Evaluation and Development Perspectives of Stream Data Processing Systems

Marcin Gorawski[1,2], Anna Gorawska[2], and Krzysztof Pasterak[2]

[1] Wroclaw University of Technology, Institute of Computer Science,
Wybrzeze Wyspianskiego 27, 50-370 Wroclaw, Poland
Marcin.Gorawski@pwr.wroc.pl
[2] Silesian University of Technology, Institute of Computer Science,
Akademicka 16, 44-100 Gliwice Poland
{Marcin.Gorawski,Anna.Gorawska,Krzysztof.Pasterak}@polsl.pl

Abstract. The following paper describes some common aspects of stream data processing systems. The paper consists of two main parts – first showing the short description, tests results and conclusions of an implemented system – the AGKPStream, while the second part focuses on proposed solutions, created upon experiences gained during development of mentioned system, as well as knowledge collected during learning about some concepts of a StreamAPAS system. The first discussed issue is a tuple construction – basic data representation. It concerns tuple time model, tuple schema and a tuple decorator. Afterwards, the stream query and scheduling problems are described.

Keywords: stream data processing, tuple, tuple time model, tuple schema, joined tuples decorator, stream query, stream schedulers.

1 Introduction

Designing a system for events monitoring, which is efficient and scalable is a major interest in recent studies. Processing multiple data connected with critical events is a crucial issue. The increasing number of data generated by on-line sources created unpredictable data streams processing a challenging problem that cannot be solved using traditional data bases. Therefore it is highly probable that the paradigm of data stream processing will became an important part of managing such data volumes.

The Data Stream Processing Management System (DSMS) assumes that data sources, called data streams, produce data continuously in an unpredictable matter. Data streams are considered to be open-ended and theoretically unbounded in size. The system does not have any control over data volumes arrival order and structure (schema), since DSMS is usually connected to remote sources and sinks. Moreover, once a tuple is taken from a data stream it is processed and then archived or discarded. Unless there is a data warehouse [1–10] storing historical data it is almost impossible to retrieve the processed tuple. Data stream processing systems are considered to support rapidly changing data sources.

A. Kwiecień, P. Gaj, and P. Stera (Eds.): CN 2013, CCIS 370, pp. 300–311, 2013.

Detailed description of the data stream processing paradigm can be found in [11, 12, 3, 13–17].

Main aspects of the stream data system described in this paper concern tuple time model, tuple schema and tuple decorator. Next discussed issues are stream queries and scheduling policies.

First part of the paper shows main ideas of a prototype Data Stream Management System called AGKPStream. The goal of the research is to assure continuity of reading data from the stream sources. Minimization of delay between measuring and storing the data is main criteria of evaluation. If it is possible, we want to assure successful recovery of the interrupted processing without losing any stream data. Next part is dedicated to presentation of new ideas, which are planned to be implemented in a forthcoming system [18–24].

2 Implemented Aspects of Stream Data Processing Systems

The AGKPStream system is based on the data stream processing paradigm. Therefore, all fundamental ideas and conditions consistent with this model had to be met. This part focuses on several interesting aspects of the AGKPStream system, which results from applying mentioned paradigm. Similarly to previously described system, a StreamAPAS system allows user to define temporal data analysis as well as positive/negative tuples processing. Some of further mentioned aspects are also described in [25, 26].

During all tests presented in Sect. 2, three main metrics were measured. First, a tuple response time, represents the time interval between tuple departure and arrival time. Next metric is a tuple slowdown [11], which is a ratio between tuple response time and ideal tuple response time (i.e. the time tuple would be processed by the system without waiting in queues). This metric illustrates how big delay each tuple is experiencing during processing. Algorithm used to measure ideal processing time can be found in [20]. The last tested metric was memory usage calculated by measuring both the number of tuples present in the system and total system memory consumption.

2.1 Tuple Time Model

The AGKPStream system adopts a temporal tuple time model. Each tuple contains, apart from data (attributes), two timestamps – the beginning and the end of tuples life time, determining the slice of time, where event described by tuple is valid. There were some operators, which had to store their input tuples in internal data structures. Those structures were cleaned each time new tuple was taken by those operators. During such cleaning operation the remove condition was based on the tuples life time – tuples with end timestamp value less than current system time were deleted.

The temporal tuple time model was satisfying for the AGKPStream system applications. It is worth noticing that the simplicity of the selected model distinguishes it among other time models. The biggest asset to the temporal time

model is that, according to [25, 26], it reduces the amount of transmitted data doubly.

