UPPSALA
UNIVERSITET

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 1431

Real-time data stream clustering over sliding windows

SOBHAN BADIOZAMANY

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2016

ISSN 1651-6214
ISBN 978-91-554-9698-2
urn:nbn:se:uu:diva-302799

Dissertation presented at Uppsala University to be publicly examined in ITC 2446, Lägerhyddsvägen 2, Uppsala, Wednesday, 23 November 2016 at 10:00 for the degree of Doctor of Philosophy. The examination will be conducted in English. Faculty examiner: Professor Tamer Özsu.

Abstract

Badiozamany, S. 2016. Real-time data stream clustering over sliding windows. *Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology* 1431. 33 pp. Uppsala: Acta Universitatis Upsaliensis. ISBN 978-91-554-9698-2.

In many applications, e.g. urban traffic monitoring, stock trading, and industrial sensor data monitoring, clustering algorithms are applied on data streams in real-time to find current patterns. Here, sliding windows are commonly used as they capture concept drift.

Real-time clustering over sliding windows is early detection of continuously evolving clusters as soon as they occur in the stream, which requires efficient maintenance of cluster memberships that change as windows slide.

Data stream management systems (DSMSs) provide high-level query languages for searching and analyzing streaming data. In this thesis we extend a DSMS with a real-time data stream clustering framework called *Generic 2-phase Continuous Summarization framework (G2CS)*. G2CS modularizes data stream clustering by taking as input clustering algorithms which are expressed in terms of a number of functions and indexing structures. G2CS supports real-time clustering by efficient window sliding mechanism and algorithm transparent indexing. A particular challenge for real-time detection of a high number of rapidly evolving clusters is efficiency of window slides for clustering algorithms where deletion of expired data is not supported, e.g. BIRCH. To that end, G2CS includes a novel window maintenance mechanism called Sliding Binary Merge (SBM). To further improve real-time sliding performance, G2CS uses generation-based multi-dimensional indexing where indexing structures suitable for the clustering algorithms can be plugged-in.

Keywords: Data streaming; Sliding windows; Clustering;

Sobhan Badiozamany, Department of Information Technology, Division of Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden. Department of Information Technology, Computing Science, Box 337, Uppsala University, SE-75105 Uppsala, Sweden.

ISSN 1651-6214
ISBN 978-91-554-9698-2
urn:nbn:se:uu:diva-302799 (http://urn.kb.se/resolve?urn=urn:nbn:se:uu:diva-302799)

To my family

List of Papers

This thesis is based on the following papers, which are referred to in the text by their Roman numerals.

I Badiozaman, S., Risch, T. (2012) Scalable ordered indexing of streaming data. *International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures*

II Badiozamany, S., Melander, L., Truong, T., Xu, C, Risch, T. (2013) Grand Challenge: Implementation by Frequently Emitting Parallel Windows and User-Defined Aggregate Functions. *The 7th ACM International Conference on Distributed Event-Based Systems*

III Badiozamany, S. (2014) Distributed multi-query optimization of continuous clustering queries, *VLDB PhD Workshop*

IV Badiozamany, S., Orsborn, K., Risch, T. (2014) Framework for real-time clustering over sliding windows. *SSDBM 2016 Conference on Scientific and Statistical Database Management*

Reprints were made with permission from the respective publishers. All papers are reformatted to the one column format of this book.

Table of contents

1 Introduction 11

2 Background and related work 13
2.1 Data Stream Management Systems 13
2.2 Real-time data stream clustering 15
2.3 Sliding windows 16
2.4 Indexing sliding windows 17
2.5 Two-phase aggregation over sliding windows 17
2.6 Two-phase clustering over sliding windows 19

3 Generic 2-Phase Continuous Summarization 22

4 Contributions 25
4.1 Paper I 25
4.2 Paper II 25
4.3 Paper III 26
4.4 Paper IV 26

5 Conclusion and future work 27

6 Summary in Swedish 28

7 Bibliography 31

Abbreviations

DSMS	Data Stream Management System
G2CS	Generic 2-layer Continuous Summarization
SBM	Sliding Binary Merge
PGS	Partial Grouped Summary
RM	Repetitive Merge

Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Tore Risch for his continuous support of my research, for his patience and encouragement. Thank you Tore for the sleepless nights we were working together before deadlines. I would like to thank my co-supervisor Kjell Orsborn for discussing research ideas and providing valuable feedback.

I thank former and current colleagues Ahmad Alzghoul, Andrej Andrejev, Matteo Magnani, Khalid Mahmood, Lars Melander, Silvia Stefanova, Thanh Troung, Cheng Xu, Erik Zeitler, and Minpeng Zhu for sharing their experience and helping me in teaching and research activities.

More specifically, I would like to thank Silvia, Minpeng, Matteo, and Thanh for all the fun, the barbeques and parties. Thanks for all the chats we had about ordinary life as PhD students, your friendship made me feel at home during the PhD studies. Thanh, I am also grateful for sharing your extensive knowledge of indexing.

Last but not the least; I would like to thank my family: Elham and Avina for supporting me spiritually throughout my PhD and my life in general. I would also like to thank my parents for their dedication and for many years of support during my whole life that has provided the foundation for this work.

This project was supported by EU FP7 Project Smart Vortex, the Swedish Foundation for Strategic Research, and eSSENCE under contract RIT080041.

1 Introduction

In the big data era, the data is produced at extremely high velocities and volumes. In many cases the data is in the form of data streams, making the approach of storing and querying the data offline infeasible. Examples of data streaming applications are urban traffic monitoring, stock trading, and industrial sensor data monitoring. To address these applications, Data Stream Management Systems (DSMSs) are used where queries continuously process data in real-time and emit output, as opposed to querying stored data.

Many data streams represent the current state of dynamic systems, e.g. geo-located streams of vehicle positions in an urban area, where knowing the current characteristics of the systems with low latency is highly desired. To enable real-time stream processing the DSMS should not fall behind the current stream and have a low response time, which requires efficient data indexing and stream maintenance mechanisms. Most DSMSs use main memory for query processing to meet low latency requirements.

To dynamically capture evolution of the underlying system, usually a window of most recent data is continuously queried where the window slides forward as a data stream progresses using continuous GROUPBY queries.

When the exact grouping of data is unknown conventional GROUPBY is insufficient because it is based on equality of group keys. In this case clustering algorithms are used, e.g. KMEANS [1] and DBSCAN [2], to form the groups and maintain the statistics. Clustering streaming data is particularly challenging because it involves dynamically merging and splitting evolving clusters over which statistical summaries are maintained in real-time as the stream progresses [3] [4] [5].

Most of the previous work on data stream clustering is focused on developing monolithic algorithms where the sliding window and indexing mechanisms are included in the algorithm, resulting in intertwined implementations that are not easily reusable. For example EXTRA-N [4] and SGS [5] have a spatial index on window contents: both windowing and indexing mechanisms are implemented within the clustering algorithm. BIRCH [6] is another example of a clustering algorithm that uses an application specific indexing technique, called CF-tree. Implementing data stream mining algorithms from scratch requires a very rare combination of skills so there is a need for high-level frameworks where data scientist can express data analyses on a high level, while complex algorithms, indexing and other low-level storage options can be reused and plugged-in beforehand [7].

