes of quality: one refers to the methodical quality of a mining technique (the most important and popular is the quality of approximation, e.g., the *Sum of Square Distances (SSQ)* for clustering techniques), the other refers to time sensitiveness – from our point of view, this is a mandatory prerequisite for each mining technique to be applicable in streaming environments.

The focus of this work is to answer the question of how clustering algorithms have to interact with stream processing systems in order to achieve QoS-driven stream clustering. In such environments, changes in the output quality of an applied clustering algorithm are due to changing resources provided by the stream processing system. Thus, we will introduce different approaches for adapting dynamically to changing resources and analyze which of the quality measures are influenced and how. Figure 3 summarizes the concrete quality measures we investigate during the remainder of this paper.

Q	**Output**
Q_{Ma}	SSQ
Q_{Tr}	maximal "look back time"
Q_{Tg}	minimal granularity
Q_{Tc}	minimal time till detection of changes

Figure 3. Investigated Quality Measures

3.2. Quality Mapping

In the past sections, we argued that Quality of Service is achieved by implementing resource- and quality-aware stream operators. For creating scalable QoS-driven DSMS, two ways of putting resource and quality awareness into practice become mandatory:

1. Claim specific quality requirements and deduce the needed resources to achieve this quality.
2. Limit the maximal memory available for processing and deduce the achievable quality.

Thus, in principle, we have to determine two (theoretical) kinds of functions:

1. $r : Q \rightarrow R$ - maps claimed quality to the resources needed, and
2. $\overline{r} : R \rightarrow Q$ - maps provided resources to the best achievable quality, as an inverse function to r.

More detailed, r is a function $r(args_x, Q_x(args_x), args_y, Q_y(args_y), \dots)$ taking all claimed quality measures $Q_x, Q_y, \dots$ and their factors as input, but we write $r(Q)$ for short. In contrast, $\overline{r}$ represents a bundle of inverse functions $\overline{r_x}, \overline{r_y}, \dots$, each corresponding to one quality measure $Q_x, Q_y, \dots$ Moreover, as we do not state the distribution of the stream elements as input factor for any function, r and $\overline{r}$ differ with different stream characteristics.

4. QoS-Driven Clustering

In this section, we will introduce two clustering approaches, both applicable for achieving QoS-driven stream mining as introduced before. Both approaches are based on *CLUStream* [1] introduced by Aggarwal et al. The idea of this work is to maintain a summary of the stream data, called the *snapshot pyramid*. The snapshots of this pyramid contain micro-clusters (similar to the idea of clustering features proposed in the *BIRCH* algorithm [15]), where more current snapshots are stored in shorter time intervals than older ones. Based on this data structure, the clustering of arbitrary time intervals can be processed as follows: a number of micro-clusters q, corresponding to the queried interval, is approximated from the snapshot pyramid and a static clustering algorithm (e.g., k-means) is applied on these q *micro-clusters* in order to compute k output clusters. From the work of Aggarwal et al. and our own experiments, we learned that the ratio between q and k has a strong impact on the memory consumption of the algorithm and perfectly reflects the approximational quality Q_{Ma}. Moreover, the utilization of the snapshot pyramid provides an alternative way for adapting to provided resources, reflecting different quality measures than Q_{Ma}.

As a first approach, we implemented a resource-aware clustering operator on the basis of CLUStream which follows the same idea of applying a statical k-means algorithm

on the q micro-clusters. The resulting algorithm extends the original CLUStream by adding resource awareness and a dynamic snapshot pyramid, utilizing memory more efficiently and adaptively right from the beginning of processing. As the usage of k-means is not satisfyingly accurate for each data distribution, we developed a second alternative technique relying on a density-based approach. Both algorithms are still time-sensitive, i.e., arbitrary time intervals can be analyzed at any point of processing. In the following, we will introduce the main idea behind both techniques and analyze their concrete impact on the proposed QoS-driven mining approach.

4.1. Adapting to Changing Resources

Before introducing the implemented clustering operators, we briefly describe how the adaptation to changes in the provided resources is achieved. Both techniques utilize a filling factor

$$f := \frac{\textit{actual memory usage}}{\textit{maximal memory provided}}.$$

Reacting to changing resources now means monitoring the filling factor and reacting accordingly. As a first idea, we distinguish between three different intervals for f:

1. $f < 0.85$: memory utilization is too bad.
2. $0.85 \leq f \leq 1.0$: memory utilization is fine.
3. $f > 1.0$: overfill situation.