2.2 Tuple Schema

In the AGKPStream system, a tuple schema was assigned to the stream. Each tuple contained data, but proper interpretation of them could be obtained only when the tuple was stored or read from the particular stream. As a consequence, an isolated tuple could not be used (or even its existence was indefinite) without a stream. The only way to properly extract the desired attributes value was to address tuples attribute using name taken from the schema.

Tuple schemas for all streams were calculated before appropriate data processing. When the first tuple arrived to the system, all streams already knew their schemas. Due to such strategy, all query defining and system configuration errors are detected before the data processing start.

Other advantage is the construction of a projection operator. It was intended to trim tuple schemas. Since all schema calculations were performed before data processing, i.e. in the system's initialization phase, projection operator simply set trimmed input schema to its output stream before the first tuple arrived to the system. After that, its work was equal to sending input tuples to the output. They contain untrimmed data indeed, although the view of that data was trimmed by the schema located in the stream. Besides all these advantages, schema-in-stream strategy had also some disadvantages, such as complications of union operator (detailed description can be found at [11]).

2.3 Tuple Decorator

A joined tuples decorating mechanism was implemented in the AGKPStream system and was used as a part of join operators algorithm. The idea was to substitute two input tuples with a single output tuple of mentioned operators without any data copying. In the case of multiple subsequent joins, the final structure of the output tuple was formed by a number of decorators, pointing each other.

The attribute mapping in a tuples decorator was based on a simple strategy of finding first matching pattern. The method is current until the condition of unique attribute names in each decorated tuple is preserved, however, this assumption cannot be always assured. Therefore, the AGKPStream system had been provided with an attributes name prefix mechanism, which was based on the origin of the tuple (i.e. a tuples source). That strategy allowed to ensure that each attribute of the two joined tuples would be unique (unless join operation was performed over two copies the same tuple).

The aim of performed tests of join operators, was to verify the thesis which assumes that using a joined tuples decorator instead of a simple data copying approach to newly created output tuple is better in the issue of memory usage, due to limitation of total tuple number. Furthermore, the secondary aim was to show the differences between other metrics (such as tuple response time or

tuple slowdown). This chart (Fig. 1) clearly shows that using the joined tuples decorating mechanism visibly reduces total memory usage of tested cross-join query. In addition, the amplitude of changes is significantly less in contrast to the copying and creating new tuple strategy. Therefore it has positive influence on the stability of the system.

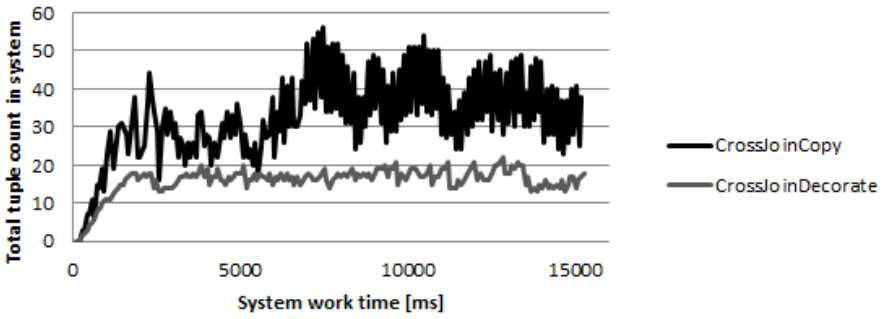

Fig. 1. Joined tuples strategies: total tuple count in system vs. system work time [ms]

The following charts (Fig. 2 and 3) reflect the results of comparison between those two joining strategies with respect to measured tuple response time and tuple slowdown.

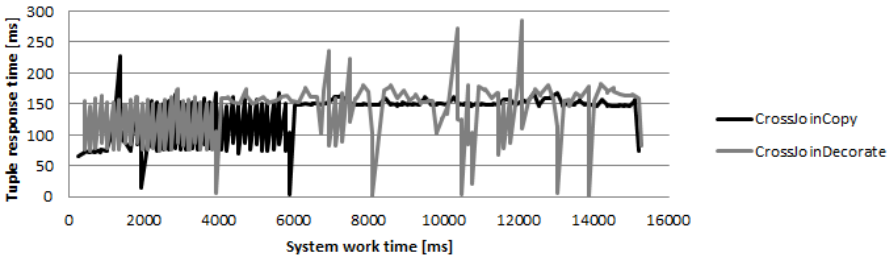

Fig. 2. Joined tuples strategies: tuple response time [ms] vs. system work time [ms]

Tuple slowdowns in both strategies are comparable, in favor of decorating strategy. However, the amplitude changes more visibly than in the copying strategy, which is quite balanced after the initial peak. Thus, it is hard to point a better strategy with respect to tuples slowdown optimization.