In the context of this Thesis, clustering algorithms and conventional GROUPBY aggregation are different forms of *data stream summarization algorithms* as they both group and summarize the data streams.

This Thesis addresses the generic problem of data stream summarization over sliding windows in real-time by the *Generic 2-layer Continuous Summarization (G2CS)* framework where different summarization algorithms and indexing structures can be plugged-in.

The following research questions are investigated:

1. What is the most suitable window sliding mechanism for different kinds of stream summarization algorithms?
2. What is the suitable indexing mechanism for data summarization over sliding windows?
3. How can the window sliding mechanism be separated from both indexing and the applied summarization algorithm to avoid intertwined implementations?

To address the research questions we developed G2CS where we evaluated different implementation alternatives.

To address research question 1, in Paper II we analyzed a use case to investigate different query processing methods for GROUPBY queries. A two-phase approach [8] [9] [10] [11] [12] for stream summarization over sliding windows was then picked for further development. Based on this case study, in Paper IV a two phase approach is presented to efficiently support clustering queries using a novel window maintenance mechanism called Sliding Binary Merge (SBM).

Paper I addresses research question 2 by presenting a generic approach for indexing the data in sliding windows that continuously maintain GROUPBY aggregates by slicing both the data and the index in sliding windows. This approach is generalized in Paper IV to index the data required for maintaining dynamically changing clusters over sliding windows.

To address research question 3, in paper IV it is shown how G2CS separates sliding window maintenance and indexing from plugged-in summarization algorithms, which makes it significantly easier for data scientists to perform clustering over data streams. G2CS allows for re-use of software components and simplify the introduction of new algorithms.

This Thesis overview is organized as follows. Chapter 2 presents the technology background and reviews the related work. Chapter 3 presents G2CS, the framework that implements all the contributions. Chapter 4 states the contributions made in each of the four papers, Chapter 5 presents possible future directions of this Thesis work, and the Thesis summary in Swedish is presented in Chapter 6.

2 Background and related work

In this chapter first general technologies and definitions for real-time data stream analysis are presented. Then approaches for real-time data stream mining related to G2CS are discussed in details.

2.1 Data Stream Management Systems

A data stream is a continuously extended sequence of tuples, which is usually ordered, commonly by tuple arrival time or tuple number. In most scenarios, due to high data volume and velocity, the elements can be read only once. Data streams are produced for example by sensor readings from machines [13], live update of stock prices [14], and spatio-temporal readings from a number of moving objects [15] [16]. The massive amount of data produced by data streams need to be systematically processed and analyzed.

There are a number of systems, e.g. Storm [17], Amazon Kinesis [18], MillWheel [19], Flink streaming [20], and Microsoft Streaminsight [21] for high-performance data stream processing using a regular programming language where a processing pipeline is explicitly programmed. In general, it is desirable to have a high level query language to analyze data streams, as in Streambase [22], SQLStream [23], and Gigascope [24]. This is because, similar to processing data in conventional applications, a high level query processor enables data analysis for non-programmers since the users express "what" information is to be retrieved instead of "how" to retrieve it.

Data Stream Management Systems (DSMSs) [25] are software systems that manage and support querying of continuous data streams. Example DSMSs are STREAM [26], TelegraphCQ [27], Gigascope [24], and Super-Computer Stream Query processor (*SCSQ*) [28]. Since data streams are unbound and echo the changing behavior of a monitored system, the query model in a DSMS reflects this dynamic behavior using *continuous queries* [29] where, unlike traditional database queries, the result continuously changes as the stream progresses. Continuous queries run and emit updated output until they are explicitly terminated by the user.

Data streams often have high volume and velocity, so DSMSs need to meet the following requirements:

- The arriving stream elements have to be processed on-the-fly.

- Processing data streams usually is done in a single pass, in contrast to regular query processing methods that rely on visiting tuples multiple times.
- To be able to keep up with the stream flow, the processing engine needs to have low latency and high throughput. Therefore, the stream processing engines often operate in main-memory and use responsive main-memory indexing structures.

Event driven systems [30] are similar but optimized for analyzing complex event patterns using a reactive language.

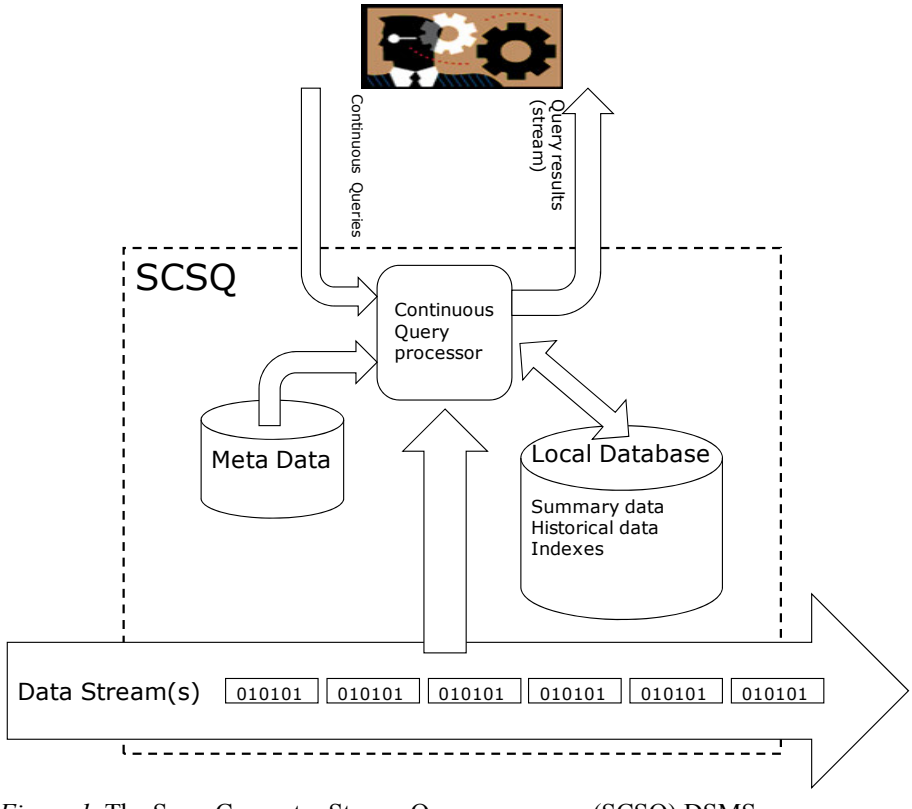

Figure 1. The SuperComputer Stream Query processor (SCSQ) DSMS

Figure 1 illustrates the architecture of SCSQ. The *meta-database* stores the schema used to express queries. The *local database* maintains in main-memory data representing current data summaries as well as historical data using main-memory indexes for efficient search. The *continuous query processor* optimizes and executes continuous queries over the data streams by accessing the meta-data and the local database. Continuous queries can span both streaming and local data.