For each situation, an individual reaction is processed, depending on the applied clustering technique.

This technique is applicable to a wide range of mining tasks, not only to clustering. In [6, 7], we proposed the same idea for frequent itemset mining and change detection tasks – which enforces the idea of creating a general framework for resource- and quality-aware stream mining based on the filling factor f. Resource and quality awareness are the main requirements on the operator level in order to create QoS-driven DSMS.

4.2. K-Means-Based Approach

As mentioned before, in [6], we proposed a first extension of CLUStream capable of dealing with changing resource constraints. The processing of the stream and the final clustering is analog to the original CLUStream but extended by a dynamic snapshot pyramid and resource-aware features. Thus, we omit details of the actual processing here. As this first solution is also based on the usage of a statical k-means algorithm, parameters that influence the amount of memory are: k: number of end-clusters, d: dimensionality of data, $[t_s, t_e]$: time interval for calculating the clusters, T: time elapsed since the beginning of the stream when the determination of the end-clusters starts, and Q_{Ma}: quality of clusters (in relation to the SSQ achievable with maximal q/k; equals 10 in our tests).

Resource awareness is implemented by reacting to the three different states of the filling factor introduced above as follows:

1. The width of the pyramidal time frame is increased by one.
2. The width of the pyramidal time frame remains unchanged.
3. The width of the pyramidal time frame is reduced, as long as f is above 1.0.

While in the first two cases, no deletion operations need to be processed, an overfill situation as in case three causes the deletion of snapshots. Reducing the actual amount of memory includes the following operations:

1. Decrease the pyramidal time frame width by one.
2. If an order i of the pyramid has more snapshots than the given width, delete the oldest snapshots until the width is equal to the number of snapshots.
3. Check the actual filling factor: if $f > 1$, go to 1, otherwise exit.

The procedure for adjusting the width in combination with the filling factor f is processed every time a snapshot is added to the pyramidal time frame.

To summarize, we identified q/k and the width w of the pyramidal time frame as main factors of resource consumption and resulting quality. The deduced formulas for $r(Q)$ and $\overline{r}(R)$ (see Section 3) are listed below – for details, we refer to [6].

$$r(Q) := q{\cdot}(2{\cdot}d{+}1{+}c_{LRU}){+}w{\cdot}\lceil log_\alpha(T)\rceil{\cdot}(q{\cdot}(2{\cdot}d{+}1){+}1)$$

The first formula approximates the number of floating point numbers (#FPN) needed in order to achieve the desired mining quality Q. This can easily be mapped to actual memory consumption in number of bytes. Q is represented by the values of q and w, d and T describe the current characteristics of the data stream, and α as well as c_{LRU} are constants provided by the CLUStream algorithm. Note that k influences the mining quality but has no direct impact on the memory consumption of the algorithm's online component. However, in order to achieve a certain cluster quality, the ratio q/k must be kept in corresponding intervals, which reflects q again.

$$\overline{r}(R) := w = \frac{R - q \cdot (2 \cdot d + 1 + c_{LRU})}{\lceil log_\alpha(T)\rceil \cdot (q \cdot (2 \cdot d + 1) + 1)}$$

The proposed algorithm adapts to changing resource limits R (again, in #FPN) by adjusting w, and thus the snapshot pyramid, accordingly. Hence, the formula for determining the achievable cluster quality computes the maximal w applicable under a provided resource limit R. On this basis, the following heuristic is applied: by varying the number of micro-clusters q from $10 \cdot k$ down to k, the corresponding average width w of the pyramidal time frame is calculated. If w is above 2 for a good choice of q ($3 \cdot k$ to $10 \cdot k$), acceptable clustering quality can be achieved. Otherwise, w

or q/k are too small to guarantee near-optimal results. The user specifies QoS requirements by providing a certain quality level, where each level is mapped to a specific combination of quality parameters q/k and w – see Section 5 for a corresponding list of quality levels. The heuristic is used in order to ensure QoS guarantees during stream processing. The quality mapping is analogously used for approximating resource requirements based on a user-specified quality level as well as the formula for $r(Q)$ above.

4.3. Density-Based Approach

To detect cluster formations of arbitrary shape (spherical, drawn-out, linear, elongated etc.), we developed a temporal-density-based stream clustering approach. As this approach partly relies on different principles than CLUStream and the k-means-based approach introduced before, we present more details of the actual processing in Sections 4.3.1 and 4.3.2. Some initial QoS considerations regarding the data stream processing for this algorithm are done in Section 4.3.3.