In all considered and tested implementations of join operator (i.e. cross-join, equi-join and theta-join), the tuple decorating strategy consumes less memory than corresponding copying technique. Therefore, in case of memory usage optimization, it is suggested to use joined tuples decoration strategy.

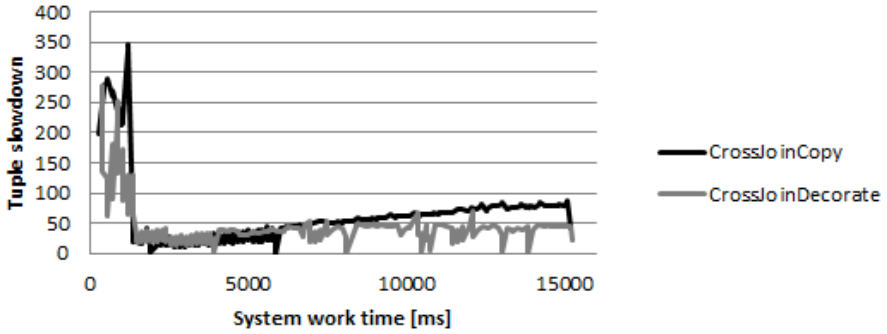

Fig. 3. Joined tuples strategies: tuple slowdown vs. system work time [ms]

Test results of two other metrics have not shown which of these compared strategies were better. In each of tested join operator implementations, tuple response time and tuple slowdown have had different dependences.

2.4 Queries

The AGKPStream system allows to define and create stream queries only before system's start. It was done during manual editing of multiple configuration files.

Each stream query is processing tuples in separation of the other queries, both with respect to scheduling algorithms and operator sharing. Furthermore, each stream query can have multiple inputs and outputs. As the result of such structure, defining query is more like defining whole query plan, where individual queries interlace with each other due to operator and stream sharing. In the AGKPStream system, each stream query has a local scheduler, which controls work of operators contained in this query. There is also a global scheduler, which controls all queries in the system.

This solution had many disadvantages. For example, when two similar queries (i.e. queries that had a nonempty set of identical operators) were registered in the system, the redundancy of operators and streams occurred (in the whole query plan, there was more than one element, which had been performing the same operations). However, there was a possibility of manual optimization of such query plan, by creating a double-output query, where data collected from each input would be identical as in two regular queries. Such joined query should not contain any redundant operators nor streams, although the end user skills need to be involved in an optimization process.

2.5 Schedulers

There were six schedulers implemented in the AGKPStream system. They could be divided into categories: simple schedulers (FIFO, Round Robin) [27, 28] and priority schedulers (Greedy, HR, HNR, MTIQ) [11, 27, 28]. Detailed information

on their algorithms was described in [29]. We distinguish two levels of scheduling policies: contained operator-level and query-level scheduling. Each query had a local scheduler, while all queries were controlled by a global scheduler.

The aim of performed tests was to choose the best priority scheduler of all four tested strategies (Greedy, HR, HNR, MTIQ). All of them were based on Round Robin scheduling policy. Three metrics were measured: tuple response time, tuple slowdown and memory usage (showed as total tuple count and total memory usage). Undoubtedly, the best priority scheduler was MTIQ according to Fig. 4–7. It is probably caused by simplicity of priority counting algorithm. The three remaining priority schedulers (Greedy, HR, HNR) were determining operator priority by performing complex calculations over selected operator statistics (such as selectivity and cost), which were often derived by expensive recursive floating point operations. The MTIQ scheduling strategy used stream tuple count. This statistic's value is simply actual buffer size.

3 Proposed Solutions

In the previous part, several aspects of the data stream processing AGKPStream system was presented. The analysis presented in Sect. 2 led to conclusions and ideas presented in following paragraphs.