This Thesis work extends the DSMS SCSQ to process real-time data mining queries over sliding windows.

2.2 Real-time data stream clustering

In many applications, complex data mining algorithms are applied on the stream in real-time to find current data patterns. For example, automated financial systems use real-time modeling and monitoring of the stock price trends for predictive applications. Another example is monitoring urban traffic in real-time where the data streams produced by connected vehicle GPS systems are analyzed to detect traffic regions by forming clusters of cars.

Traditional data clustering algorithms such as K-means [1], Self Organizing Maps [31], density based clustering techniques such as DBScan [2] and CLIQUE [32], are applied on finite static data. This allows for several passes through the stored data. In contrast, because data streams are infinite, data stream mining algorithms need to process the data in single pass as in Denss-tream [3], BIRCH [6], Extra-N [4], SGS [5]. In addition to the single-pass requirement in data stream clustering, real-time data stream clustering requires detecting the continuously evolving clusters formed as the stream progresses. Responsive real-time cluster detection requires early detection of clusters as soon as they occur in the stream, which is addressed by G2CS. For example, a moving car can use traffic data streams to form clusters of cars in the road to actively detect traffic congestion ahead and slow down. In this case responsive detection of clusters is essential for safety reasons.

While data stream clustering can be done using data stream processing engines (e.g. Storm, Kinesis, Stream Mill, Flink Streaming, stream insight, IBM system S), DSMSs are better fit for data stream clustering because the end users can express the clustering task in terms of high level queries. For example, the following query [Paper IV] detects congested areas with radius 50 meters over a window of vehicle positions X and Y using a modified version of the clustering algorithm BIRCH [6] for sliding windows, C-BIRCH.

```
SELECT CENTER(cid), COUNT(cid)
FROM   VEHICLE_POSITIONS (RANGE = 10, STRIDE = 2)
WHERE  SPEED<30
CLUSTER BY X, Y AS cid
USING C-BIRCH(50)
```

Here, C-BIRCH is an algorithm that is plugged into the DSMS and the FROM clause specifies a sliding window, which is discussed in the next section. G2CS allows for plugging-in clustering algorithms such as C-

BIRCH and executing them with low response times over sliding windows. The plug-ins can be defined in terms of high level queries to the local main-memory database. This simplifies the implementation of the algorithms and improves their performance by utilizing query optimization techniques, e.g. automatic index utilization.

2.3 Sliding windows

To enable processing of blocking operators like average and sum over infinite data streams, *windowing* is commonly used in data stream processing because it limits the extent of data to a sequence of most recent elements in the data stream. Clustering algorithms are blocking since they require having all the data points to compute the clusters. Therefore, windowing is needed for data stream clustering.

There are different ways of defining a window over a data stream [33]. The *window specification* defines how recent stream elements are selected for windowing in the FROM clause of a continuous query. When the window specification is applied on a live data stream it produces new *window instances* at different points in time. A window instance logically contains a set of stream elements.

For example, a *sliding window* is specified by defining its *range* and *stride*. The range R of a sliding window specifies the length of the window while the stride S specifies the portion of the range that is evicted from the window when the window moves forward. A sliding window is specified as a tuple $<R,S>$, where $S<R$. Two common kinds of sliding windows are *time-based* and *count-based* sliding windows. In time-based sliding windows R and S are defined using time intervals while in count-based sliding windows they are defined in terms of the number of elements. For example a time based sliding window with $R=10min$ and $S=2min$ produces window instances that cover the data in the last 10 minutes of the stream and a new window instance is created every 2 minutes. Without loss of generality, we present sliding windows using time-based sliding windows.

Real-time data stream clustering can be done using sliding windows because they reflect the recent elements in the stream. To responsively detect rapidly changing clusters, a smooth sliding specification is highly desired where the stride S is small relative to the range R, i.e. the *partitioning ratio* $PR=R/S$ is high. G2CS provides a sliding mechanism that allows for efficient processing of clustering algorithms with high PR [Paper IV].

2.4 Indexing sliding windows

The contents of window instances can be stored as a sequence of data items in a main-memory buffer. However, when the execution of continuous queries involves searching the contents of the window instance, the search can become expensive if there are many elements in it, e.g., finding vehicles with a particular pattern in their plate number in a window of a live traffic stream, or finding the activities of certain mobile phone users within the past hour. Data indexing [34] is a general technique that provides efficient search in databases which can also be applied on indexing large data stream windows [35].

Real-time data clustering algorithms often require continuous search for similar objects in a multi-dimensional space. Calculating the similarity of all the objects in a large window can be prohibitively expensive; therefore multi-dimensional indexing that supports efficient similarity search is required for responsive real-time query processing. Each clustering algorithm might have a different indexing structure so it is desirable to be able to plug-in different indexing data structures.

Indexing the contents of sliding window instances brings about the following challenges:

- The indexing structure need to support very high insertion rates to be able to add the arriving data to the window.
- They also need to have a high performing deletion mechanism to evict expired data.

In data stream processing main memory indexing structures need to be used to keep up with the stream flow.

G2CS provides a method for plugging-in algorithm-specific main-memory indexing of the elements in window instances, while supporting high insertion rates and a bulk deletion method [Paper I]. For responsive clustering over sliding windows, G2CS uses a general indexing framework described in [Paper IV] that separates the indexing from both the applied clustering algorithms and the sliding mechanisms. This is shown to significantly improve the response time.

2.5 Two-phase aggregation over sliding windows

Figure 2 illustrates how data overlaps when windows slide. A window instance $W_{b,e}$ represents the state of the window W during the valid time interval *[b,e)*. In Figure 2 the data in window instance $W_{0,10}$, covering the time interval *[0,10)* overlaps (gray boxes) with the data in the window instance $W_{2,12}$ covering *[2,12)*. The window instance $W_{2,4}$ is a common *partial window* instance of the complete window instances $W_{0,10}$ and $W_{2,12}$.

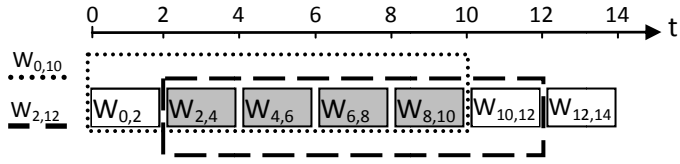

Figure 2. Data overlap in sliding windows

To avoid unnecessary recomputations when data is summarized over sliding windows, efficient *differential maintenance* techniques can be used [36]. For example in Figure 2, the summary for $W_{2,12}$ can be obtained by adding $W_{10,12}$ to the summary for $W_{0,10}$ and removing $W_{0,2}$ from the summary for $W_{0,10}$.

Differential processing is usually done by introducing functions for adding/removing deltas to/from the aggregation state [36]. For example, the aggregate function COUNT is differential because both of the following equations hold:

$$COUNT(A\ B) = COUNT(A) + COUNT(B)\ \textit{(Incremental)}$$
$$COUNT(A \backslash C) = COUNT(A) - COUNT(C)\ \textit{(Decremental)}$$

Here A, B, and C are sets. Using the incremental and decremental properties, the summary for a sliding window is differentially maintained as follows. When a slide happens the summary of the most recent window instance is reused by adding the incoming elements using the incremental function and removing the expired elements using the decremental function.