4.3.1 Temporal CF Tree

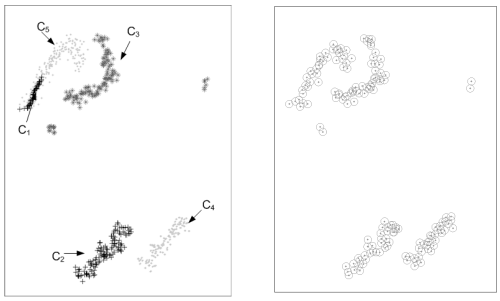
Our temporal-density-based stream clustering algorithm is a temporal extension of DBSCAN [5]. In our approach, we construct a temporal CF tree (TCF tree) to compress the data stream elements according to their spatial as well as temporal closeness. Therefore, the TCF tree is a slight adaptation of the CF tree. The TCF tree is based on the concept of the already introduced micro-clusters [1] by Aggarwal et al. In contrast to the well-known CF vector of the CF tree, the micro-clusters store summarizing temporal information on the data stream elements. The CF Additivity Theorem [15] is also valid for the TCF vector; the proof consists of straightforward algebra.

Thus, our TCF tree is a height-balanced tree similar to the CF tree with the following three parameters: (i) branching factor B, (ii) spatial threshold S and (iii) temporal threshold T. The temporal threshold T is an additional parameter compared to the CF tree, which is absolutely necessary in the data stream environment. The leaf and non-leaf nodes are similar to the CF tree but all entries in a leaf node have to satisfy the following:

1. the spatial threshold requirement with respect to the spatial threshold S: the spatial diameter or radius has to be smaller than S, and

2. the temporal threshold requirement with respect to the temporal threshold T: the temporal standard deviation has to be smaller than T.

The TCF tree will be built up dynamically as new data stream elements arrive. The insertion algorithm of the CF tree is slightly adapted for the TCF tree as follows. The appropriate leaf node for the new data stream element is located by recursively traversing the TCF tree on the basis of the spatial distance metric. When the appropriate leaf node is reached, the leaf entries are sorted according to the temporal distance metric. The temporal distance is the distance from the temporal centroid of an entry to the actual time stamp of the streaming element. Starting from the closest temporal leaf entry, the entries are tested to see whether or not the entry can absorb the new element without violating the spatial and temporal threshold conditions. The first entry that does not violate these conditions is updated to reflect the addition of the new data stream element. If no entry satisfying the two threshold conditions exists, a new leaf entry is created. If there is space in the leaf entry for this new entry, we are done; otherwise, we must split the leaf node. This node splitting algorithm corresponds to the original CF tree algorithm. The farthest spatial distance pair of entries is chosen to represent the first entries for the two leaf nodes, and the remaining entries are redistributed based on the closest spatial criteria. After the insertion of the new data stream element in a leaf entry, each non-leaf entry on the path from leaf node to root must be updated. In case of a leaf node split, we have to add a new non-leaf node entry into the parent node. If the parent node does not have enough space to add the new entry, we also have to split this non-leaf node and the procedure repeats for the parent of the non-leaf node.



(a) Data Stream Chunk (b) Leaf Entries

Figure 4. Example TCF Tree

A two-dimensional example is illustrated in Figure 4. The considered data stream chunk is depicted in Figure 4(a). The points marked as black are the first data stream elements arriving in the data stream chunk (Clusters C_1 and C_2). The next elements are marked by the dark gray color (C_3). The light gray illustrates data stream elements that arrive last in the chunk (C_4 and C_5). The data stream elements of the chunk produce clusters in different time intervals. The corresponding leaf entries of the TCF tree are depicted in Figure 4(b). The subclusters are represented by circles, where the center is the spatial centroid and the radius is the spatial threshold. The spatial radii of the subclusters are equal to or smaller than the spatial threshold S.

4.3.2 Temporal-Density-Based Clustering

Since not every leaf node corresponds to a temporally and spatially dense cluster, we require a clustering algorithm determining the final temporally and spatially dense clusters from the leaf entries. The leaf entries of the TCF tree represent subclusters, where the data stream elements are grouped together according to the spatial as well as the temporal closeness of the streaming elements with regard to the threshold parameters S and T. The set of subclusters $SC = \{s_1, \ldots, s_l\}$ is now the basis for the determination of the final clusters of the data stream using a temporal extension of DBSCAN [5].