3.1 Tuple Time Model

Knowledge of tuples life time was required in the AGKPStream system all of the time. When a new tuple arrived to the system, its end timestamp had to be known. The impossibility of changing that value and also lacks of such information in real life causes some difficulties for a certain area of applications. E.g. real-time traffic monitoring systems, where the character of events described by tuples is not determined (especially in terms of tuple life time). The need for creating more universal system led to development of different, more flexible tuple time model.

In the following system it is proposed to implement a punctuation-negative tuple time model, which assumes an existence of two kinds of tuples: regular data tuples and special punctuation tuples. The first one contain data, determined by proper schema. The punctuation tuples inform the system about groups of data tuples, which are outdated and have to be deleted from system. This model combines the features of punctuation model with the negative tuples model (signed tuples model).

In such model, operators would perform their data structure cleaning only when a punctuation tuple arrives (in the previous system, cleaning was done before processing each tuple). Moreover, there is no need for knowing the moment of tuple deactivation, i.e. tuple end timestamp is set when a new tuple is created, because it is determined by creation of the other – punctuation tuple. In graphical representation, such tuples (data and punctuation) can be described as points. In temporal model, tuples could be represented as lines.

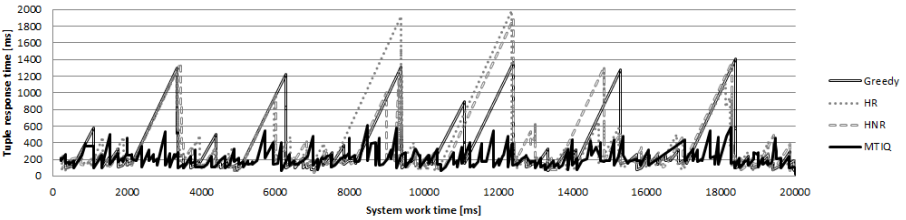

Fig. 4. Priority schedulers: tuple response time [ms] vs. system work time [ms]

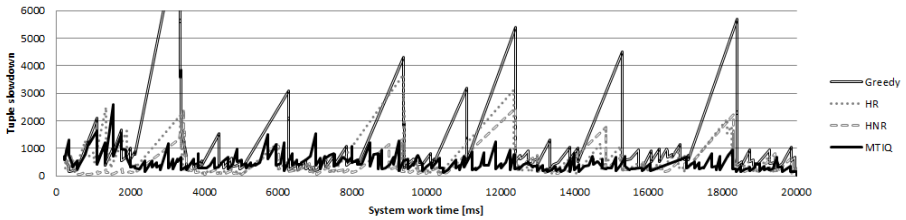

Fig. 5. Priority schedulers: tuple slowdown vs. system work time [ms]

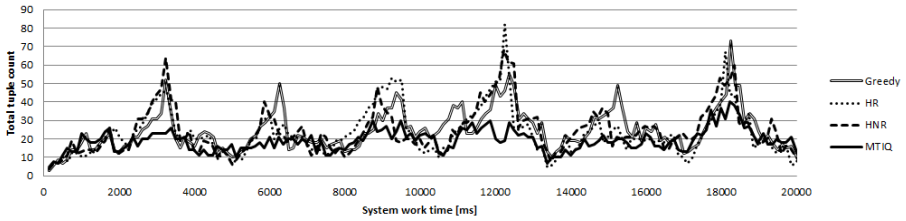

Fig. 6. Priority schedulers: total tuple count in system vs. system work time [ms]

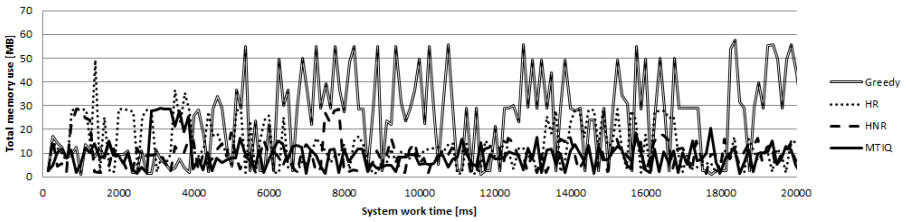

Fig. 7. Priority schedulers: total memory usage [MB] vs. system work time [ms]

3.2 Tuple Schema

In the previous section, a tuple schema realization in the AGKPStream system was described. Even though it had advantages mentioned in Sect. 2.2, such approach complicates the system. Checking validity of each stream and operator (with respect to tuple schema) with such precision led to unexpected difficulties.