For many data stream summarization algorithms, e.g. clustering algorithms, there is no decremental function, e.g. in BIRCH. Even when a decremental method can be devised as in [37], it can be very expensive and must be avoided, as suggested by previous work [4]. G2CS provides a novel window maintenance technique for efficiently maintaining summaries without requiring the decremental function [Paper IV].

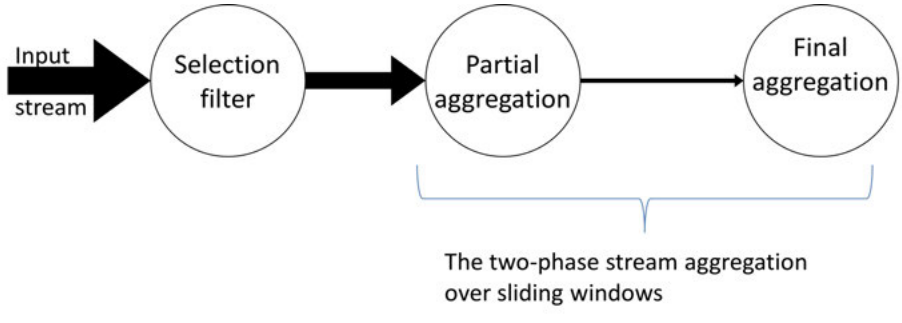

Figure 3. Sliding window aggregation

Figure 3 shows the widely used *two-phase aggregation* technique [8] [9] [10] [11] [12] for computing aggregation over sliding windows. The thick-

ness of the arrows in the figure indicates the volume of the stream. First, the *selection filter* removes the stream elements that are outside the query selection criteria. The second and third processes constitute the two-phase aggregation. After applying the selection filter, the two-phase aggregation summarizes the stream per group as follows:

1. In the first phase, called *partial aggregation*, fine-grain non-overlapping partial window instances are formed where aggregated data is accumulated.
2. The second phase, called *final aggregation*, combines consecutive aggregates from the first phase to produce the total aggregate over the complete window instances.

With the 2-phase window maintenance approach the performance is improved because the incremental property of an aggregate function enables pushing down incremental computations into the first phase, thus reducing the data volume in the second phase. Second, the decomposition allows distributed and parallel processing since phase one and two form a pipeline [24]. In the 2nd phase, at every slide, the incremental property enables a partial aggregate to be *merged* into the total aggregate, while the decremental property allows the contributions of expired partial aggregates to be *excluded*.

G2CS extends the two-phase approach to also support non-decremental summarization algorithms such as clustering algorithms by introducing a sliding mechanism for responsive maintenance of evolving clusters in the second phase for clustering algorithms that do not have the decremental property. The first phase is similar for both clustering and aggregation, only the aggregated data is algorithm dependent.

2.6 Two-phase clustering over sliding windows

We note that the 2-phase approach is also beneficial for clustering-algorithms, where expensive cluster formation can be done in phase one and the formed partial clusters are combined using the clustering algorithm in phase two. However, there is a fundamental difference between GROUP-BY queries and clustering queries, which has implications on how the two phase approach is implemented. In GROUP-BY queries, the groups on which aggregate functions are applied are formed based on equality of grouping keys, whereas clusters are formed based on algorithm dependent similarity between data points. Therefore, a window slide in a GROUP-BY query does not move elements between groups. In contrast, for clustering algorithms the window slides dynamically change cluster memberships as clusters might merge or split when new data arrives or old data expires.

Figure 4a shows an example of dynamic group membership in clusters *a1*, *a2*, and *a3*. Figure 4b illustrates two arriving points (green) and two expired ones (red). The resulting point-to-cluster memberships in Figure 4c are completely new.

G2CS allows for such dynamic change of group memberships for clustering algorithms by allowing for combined group formation and data summarization [Paper IV].

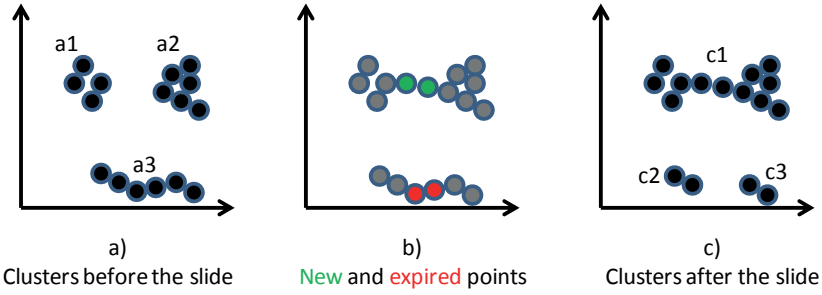

Figure 4. Evolving group memberships in clustering algorithms over sliding windows

The dynamically changing group memberships in clustering algorithms has the following implications on how they are processed over sliding windows, compared to conventional GROUP-BY queries:

a. Streamed clustering algorithms require grouping and aggregation to be combined, whereas group formation mechanisms in GROUP-BY queries are implemented by first splitting the stream based on the group key in a grouping operator followed by an aggregation operator [8] [38]. Therefore stream clustering algorithms needs to maintain their own data structures to represent clusters that are updated as the window slides.
b. For many clustering algorithms, incremental deletion of data points from clusters is not defined [39], i.e. they are not decremental. Even when a decremental method can be devised as in [37], it can be very expensive and must be avoided, as suggested by previous work [4].
c. Efficient grouping by similarity in streamed clustering algorithms require multi-dimensional indexing to find which clusters are influenced by a regrouping, while streamed GROUP-BY queries can hash on fixed group keys.

G2CS addresses a. by allowing clustering algorithms to store multiple generations of summarization data as the cluster memberships evolve over time with window slides. G2CS addresses b. by a novel window maintenance technique called Sliding Binary Merge (SBM) which is very efficient when the applied summarization function is non-decremental allowing for respon-

sive slides. To address c. G2CS provides transparent index plug-ins to speed up the multi-dimensional search in clustering algorithms, which improves the response time.

In related work [40] [4] [5] [41], to support non-decremental clustering algorithms, the summary in each partial window instance, here called *Partial Grouped Summary (PGS)*, is *repetitively merged* into all complete windows it is part of. The repetitive merge (RM) approach is illustrated in Figure 5 where a sliding window of range R=10 and stride S=2 is formed in the 2nd phase. When PGS_5 arrives, it is merged into the five complete window instances $W_{0,10}$, $W_{2,12}$, $W_{4,14}$, $W_{6,16}$, and $W_{8,18}$. This causes redundant computations, e.g. both $W_{8,18}$ and $W_{10,20}$ merge all the common partial summaries $PGS_6 - PGS_9$.