For each subcluster s_i, the centroid $\mu_i^{spatial}$, the radius $r_i^{spatial}$, the temporal centroid $\mu_i^{temporal}$, the temporal deviation $\sigma_i^{temporal}$ and the number n_i of absorbed original data stream elements are known or can be computed from the corresponding TCF vector (micro-cluster). Furthermore, we know that each radius $r_i^{spatial}$ is equal to or smaller than the spatial threshold S. Moreover, we know that each subcluster s_i represents data stream elements, whose number is equal to or smaller than an upper boundary. This upper boundary is defined by the temporal threshold T of the TCF tree. This observation implies that a temporally and spatially dense region in the data stream chunk is represented by a set of overlapping subclusters (see Figure 4(b)). Another reason for high overlapping of subclusters is that in a data stream, a spatially dense region can occur at different points in time. In this case, the same spatial subclusters can exist but with different time information. From this observation, we can determine the final temporally and spatially dense clusters with a temporal extension of DBSCAN. According to DBSCAN [5], we can define core subclusters as follows:

Definition 1 (core subcluster) A subcluster $s_i \in SC$ is a core subcluster when $n_i \geq MinPts$, where n_i is the number of absorbed data stream elements in the subcluster s_i.

A core subcluster results only from the number of absorbed data stream elements. Therefore, we can use the $MinPts$ parameter to determine the subclusters to become core subclusters. For each subcluster s_i, we know the number of absorbed data stream elements n_i. A good heuristic for $MinPts$ is:

$$MinPts = \frac{\sum_{i=1}^{l} n_i}{l},$$

where l is the number of subclusters. The advantage of this heuristic is that the $MinPts$ parameter is adjusted to the result of each TCF tree. The $MinPts$ parameter may be different for different data stream stages. Next, we extend the definition of density reachability, direct density reachability and density connection [5] so that the temporal relationship can be regarded.

Definition 2 (direct time-density reachability) A subcluster $s_i \in SC$ is directly time-density-reachable from a subcluster $s_j \in SC$ if

1. $\mu_i^{spatial} \in N_S(\mu_j^{spatial})$,
2. s_j is a core subcluster, and
3. $|\mu_i^{temporal} - \mu_j^{spatial}| \leq maxTimeGap$

In other words, a subcluster s_i is directly time-density-reachable if the centroid $\mu_i^{spatial}$ lies in the S-neighborhood ($N_S()$) of the subcluster s_j with centroid $\mu_j^{spatial}$ if s_j is a core subcluster, and if the time distance between the subclusters is equal to or smaller than an application-specific $maxTimeGap$. We use the S-neighborhood instead of the r_j-neighborhood because of the expected high overlapping and to establish a parameter linkage with the TCF tree. The TCF tree parameter determines the spatial density.

Definition 3 (time-density reachability) A subcluster $s_i \in SC$ is time-density-reachable from a subcluster s_j if there exists a chain of subclusters $p_1, \ldots, p_n$, where $p_1 = s_j$ and $p_n = s_i$ such that p_{i+1} is directly time-density-reachable from p_i.

The time-density reachability is the transitive extension of the direct time-density reachability.

Definition 4 (time-density connection) A subcluster $s_i \in SC$ is time-density-connected to a subcluster s_j if there exists a subcluster s_k such that s_i and s_j are time-density-reachable from s_k.

Now, we can define our time-, space- and density-based notion of a cluster. A cluster is defined to be a set of time-density-connected subclusters which is maximal regarding the time-density reachability. Noise will be defined relative to a given set of clusters. Subclusters which do not belong to any clusters are considered noise.

Definition 5 (temporally and spatially dense clusters) Let SC be a set of subclusters. A temporally and spatially dense cluster C with regard to S, $MinPts$, $maxTimeGap$ is a non-empty subset of SC satisfying the following conditions:

1. $\forall s_i, s_j$: if $s_i \in C$ and s_j is time-density-reachable from s_i regarding S, $MinPts$, $maxTimeGap$, then $s_j \in C$ (Maximality).
2. $\forall s_i, s_j \in C$: s_i is time-density-connected to s_j regarding S, $MinPts$, $maxTimeGap$ (Connectivity).

The output of the clustering algorithm is a set of clusters $C = \{C_1, \ldots, C_m\}$, where the number of clusters may be

different. Each cluster C_i is represented by a set of subclusters $C_i = \{sc_i^1, \ldots, sc_i^{k_i}\}$, where k_i is the number of subclusters for C_i. Therefore, the cluster formations could be of arbitrary shape (e.g., spherical, drawn-out, linear, elongated, etc.). Furthermore, each cluster is only valid for a time horizon which can be computed from the TCF vectors of the subclusters.