The proposed solution reduces checking mechanisms for operators and streams in favor of increasing their fault tolerance. In consequence each operator should adjust to input data (with reasonable limits), e.g. when it needs to extract nonexistent attribute, it is better not to throw exception and stop the whole system, but to attempt to process that tuple as precisely as possible.

3.3 Tuple Decorator

Basing on experiences gained during creating and testing the AGKPStream system, joined tuples decorating strategy seems to be the most promising approach. Therefore, it is planned to implement this mechanism in the forthcoming stream data processing system.

In order to overcome problems connected with the proper tuple attributes addressing, it is proposed to extend each tuple with a special field, determining its source, called tuple origin (in the AGKPStream system, each attribute had special prefix). Moreover, tuple decorator ought to map tuple origin values to the decorated tuple pointers. Thus, when decorator is given a full attribute name (i.e. attribute name preceded by origin value), requested attribute value will be extracted from origin defined tuple, which is also solution for ambiguous attribute name in joined tuples problem. Furthermore, an origin based tuple mapping mechanism is flexible for multiple tuples being decorated, instead of only two tuples limit in the AGKPStream system.

3.4 Queries

In the forthcoming system, it is planned to introduce a different stream query model. First of all, proposed model ought to handle dynamic query registration and removal. Each stream query should have one output – representing user desired data. As input data for queries, it is assumed that each system's source (input stream, which is connected to the system) as well as actual running query results could be used. In terms of scheduling, it is planned to apply one global scheduler to all operators working in the system.

To solve operator and stream redundancy problem, a dynamic query merging and a splitting mechanism is intended to be used. The aim of such operation is to provide a query plan without redundant elements, but containing shared operators and streams [30]. In the AGKPStream system, similar result could be obtained only manually, but in future system it ought to be a regular procedure.

Two basic query operations should be defined. The result of merging two stream queries is a query, where all operators and streams are unique. It means that for each pair of identical elements in these two merged queries, the final

query will contain only one of them. It is also important to define when two elements (operators, streams) are identical. In the AGKPStream system, operators were distinguished by name. There was no such mechanism for streams. The forthcoming system will assume that two operators are identical while performing the same operations on the same data streams. Therefore two streams are identical when they are connecting identical operators.

On the other hand, when a stream query is no longer needed it should be removed from the query plan. The result of splitting one query with a query pattern are two separate queries. When a primary query contains some shared elements, they ought to be split into two separate copies of the same element, each in one of result queries. When removing an unnecessary query from a query plan, a primary query becomes query plan, while removed query is a query splitting pattern. One of the result queries is a new structure of a query plan, while the other is equal to the splitting pattern and is no longer used.

Currently developed system will implement query merging and splitting mechanism using a special stream attribute called sharing factor. It is simply the number of queries sharing that stream. Each query merging operation increments this value (only in affected elements) and each split operation decrements it. When sharing factor is equal to 0 it means that no queries share this stream and it could be removed from the query, also with operator connected to its output. When new element is inserted to the query, its sharing factor is equal to 1.

3.5 Schedulers

In the future, it is planned to limit the number of complex priority scheduler statistics and instead of them use simpler one. Most of all it is the stream size and optionally tuple wait time (time which each tuple has to wait to be processed) that will be used. These values do not require any calculations in order to obtain expected data. It is planned to test many different priority ratios, which can be obtained from these statistics.

In the AGKPStream system, each stream query had its own local scheduler. In the forthcoming system that strategy cannot be applied because of sharing query elements – it is impossible to determine clear borders between particular queries. Thus, it is proposed to apply only one global scheduler to all operators contained in query plan (which was mentioned in the previous part).

Different scheduling policies are considered, but each is based on simple statistics. The general rule is to call operator with the highest priority. Quite similar strategy is to creating a group of operators which are called periodically (like in Round Robin strategy). Operators which belong to such group are chosen from all operators and the choice is based on priority. It is also possible to split all working operators to many groups and to call all of them, but the frequency of calls depends on group priority, which depends on included operators priority.