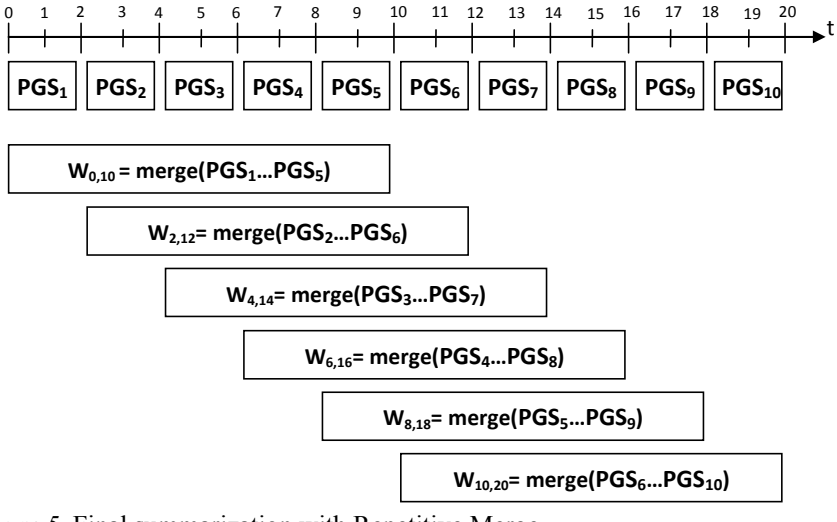

Figure 5. Final summarization with Repetitive Merge

With repetitive merge the number of merges per slide becomes high when *PR* is high, which substantially decreases responsive detection of clusters, as shown in [Paper IV]. Unlike the repetitive merge approach, G2CS avoids the overlapping merges by maintaining small intermediate data stream summaries and organizing them using SBM.

3 Generic 2-Phase Continuous Summarization

The different user roles in G2CS are shown in Figure 6. The *data scientists* are the end users submitting *continuous queries* to G2CS to find clusters and perform other analyses. The *algorithm designers* implement new stream summarization algorithms and plug them into the system. Plug-ins can be written to support either conventional GROUPBY aggregation or complex clustering algorithms; the latter is the focus of this Thesis. Clustering algorithms often require specific indexing for responsive cluster maintenance as the clusters evolve. G2CS allows arbitrary indexing structures to be plugged-in by *indexing experts*. By defining plug-ins in terms of queries over the main-memory local database [Paper IV], the algorithm design is separated from index implementation. This is because the query optimizer automatically utilizes indexing [42] when executing the queries over the local main-memory database.

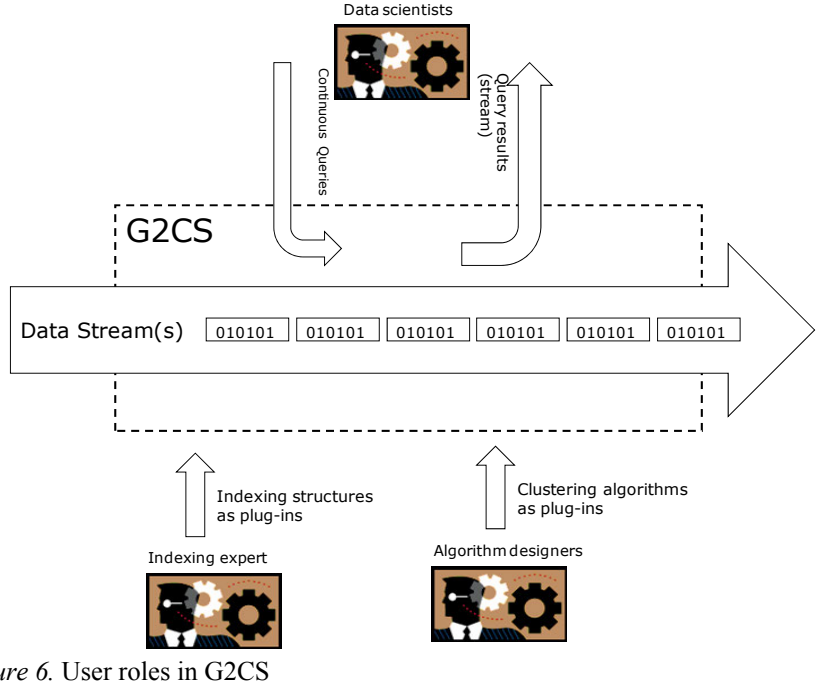

Figure 6. User roles in G2CS

Figure 7 illustrates the architecture of the framework. It utilizes previous work on query processing [43], data stream management [28], and extensible indexing [42]. The contributions of this Thesis are the modules that are blue-shaded in the figure. The *query processor* receives continuous queries and produces execution plans that invoke the G2CS kernel.

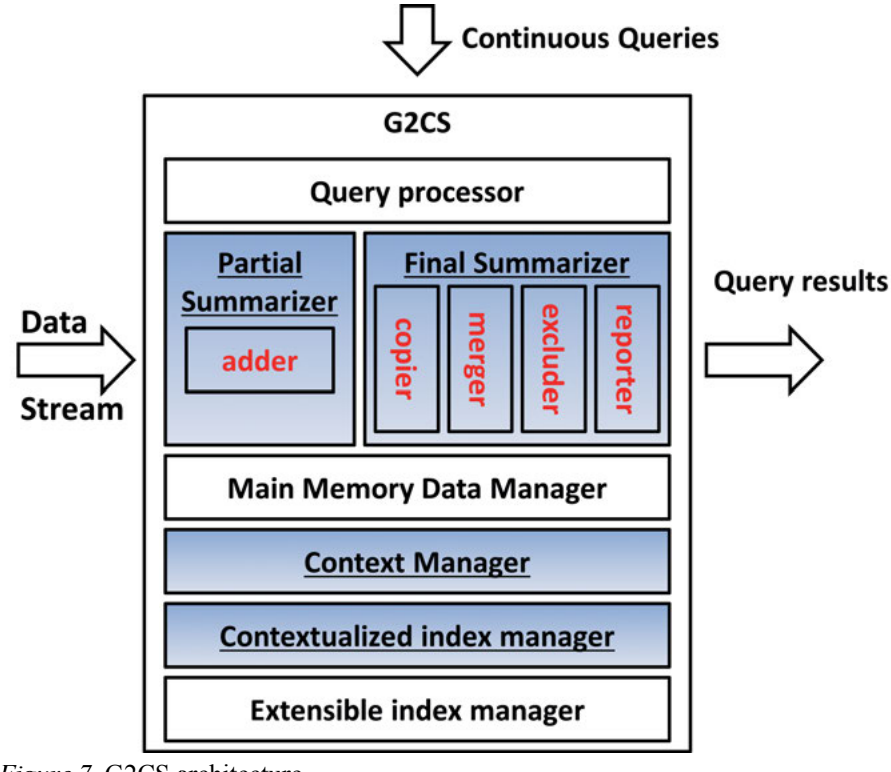

Figure 7. G2CS architecture

The *context manager* organizes window instances by contexts. A context represents the valid time interval of a window instance as a triple <*b, e, cxtid*> where *cxtid* is a unique *context identifier* of the time interval *[b,e)* per window. Contexts are allocated by the context manager and their identifiers are passed to the plugged-in clustering algorithms. The *contextualized index manager* in G2CS maintains an index per context in the local database and separates indexing from the sliding mechanism and the plugged-in clustering algorithms.