4.3.3 QoS-Driven Processing

In the previous introduction sections on our time-density-based clustering approach, we did not discuss the processing of data streams. We combine this issue with QoS considerations in this section.

During the data stream processing, one TCF tree is kept in memory as a compact representation of the data stream elements. The final clustering results can be derived from this tree at any time with our present algorithm from Section 4.3.2. Thus, parameters that influence the amount of memory are the parameters of the TCF tree and data parameters: the branching factor B of the TCF tree, the spatial threshold S, the temporal threshold T and the dimensionality d of data.

Resource awareness is ensured by reacting to the three different stages of the filling factor f introduced in Section 4.1.

1. In case the memory utilization is too bad, we adjust the temporal and/or the spatial threshold. At the moment, we increase each parameter slightly until f is in the right interval.
2. If the memory utilization is fine, we do nothing.
3. If there is an overfill situation, three different strategies are possible, which are described next.

The most important situation for a reaction is the overfill stage, where the filling factor f is larger than 1. In this case, the size of the TCF tree has to be resized in order to collect new data stream elements. From our point of view, three different reaction strategies arise. The first strategy is to compress the TCF tree with adjusted spatial and temporal thresholds. This procedure influences the approximate quality of the clustering result Q_{Ma}. Therefore, this strategy can only be applied as long as the user-specified quality is not violated.

The second strategy is to delete some entries from the TCF tree. This procedure influences the time range quality metric Q_{Tr} because we reduce the captured time horizon of the data stream in the TCF tree. In this case, we need two additional tree operations: *Delete* and *Merge*. In data stream applications, very old streaming elements are considered less useful and relevant. Therefore, we can delete the leaf entries with the oldest temporal centroid. To determine the oldest leaf entries, we require one scan over all leaf entries. In general, the delete operation is simple because only the parent nodes from the leaf up to the root of the tree must be updated to reflect the deletion. The problem here is that the TCF tree can degenerate through insert and delete operations. To avoid high degeneration of the tree, we merge leaf nodes according to their spatial closeness.

The third strategy is a rather simple one: we store the TCF tree on disk and create a completely new TCF tree. In this case, the quality Q_{Tc} will decrease because disk access is much more expensive than memory access. Moreover, this has strong impact on the CPU time as another QoS-influencing resource. In order to process all data stream elements and to fulfill user quality measures, a combination of all three strategies is essential and possible.

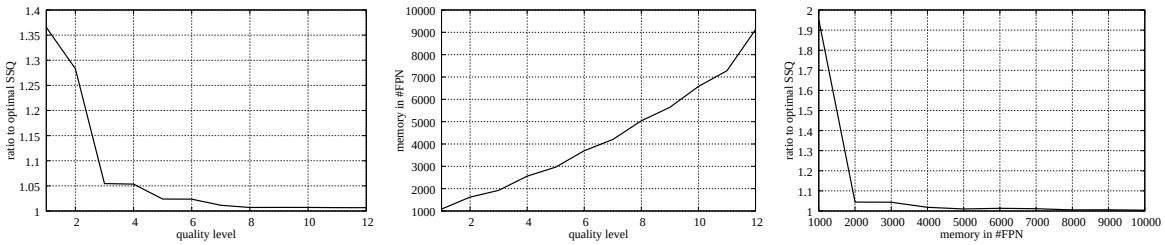
The necessary values for $r(Q)$ and $\overline{r}(R)$ can be approximated with a straightforward heuristic. Currently, we are developing a concrete formula for computing the size of the TCF tree. The heuristics applied on this formula follow the idea of the heuristics used in the k-means approach with an extension to respect the tree structure. The idea is to use the number of leaf entries in a TCF tree in order to reflect the number of nodes. Based on this, the computation of needed memory is done at ease. As in the case of the k-means approach, the computation of achievable quality measures from given resources is based on solving the formula for the respective parameters.

5. Evaluation

In this section, we evaluate main quality aspects of the proposed mining operators and the provided resource and quality awareness as main QoS requirements. The evaluation of the k-means-based approach is extended compared to prior works, whereas the density-based technique is still in the final development phase. Thus, this method is evaluated rather preliminary.