4 Conclusions

In this paper we introduce interesting aspects of data stream processing that are connected to the prototype AGKPStream system or the StreamAPAS system [25]. Described ideas are the result of tests and conclusions drawn from mentioned systems. Although in the AGKPStream temporal model was satisfying, in the forthcoming work we consider punctuation-negative tuple time model as more suitable and flexible. Described measurements of joined tuples decorating mechanism performance in comparison to the results of copying and creating strategy shows that using tuple decorator has positive influence on system's efficiency. Disadvantage of the AGKPStream system was that queries were allowed to be defined only in system's initialization phase. A new system ought to handle dynamic query registration and removal. Mentioned features will make continuous system more flexible and efficient. With this motivation in mind it is likely that the forthcoming system will be more advanced than the previous one, e.g. by applying the distributed model [18, 19, 31, 21, 22] instead of a single application. In further future, it is also planned to include presented ideas in development of the stream data warehouse [1, 7, 22, 8, 23, 24]. Applying data stream processing mechanism is also being considered in addition to problems which descriptions can be found in [32–36, 29, 37–39].

References

1. Abhirup, C., Ajit, S.: A Partition-based Approach to Support Streaming Updates over Persistent Data in an Active Data Warehouse. In: Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing, IPDPS 2009, pp. 1–11. IEEE Computer Society, Washington, DC (2009)
2. Gorawski, M.: Extended Cascaded Star Schema and ECOLAP Operations for Spatial Data Warehouse. In: Corchado, E., Yin, H. (eds.) IDEAL 2009. LNCS, vol. 5788, pp. 251–259. Springer, Heidelberg (2009)
3. Gorawski, M.: Time complexity of page filling algorithms in Materialized Aggregate List (MAL) and MAL/TRIGG materialization cost. Control and Cybernetics 38(1), 153–172 (2009)
4. Gorawski, M., Gorawski, M.: Balanced spatio-temporal data warehouse with RMVB, STCAT and BITMAP indexes. In: PARELEC 2006: International Symposium On Parallel Computing In Electrical Engineering, pp. 43–48 (2006)
5. Gorawski, M., Malczok, R.: Indexing Spatial Objects in Stream Data Warehouse. In: Nguyen, N.T., Katarzyniak, R., Chen, S.-M. (eds.) Advances in Intelligent Information and Database Systems. SCI, vol. 283, pp. 53–65. Springer, Heidelberg (2010)
6. Gorawski, M., Marks, P.: Checkpoint-based resumption in data warehouses. In: Software Engineering Techniques: Design for Quality. IFIP, vol. 227, pp. 313–323. Springer, US (2006)
7. Gorawski, M., Marks, P.: Resumption of data extraction process in parallel data warehouses. In: Wyrzykowski, R., Dongarra, J., Meyer, N., Waśniewski, J. (eds.) PPAM 2005. LNCS, vol. 3911, pp. 478–485. Springer, Heidelberg (2006)
8. Gorawski, M., Morzy, T., Wrembel, R.: Special Issue on: Techniques of Advanced Data Processing and Analysis Introduction. Control and Cybernetics 38(1) (2009)