The *partial summarizer* implements the first phase of clustering over sliding windows. As new data arrives, it slices the incoming stream into partial window instances. It then assigns a new context for each new partial window instance and iteratively calls the *adder* plug-in for each arriving data point to incrementally populate summary data for the context identifier.

When the summary data for the partial window instance is fully populated the *final summarizer* is called, causing the sliding mechanism to be invoked in order to form and emit the clusters in a complete window instance. The final summarizer implements the second phase of the clustering. For differential algorithms the user can provide methods for both incremental maintenance of clusters (*merger* plug-in) and decremental ones (*excluder* plug-in). For non-decremental algorithms it is crucial to avoid the repetitive merge approach in the final summarizer because of its redundant calculations. In this case G2CS maintains and reuses several layers of intermediate window instances by organizing them by contexts using SBM.

Since SBM maintains a number of intermediate window instances to optimize the sliding mechanism, the *copier* plug-in is invoked to populate new window instances by copying data from old to new window instances. Then G2CS makes a number of calls to the *merger* and *excluder* plug-ins to generate complete window instances. By calling the copier calls prior to the merger and excluder, G2CS retains the necessary old and intermediate window instances. The *reporter* plug-in extracts the data to be emitted from a complete window instance. An incremental garbage collector deallocates summary data for window instances whose contexts are no longer needed.

4 Contributions

This section summarizes the contributions made by each of the four papers in the Thesis.

4.1 Paper I

This paper presents an efficient main-memory ordered indexing framework for sliding windows over data streams. There are three requirements for indexing sliding windows. First, the search needs to be responsive, second high insertion rates need to be handled to support high stream rates, and third, piecewise bulk deletion need to be supported for responsive deletion of expired data as the window slides. A number of main-memory ordered indexes were investigated where the highly optimized cache-aware compact trie implementation Judy [44] was particularly interesting for responsive indexing as it supports very fast insertion in constant time. However, Judy had a very slow range search. We developed a mapper function that applies a user defined aggregate function while traversing the Judy data structures. The mapper approach does not require source code modification of the very complex Judy implementation while significantly improving its range search. The need for piecewise bulk deletion in sliding windows was addressed by a framework for indexing sliding windows that allows bulk deletion for any plugged-in indexing structure. This framework was further improved in Paper IV by contextualized indexing.

I am the primary author of this paper. The second author contributed in discussions and in writing the paper.

4.2 Paper II

In this paper a solution to the DEBS2013 grand challenge [16] is presented. The problem is implementing four continuous aggregation queries over a stream of spatio-temporal sensor readings produced in a real soccer game. The particular challenge was to minimize the response time of the four continuous queries running in parallel on a single 4-core machine. We used the two-phase stream aggregation over sliding windows to improve the performance of standalone queries. Furthermore, to decrease overall resource con-

sumption, we needed to utilize shared execution for the queries when possible. The two-phase approach allowed for sharing partial aggregations between queries, reducing the overall resource consumption, and providing responsive execution of the four queries.

I am a co-author to this paper. I designed and implemented two of the queries, specifically the two-phase approach for sharing the partial aggregations. I also contributed in writing most of the manuscript.

4.3 Paper III

This paper outlines the overall research project. It includes an investigation of query processing over sliding windows, common multi-query optimization techniques for aggregate queries over sliding windows, and identifies the challenges of designing a framework for responsive data stream clustering. In particular, it identifies supporting non-decremental clustering algorithms as one of the main challenges for real-time data stream clustering over sliding windows, which is addressed in Paper IV.

I am the author of this paper.

4.4 Paper IV

This paper presents the G2CS framework in details. G2CS uses SBM and contextualized indexing for real-time data stream clustering over sliding windows. The design details of SBM and contextualized indexing is outlined in the paper. Furthermore, the paper includes a thorough computational complexity analysis for SBM and contextualized indexing, which is also verified by extensive performance experiments. The paper uses the clustering algorithm BIRCH as a running example to explain how clustering algorithms can be plugged-in, resulting in a variant of BIRCH for sliding windows called Continuous BIRCH, C-BIRCH.

I am the primary author of this paper, developed G2CS and C-BIRCH, and performed extensive performance experiments. The other authors contributed in discussions and writing the manuscript.

5 Conclusion and future work

G2CS is a framework that supports real-time data stream clustering over sliding windows by extending the two-phase approach [8] [9] [10] [11] [12] for stream summarization over sliding windows. It supports responsive detection of clusters over streaming data by a novel sliding mechanism for non-decremental clustering algorithms and multi-dimensional indexing. It uses SBM to avoid overlapping computations which is shown to perform more responsive compared to the previous repetitive merge approaches [40] [4] [5] [41]. To support scalable index insertion and deletion under high volume stream rates, G2CS uses contextualized indexes that separate the implementation of indexing mechanisms from both the applied clustering algorithms and the window maintenance mechanisms. This modularization structures and simplifies the implementation of clustering algorithms over sliding windows and allows for responsive bulk deletion of indexes as the windows slide.

There are a number of future research directions. First, parallelizing and distributing the query processing in G2CS can be investigated to further improve the response time and throughput of the query processing. Second, there are often opportunities for minimizing resource consumption by sharing computations when several continuous queries are submitted to the system having similar query components [9] [10] [11] [12] [40]. For example, queries might share window fragments and selection predicates. Third, more dynamic windowing semantics, like predicate-based and partition-based windowing [45], can be supported by SBM. Fourth, the Thesis assumes that stream elements arrive in order, whereas in some applications some elements might arrive with delays. Therefore, supporting out-of-order element arrival in SBM is desirable.

6 Summary in Swedish

Digitala data produceras numera i extremt höga hastigheter och volymer. Databaser som MySQL och sökmotorer som Google används ofta för att söka och analysera stora datamängder som lagrats på disk i datorer på internet. Emellertid produceras i många fall kontinuerligt strömmande data, t.ex. ljud, aktiedata, trafikdata och mätvärden från sensorer på maskiner. Eftersom strömmande data är obegränsade i storlek och produceras i realtid är det ofta inte möjligt att först lagra dem på disk innan man söker bland dem. I stället vill man söka och analysera data direkt i strömmen snarare än att först mellanlagra dem.

För att hantera strömmande data har speciella sökmotorer utvecklats som brukar kallas *dataströmhanteringssystem* (eng. DSMS, Data Stream Management Systems) till vilka man kan ställa frågor mot dataströmmar på liknande sätt som *databashanteringssystem* (eng. DBMS, Data Base Management Systems) som MySQL hanterar frågor mot lagrade data. En fråga mot strömmande data ger ett kontinuerligt svar i form av en ström, t.ex. genom att kontinuerligt detektera skadliga resonansfrekvenser från en eller flera sensorer som mäter vibrationer i en maskin. En sådan *stående fråga* filtrerar data kontinuerligt så länge den är aktiv medan konventionella databasfrågor utförs omedelbart och avslutas när resultatdata returnerats.