The k-means-based approach was evaluated on generated 2-dimensional data sets. We defined 5 cluster centers and a randomized procedure generated 25,000 points following a Gaussian distribution around these cluster centers. Additionally, in order to simulate evolving streams, we changed the centers in mean and variance every 5,000th point. Each test was repeated 5 times on such generated data and average values were measured. Internal parameters were set to $\alpha = 2$, $k = 5$, initPoints=1,000, snapshot saving interval=1s and input rate=200 points/s. In order to evaluate the resulting mining quality of the stream algorithm, a static clustering (LBGU) was run in order to compute a reference result and the corresponding SSQ SSQ_R. In the following figures, Q_{M_a} is represented by the ratio SSQ_S/SSQ_R, where SSQ_S stands for the SSQ resulting from the clusters identified during stream processing.

In a first series of tests, we wanted to show the method's resource awareness by proving its ability to dynamically adjust to changing resources. We set $q = 20$, started the pro-



(a) SSQ for Applied Quality Levels (b) Memory Consumption (c) Memory Effects on Quality

Figure 5. Quality of Service in the k-Means-Based Approach

cessing, and changed the provided resource limit every 250 seconds. As Figure 6 reveals, the algorithm perfectly adapts to meet the current memory limit provided, whether it is raised or lowered. The memory utilization is always at the upper limit and changed limits are met quickly and accurately.

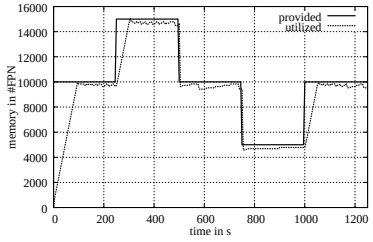


Figure 6. Resource Adaptation

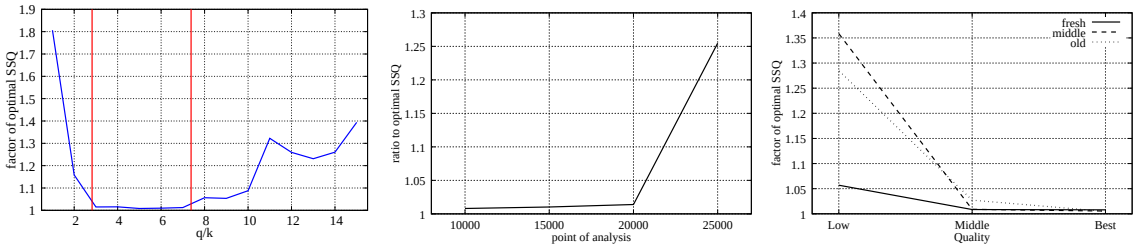
The second series of tests was run in order to analyze the QoS aspects introduced in this work. We use several levels of quality, which are mapped to a specific combination of quality parameters q/k and w. The mapping is illustrated in Figure 7. These levels are used according to the heuristic responsible for meeting QoS requirements introduced in Section 4.2.

w q/k	2	2.5	3	3.5	4	4.5	5
2	***1***						
3	***2***	***3***					
4		***4***	***5***				
5			***6***	***7***			
6				***8***	***9***		
7					***10***		
8						***11***	***12***

Figure 7. Applied Levels of Quality

In these tests, the interval from points 11,000-14,000 was clustered after the arrival of all 25,000 points. Figure 5(a) illustrates the exact SSQ ratios the different quality levels result in. Figure 5(b) shows the almost linear correlation between memory consumption and quality level. Moreover, the figure also shows the plot for the computed $r(Q)$ according to the formula introduced in Section 4.2. The approximation is satisfyingly accurate, which represents a mandatory prerequisite for providing QoS guarantees to the system and/or the user. The correlation between provided memory and resulting mining quality is shown in Figure 5(c). The algorithm was run under particular memory limits, and the heuristic to adjust the internal quality parameters during stream processing was applied. As expected, higher memory limits result in higher mining quality. It is striking that a very low memory limit results in very bad quality, but there is an early point from which on all results are in an acceptable range. Very high memory limits do not affect the output quality significantly; rather, the temporal qualities are affected because fewer snapshots are deleted from the pyramid. The plot for $\overline{r}(R)$ cannot be shown here, as the formula from Section 4.2 computes only w and is used in the introduced heuristic.