Ofta används stående frågor för att analysera dynamiska system vars uppförande kontinuerligt varierar över tiden, t.ex. strömmar av geo-positioner från fordon i ett stadsområde eller strömmar av uppmätta vibrationer hos en maskin. För att i tid upptäcka onormalt beteende hos det analyserade systemet, t.ex. trafikolyckor eller skadliga vibrationer, bör data från stående frågor produceras med så liten fördröjning som möjligt. Vidare måste dataströmhanteringssystemet vara snabbt nog att bearbeta data minst lika snabbt som de produceras, annars fördröjs systemet alltmer och klarar inte att analysera data med begränsad svarstid. Det vanligaste sättet att begränsa svarstiden i konventionella databaser är indexering, dvs. speciella datastrukturer på disk för att göra sökningar skalbara.

Dataströmhanteringssystem har stora krav på omedelbar bearbetning av mottagna data och därför måste alla data lagras och analyseras i primärminne, inklusive indexdatastrukturerna. Genom att göra all bearbetning i primärminne kan resultatet från stående frågor produceras med liten fördröjning.

Ett *fönster* mot en dataström är en begränsad senaste del av dataströmmen, t.ex. det sista 100 mätvärdena från en sensor eller alla mätvärden under den senaste millisekunden. Allteftersom strömmen fortskrider *glider* fönstret framåt över strömmen. Eftersom dataströmmar kan ha obegränsad längd måste algoritmer som kräver tillgång till en begränsad mängd data för sin analys appliceras på sådana fönster. Vidare behövs fönster för att snabbt upptäcka onormalt beteende hos strömmar från dynamiska system. Ofta samlas statistik från varje fönster m.h.a. frågor som grupperar data m.a.p. en kategori eller *gruppnyckel*, t.ex. uppmätt maximal och genomsnittligt tryck per sensor för ett antal trycksensorer på en maskin(.). Varje sensor har ett heltal som gruppnyckel och dataströmhanteringssystemet upprätthåller kontinuerligt en tabell av statistik per gruppnyckel allteftersom strömmen fortskrider och fönstret glider framåt.

I många fall kan emellertid ingen gruppnyckel identifieras för att kontinuerligt tabulera statistik, t.ex. när man vill identifiera grupper av maskiner med likartat beteende. I sådana fall används istället s.k. klustringsalgoritmer som KMEANs [1] och DBSCAN [2] för att forma grupperna och beräkna statistiken. Sådan klustring av strömmande data är särskilt utmanande eftersom det innebär att grupper dynamiskt skapas, slås samman och delar sig allteftersom strömmen fortskrider. Över dessa dynamiskt formade grupper applicerar algoritmerna därvid statistiska sammanfattningar i realtid [3] [4] [5].

Det mesta av tidigare forskning inom dynamisk klustring av strömmande data är inriktad på att utveckla monolitiska algoritmer där fönster- och indexeringsmekanismer ingår i algoritmerna, vilket resulterar i sammanflätade implementeringar där kod inte kan återanvändas. Till exempel i EXTRA-N [4] och SGS [5] ingår algoritm-specifika index över data i glidande fönster. BIRCH [6] är ett annat exempel på en klusteralgoritm som använder en egen indexeringsteknik, som kallas CF-träd. Att implementera strömmande klustringsalgoritmer från grunden kräver en mycket sällsynt kombination av kompetens. Därför finns behov av ramverk där analytiker kan uttrycka sina dataanalyser på en hög nivå, medan olika klustrings- och indexeringsalgoritmer kan utvecklas oberoende och pluggas in i ramverket [7].

Denna avhandling behandlar det allmänna problemet med sammanfattning av strömmande data i realtid. Följande frågeställningar undersöks:

1. Vilken är en lämplig generell mekanism för glidande fönster för olika sorters strömmande sammanfattningsalgoritmer?
2. Vad är en lämplig indexeringmekanism för datasammanfattning över glidande fönster?
3. Hur kan fönstrets mekanism för framåtskridande separeras från både indexering och den sammanfattningsalgoritm som appliceras för att undvika sammanflätade implementeringar?

Ansatsen är att utveckla ett generellt system G2CS *(Generic 2-layer Continuous Summarization)* där olika sammanfattningsalgoritmer och indexeringsstrukturer kan pluggas in oberoende av varandra. Olika implementeringsalternativ för G2CS har utvärderats.

För forskningsfråga ett analyserar vi i publikation II olika metoder att utföra frågor som grupperar strömmande data från en fotbollsmatch i realtid. En två-fas strategi [8] [9] [10] [11] [12] för strömmande summering över glidande fönster väljs för vidareutveckling. Baserat på denna fallstudie, utvärderas i publikation IV två tillvägagångssätt för att effektivt kunna stödja frågor över kluster med hjälp av en ny fönstermekanism som kallas *Sliding Binary Merge (SBM)*.

Publikation I adresserar forskningsfråga två genom att presentera en generisk ansats för indexering av data i glidande fönster där grupperade data kontinuerligt aggregeras genom att dela upp både data och index i ett glidande fönster. Detta tillvägagångssätt är generaliserat i publikation IV för att indexera de data som krävs för att upprätthålla dynamiska kluster över glidande fönster.

För forskningsfråga tre visas i publikation IV hur G2CS separerar sin mekanism för glidande fönster från de indexerings- och sammanfattningsalgoritmer som pluggats in. Detta förenklar implementeringen och gör det möjligt att plugga in olika klustringsalgoritmer. Således möjliggör G2CS återanvändning av mjukvarukomponenter och förenklar införandet av nya algoritmer.

7 Bibliography

[1] James MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob.*, 1967, pp. 281-297.

[2] Martin Ester et al., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Knowledge Discovery and Data Mining (KDD)*, 1996, pp. 226–231.

[3] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou, "Density-Based Clustering over an Evolving Data Stream with Noise," in *SDM*, 2006, pp. 328-339.

[4] Di Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data.," in *EDBT conf.*, Saint Petersburg, 2009, pp. 229-540.

[5] Di Yang, Elke A Rundensteiner, and Matthew O Ward, "Summarization and matching of density-based clusters in streaming environments," in *Proceedings of the VLDB Endowment*, 2011, pp. 121-132.

[6] Tian Zhang, Raghu Ramakrishnan, and Miron Livny, "BIRCH: an efficient data clustering method for very large databases," in *Proceedings of the 1996 ACM SIGMOD international conference on Management of data* , 1996, pp. 103-114.

[7] Volker Markl, "Breaking the Chains: On Declarative Data Analysis and Data Independence in the Big Data Era," in *International Conference on Very Large Data Bases (VLDB)*, Hangzhou, 2014, pp. 1730-1733.

[8] L. Jin, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker, "Semantics and evaluation techniques for window aggregates in data streams," in *SIGMOD conf.*, Baltimore, Maryland, 2005.