Our last test series aimed at evaluating different quality measures of the k-means-based approach. We expected to see the provided resource limit decrease the lower the time range quality T_R gets because older snapshots are deleted more often. Limited resources should also limit the interval of high-quality values of q/k, in contrast to the case when memory is not limited and the highest rate reflects the highest quality. In Figure 8(a), we see the results of varying the value of q and clustering the points 16,000-19,000 (a rather fresh interval) after 25,000 data points have arrived with a memory limit of 5,000 FPN. With q/k being too low (first vertical line), SSQ increases; oversized q/k (second vertical line) results in higher memory usage, which reduces the average width of the pyramidal time. The interval of q/k between the two above situations is characterized by good clustering results (0-5% beyond the optimum). Finally, a constant interval (points 6,000-9,000) was clustered at fixed points in time during continuous stream processing. We set q to 30 and the memory limit to 5,000 FPN again. The abruptly sinking quality from point 20,000 onwards signals that the clustered interval is "too old" from this point on, reflecting the time range T_R. Afterwards, we divided the synthetic data set into three different intervals, called *Fresh*, *Middle* and *Old*. The Fresh interval includes all points from

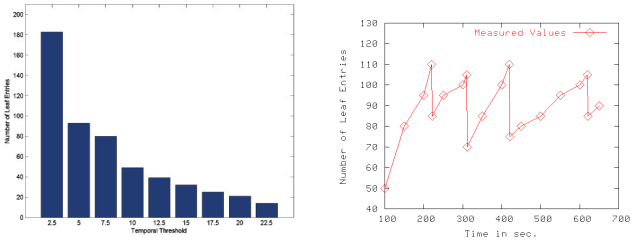


(a) Q_{M_a} under Constant Memory (b) Temporal Quality (c) Time Range for Different Qualities

Figure 8. Qualities in the k-Means-Based Approach

position 16,000-19,000, the Middle interval all from 11,000-14,000 and the Old one holds all from 6,000-9,000. The difference between fresh, middle and old results in quality differences if not enough memory is used. Due to the deletion strategy of the pyramidal time frame, old snapshots are deleted earlier than newer ones. This is why the temporal approximation of the pyramid gets worse and snapshots for older intervals cannot approximate the given time interval satisfyingly – see Figure 8(c). If the provided memory is high enough (middle and best strategy), old intervals can also be calculated with good quality.

To summarize, the k-means-based approach represents a really resource-adaptive as well as resource- and quality-aware extension of the CLUStream algorithm. The QoS requirements discussed in this paper are met perfectly, providing a basic building block for QoS-driven DSMS and a direction for defining a framework for QoS-driven operators in general.



(a) Number of Leaf Entries for Different Temporal Thresholds (b) Adaptation with Delete and Merge Operation

Figure 9. Evaluation of Time-Density-Based Clustering Algorithm

Due to the current early phase of our time-density-based stream clustering approach regarding resource awareness, we are only able to show some preliminary evaluation results. Figure 9 (a) illustrates the number of leaf entries for a data stream chunk against different temporal thresholds. The number of leaf entries is an indicator for the size of the TCF tree. The branching factor B and the spatial threshold S were fixed in the experiments. The number of all nodes can be approximated from the number of leaf nodes and the branching factor. As we can see, the larger the temporal threshold, the smaller the number of leaf entries (sub-clusters). With a large temporal threshold, a leaf entry can absorb more data stream elements in opposition to a small temporal threshold. The effect of the spatial threshold is identical and is already presented in [5]. This experiment indicates that the TCF tree can be reduced in size when the temporal and/or spatial threshold is increased.

In the second experiment, we evaluated our *delete* and *merge* operations as adaptation strategy. In this experiment, we set the maximal size of the TCF to 100 leaf nodes. If the actual number of leaf entries during the stream processing exceeded this maximal size, we deleted the oldest leaf entries until the number of leaf nodes had been reduced to 80% of the maximal size. To avoid degeneration, we then conducted our merge operation on the reduced TCF tree. As we can see in Figure 9(b), we are able to adjust the size of the TCF tree around the maximal size and thus, we can suitably react to resource limitations.

6. Related Work

Mining data streams has been an active research area for more than five years. Many algorithms have been proposed for extracting information from streams. Guha et al. [10, 11] analytically investigated the k-median technique for clustering data streams. The proposed algorithm requires a single pass over the data stream and uses only a very low amount of memory. The runtime is $O(n \cdot k)$ and the required memory is $O(n \cdot \epsilon)$, where k is the number of centroids/medoids, n is the number of points, and $\epsilon < 1$. They showed that any k-median algorithm that achieves a constant factor approximation cannot achieve a better run-time than $O(n \cdot k)$. O'Challaghan et al. [12] developed an algorithm for high quality data stream clustering. They assume that the data stream arrives in chunks and apply the LOCALSEARCH algorithms to cluster each chunk. The set of medians achieved from the chunks are clustered again with LOCALSEARCH to obtain the final k medians.

Aggarwal et al. [1] proposed a framework for clustering data streams called CLUStream. The algorithm is divided into two components. The first (online) component periodically stores summarized statistics about the data stream. The second (offline) component performs clustering on the summarized data according to user preferences like time frame and number of clusters. The final clusters are determined by using a modification of the k-means technique. In the online component, the data locality is stored in terms of micro-clusters. Each micro-cluster is a $(2 \cdot d + 3)$ tuple, where d is dimension of the considered data points. A total number of q micro-clusters are maintained, where q is usually significantly larger than the number of expected clusters. The initial q micro-clusters are determined in a pre-processing step using k-means. These micro-clusters are maintained during data stream processing. Whenever a new data point arrives, the micro-clusters are updated in order to reflect the change.

Recently, the idea of processing streams while adapting to resource and quality constraints has gained more attention. Several recent works deal with quality-aware online query processing applications in centralized and distributed systems, e.g., [4], but only few discuss concrete approaches and algorithms in stream mining scenarios, let alone the conjunction of resource adaptiveness and quality awareness.

In [8], the authors propose a resource-adaptive mining approach based on similar goals as our work. They suggest the adaptation to memory requirements, time constraints and data stream rate by solely adapting the produced output granularity. The paper also focuses on clustering but the authors do not consider quality aspects explicitly. In succeeding works, e.g., [9], they extend their approach by adding resource adaptiveness to the input rate and the actual data mining algorithm as well. However, the approach still lacks providing quality guarantees for the mining results.

As another example for resource awareness in stream mining, the algorithm RAM-DS proposed in [14] uses a wavelet-based approach to control the resource requirements. It is mainly concerned with mining temporal patterns and the method can only be used in combination with a certain regression-based stream mining algorithm proposed by the same authors. Although the proposed algorithm for mining temporal patterns is resource-aware, it is not resource-adaptive and does not provide guarantees for the quality of the mining results.

7. Conclusion

Scalable stream processing systems have to continuously manage changing resources efficiently. In this paper, we show that quality-of-service requirements are as essential as resource awareness. Those QoS requirements are mandatory to get reliable and not just best-effort results. Based on a proposed QoS model, we investigate the relationship between different QoS levels. For this purpose, we discuss relevant parameters for the various QoS levels and develop a mapping between them. In our description, we focus on resource-aware and QoS-driven data stream mining techniques, in particular, on data stream clustering. Aside from the presentation of two clustering algorithms, our work shows how to determine the consumed resources in order to provide a desired quality of service and the resource awareness of the presented techniques.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Clustering Evolving Data Streams. In *VLDB*, 2003.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. of PODS*, 2002.

[3] H. Balakrishnan, M. Balazinska, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, E. F. Galvez, J. Salz, M. Stonebraker, N. Tatbul, R. Tibbetts, and S. B. Zdonik. Retrospective on aurora. *VLDB Journal*, 13(4):370–383, 2004.

[4] L. Berti-Équille. Contributions to Quality-Aware Online Query Processing. *IEEE Data Eng. Bull.*, 29(2):32–42, 2006.

[5] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD*, 1996.

[6] C. Franke, M. Hartung, M. Karnstedt, and K. Sattler. Quality-Aware Mining of Data Streams. In *IQ*, 2005.

[7] C. Franke, M. Karnstedt, and K. Sattler. Mining Data Streams under Dynamicly Changing Resource Constraints. In *KDML: Knowledge Discovery, Data Mining, and Machine Learning*, pages 262–269, 2006.

[8] M. M. Gaber, S. Krishnaswamy, and A. Zaslavsky. Adaptive mining techniques for data streams using algorithm output granularity. In *The Australasian Data Mining Workshop*, 2003.

[9] M. M. Gaber and P. S. Yu. A framework for resource-aware knowledge discovery in data streams: a holistic approach with its application to clustering. In *SAC*, 2006.

[10] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams: Theory and Practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[11] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering Data Streams. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS) 2000, Redondo Beach, USA*, pages 359–366, 2000.

[12] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming-Data Algorithms for High-Quality Clustering. In *ICDE*, 2002.

[13] S. Schmidt, T. Legler, S. Schaer, and W. Lehner. Robust real-time query processing with qstream. In *Proc. of VLDB 2005*, pages 1299–1302, 2005.

[14] W.-G. Teng, M.-S. Chen, and P. S. Yu. Resource-aware mining with variable granularities in data streams. In *SDM 2004*.

[15] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *SIGMOD 1996, Montreal, Canada*, pages 103–114, 1996.