[9] Z. Rui, N. Koudas, B. C. Ooi, and D. Srivastava, "Multiple aggregations over data streams," in *SIGMOD conf.*, Baltimore, Maryland, 2005.

[10] Krishnamurthy S., C. Wu, and M. Franklin, "On-the-fly sharing for streamed aggregation," in *SIGMOD conf.*, Chicago, Illinois, 2006.

[11] G. Shenoda, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Optimized processing of multiple aggregate continuous queries," in *Proceedings of the 20th ACM international conference on Information and knowledge management*, Glasgow, 2011.

[12] G. Shenoda, M. A. Sharaf, P. K. Chrysanthis, and A. Labrinidis, "Three-level processing of multiple aggregate continuous queries," in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on*, Hannover, 2012.

[13] Cheng Xu et al., "Scalable Validation of Industrial Equipment using a Functional DSMS," *Journal of Intelligent Information Systems*, vol. 47, August 2016.

[14] Hillol Kargupta et al., "MobiMine: monitoring the stock market from a PDA," *ACM SIGKDD Explorations Newsletter*, vol. 3, no. 2, pp. 37-46, Jan 2002.

[15] Gyozo Gidofalvi, Torben Bach Pedersen, Tore Risch, and Erik Zeitler, "Highly scalable trip grouping for large-scale collective transportation systems," in *11th international conference on Extending database technology: Advances in database technology*, 2008, pp. 678-689.

[16] Z. Jerzak and H. Ziekow. (2013) DEBS Grand Challenge. [Online]. http://www.orgs.ttu.edu/debs2013/index.php?goto=cfchallengedetails

[17] (2016, Aug) Storm home page. [Online]. http://storm.apache.org/

[18] Mathew Sajee, "Overview of amazon web services," *Amazon Whitepapers*, Nov 2014.

[19] Tyler Akidau et al., "MillWheel: fault-tolerant stream processing at internet scale," in *Proceedings of the VLDB Endowment*, 2013, pp. 1033-1044.

[20] Paris Carbon et al., "Apache Flink™: Stream and Batch Processing in a Single Engine," *IEEE Data Engineering Bulletin*, 2015.

[21] Seyed Jalal Kazemitabar, Ugur Demiryurek, Mohamed Ali, Afsin Akdogan, and Cyrus Shahabi, "Geospatial stream query processing using Microsoft SQL Server StreamInsight," in *Proceedings of the VLDB Endowment* , 2010, pp. 1537-1540.

[22] (2016, Aug) StreamBase. [Online]. http://www.streambase.com/

[23] (Aug, 2016) SQLStream. [Online]. http://sqlstream.com/

[24] C. Cranor, T. Johnson, O. Spataschek, and V. Shkapenyuk, "Gigascope: a stream database for network applications," in *SIGMOD conf.*, New York, 2003, pp. 647-651.

[25] M. Tamer Ozsu Lukasz Golab, "Issues in Data Stream Management," *SIGMOD Record*, vol. 32, no. 2, pp. 5-14, June 2003.

[26] Arvind Arasu et al., "STREAM: The Stanford Data Stream," Stanford, 2004.

[27] Sirish Chandrasekaran et al., "TelegraphCQ: Continuous Dataflow Processing for an Uncertain World," in *SIGMOD*, 2003, pp. 668-668.

[28] E. Zeitler and T. Risch, "Massive scale-out of expensive continuous queries," in *VLDB conf.*, Seattle, 2011, pp. 1181-1188.

[29] S. Babu and J. Widom, "Continuous queries over data streams," *ACM SIGMOD Record*, vol. 30, no. 3, pp. 109-120, 2001.

[30] David Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*.: Springer, 2008.

[31] Teuvo Kohonen, "Self-organized formation of topologically correct feature maps," *Biological Cybernetics*, vol. 43, no. 1, pp. 59–69, 1982.

[32] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan, "Automatic subspace clustering of high dimensional data for data mining applications," in *SIGMOD '98 Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, 1998, pp. 94-105.

[33] Cheng Xu, "Scalable Validation of Data Streams," Uppsala University, PhD thesis ISSN 1651-6214, 2016.

[34] Ramez Elmasri and Shamkant Navathe, *Database systems*, 6th ed.: Pearson, 2011.

[35] Lukasz Golab, Shaveen Garg, and Tamer Özsu, "On Indexing Sliding Windows over Online Data Streams," in *International Conference on Extending Database Technology (EDBT)*, 2004, pp. 712-729.

[36] Carlo Zaniolo and Haixun Wang, "Logic-based user-defined aggregates for the next generation of database systems," in *The Logic Programming Paradigm*.: Springer Berlin Heidelberg, 1999.

[37] M. Ester, H-P. Kriegel, J. Sander, M. Wimmer, and X. Xu, "Incremental clustering for mining in a data warehousing environment," in *VLDB conf.*, New York, 1998, pp. 323-333.

[38] Kanat Tangwongsan, Martin Hirzel, Scott Schneider, and Kun-Lung Wu, "General incremental sliding-window aggregation," *Proceedings of the VLDB Endowment*, vol. 8, pp. 702--713, 2015.

[39] Sudipto Guha, Nina Mishra, and Rajeev Motwani, "Clustering data streams," in *Foundations of computer science, 2000. proceedings. 41st annual symposium on*, 2000, pp. 359--366.

[40] D. Yang, E. A. Rundensteiner, and M. O. Ward, "A shared execution strategy for multiple pattern mining requests over streaming data," in *VLDB conf.*, Lyon, 2009, pp. 874-885.

[41] B. Babcock, D. Mayur, M. Rajeev, and L. O'Callaghan, "Maintaining variance and k-medians over data stream windows," in *SIGMOD conf.*, San Diego, 2003, pp. 234-243.

[42] Thanh Truong and Tore Risch, "Transparent inclusion, utilization, and validation of main memory domain indexes," in *27th International Conference on Scientific and Statistical Database Management*, San Diego, 2015.

[43] Tore Risch, Vanja Josifovski, and Timour Katchaounov, "Functional Data Integration in a Distributed Mediator System," in *The Functional Approach to Data Management*. Berlin: Springer, 2004, pp. 211-238.

[44] D. Baskins, "Judy home page [http://judy.sourceforge.net/]," 2003. [Online]. http://judy.sourceforge.net/

[45] Xu Cheng, Daniel Wedlund, Martin Helgoson, and Tore Risch, "Model-based validation of streaming data: (industry article)," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*, 2013, pp. 107-114.

Acta Universitatis Upsaliensis

Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology 1431

Editor: The Dean of the Faculty of Science and Technology

A doctoral dissertation from the Faculty of Science and Technology, Uppsala University, is usually a summary of a number of papers. A few copies of the complete dissertation are kept at major Swedish research libraries, while the summary alone is distributed internationally through the series Digital Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology. (Prior to January, 2005, the series was published under the title "Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology".)

Distribution: publications.uu.se
urn:nbn:se:uu:diva-302799

ACTA
UNIVERSITATIS
UPSALIENSIS
UPPSALA
2016