9. Kozielski, S., Wrembel, R. (eds.): New Trends in Data Warehousing and Data Analysis. Annals of Information Systems, vol. 3. Springer, US (2009)
10. Morzy, T.: Extraction, Transformation, and Loading Processes. In: Data Warehouses and Olap: Concepts, Architectures and Solutions, pp. 88–110 (2007)
11. Brian, B., Shivnath, B., Mayur, D., Rajeev, M., Dilys, T.: Operator scheduling in data stream systems. VLDB J. 13(4), 333–353 (2004)
12. Gorawski, M.: Advanced Data Warehouses. Habilitation, Studia Informatica 30(3B). Pub. House of Silesian Univ. of Technology (2009)
13. Gorawski, M., Chrószcz, A.: Synchronization Modeling in Stream Processing. In: Morzy, T., Härder, T., Wrembel, R. (eds.) Advances in Databases and Information Systems. AISC, vol. 186, pp. 91–102. Springer, Heidelberg (2013)
14. Gorawski, M., Malczok, R.: Towards stream data parallel processing in spatial aggregating index. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 209–218. Springer, Heidelberg (2008)
15. Gorawski, M., Malczok, R.: Answering Range-Aggregate Queries over Objects Generating Data Streams. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 436–439. Springer, Heidelberg (2010)
16. Gorawski, M., Marks, P.: Distributed stream processing analysis in high availability context. In: Proceedings of the Second International Conference on Availability, Reliability and Security, ARES, pp. 61–68 (2007)
17. Roger, S.B., Jonathan, G., Mohamed, H.A., Hong, M.: Consistent Streaming Through Time: A Vision for Event Stream Processing. In: Third Biennial Conference on Innovative Data Systems Research, CIDR 2007, Asilomar, CA, USA (2007)
18. Gorawski, M.: Architecture of Parallel Spatial Data Warehouse: Balancing Algorithm and Resumption of Data Extraction. In: Proceedings of the 2005 conference on Software Engineering: Evolution and Emerging Technologies, pp. 49–59. IOS Press, Amsterdam (2005)
19. Gorawski, M., Chroszcz, A.: Optimization of operator partitions in stream data warehouse. In: Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP, pp. 61–66. ACM, New York (2011)
20. Gorawski, M., Gorawski, M.: Modified R-MVB tree and BTV algorithm used in a distributed spatio-temporal data warehouse. In: Wyrzykowski, R., Dongarra, J., Karczewski, K., Wasniewski, J. (eds.) PPAM 2007. LNCS, vol. 4967, pp. 199–208. Springer, Heidelberg (2008)
21. Gorawski, M., Marks, P.: Towards reliability and fault-tolerance of distributed stream processing system. In: DEPCOS-RELCOMEX 2007 International Conference on Dependability of Computer Systems, pp. 246–253. IEEE Computer Society, Washington, DC (2007)
22. Gorawski, M., Marks, P., Gorawski, M.: Collecting data streams from a distributed radio-based measurement system. In: Haritsa, J.R., Kotagiri, R., Pudi, V. (eds.) DASFAA 2008. LNCS, vol. 4947, pp. 702–705. Springer, Heidelberg (2008)
23. Waas, F., Wrembel, R., Freudenreich, T., Theile, M., Koncilia, C., Furtado, P.: On-Demand ELT Architecture for Right-Time BI: Extending the Vision. International Journal on Data Warehousing and Mining (to appear, 2013)
24. Wrembel, R.: A Survey of Managing the Evolution of Data Warehouses. IJDWM 5(2), 24–56 (2009)
25. Gorawski, M., Chroszcz, A.: StreamAPAS: Query Language and Data Model. In: Proceedings of the Third International Conference of Complex, Intelligent and Software Intensive Systems, CISIS 2009, pp. 75–82. Springer, Heidelberg (2009)

26. Gorawski, M., Chrószcz, A.: Query Processing Using Negative and Temporal Tuples in Stream Query Engines. In: Szmuc, T., Szpyrka, M., Zendulka, J. (eds.) CEE-SET 2009. LNCS, vol. 7054, pp. 70–83. Springer, Heidelberg (2012)
27. Mohamed, A.S., Panos, K.C., Alexandros, L., Kirk, P.: Efficient scheduling of heterogeneous continuous queries. In: Proceedings of the 32nd International Conference on Very Large Data Bases, VLDB 2006, pp. 511–522. Endowment (2006)
28. Timothy, M.S., Bradford, P., Zhu, Y., Luping, D., Elke, A.R.: An Adaptive Multi-Objective Scheduling Selection Framework for Continuous Query Processing. In: Proceedings of the 9th International Database Engineering & Application Symposium, IDEAS 2005, pp. 445–454. IEEE Computer Society, Washington, DC (2005)
29. Jestratjew, A., Kwiecien, A.: Performance of HTTP Protocol in Networked Control Systems. IEEE Trans. Industrial Informatics 9(1), 271–276 (2013)
30. Patroumpas, K., Sellis, T.: Subsuming multiple sliding windows for shared stream computation. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 56–69. Springer, Heidelberg (2011)
31. Gorawski, M., Marks, P.: Fault-tolerant distributed stream processing system. In: International Workshop on Database and Expert Systems Applications – DEXA, pp. 395–399 (2006)
32. Gorawski, M., Malczok, R.: AEC Algorithm: A Heuristic Approach to Calculating Density-Based Clustering *Eps* Parameter. In: Yakhno, T., Neuhold, E.J. (eds.) ADVIS 2006. LNCS, vol. 4243, pp. 90–99. Springer, Heidelberg (2006)
33. Gorawski, M., Malczok, R.: Towards automatic *Eps* calculation in density-based clustering. In: Manolopoulos, Y., Pokorný, J., Sellis, T.K. (eds.) ADBIS 2006. LNCS, vol. 4152, pp. 313–328. Springer, Heidelberg (2006)
34. Gorawski, M., Marks, P.: Towards automated analysis of connections network in distributed stream processing system. In: Haritsa, J.R., Kotagiri, R., Pudi, V. (eds.) DASFAA 2008. LNCS, vol. 4947, pp. 670–677. Springer, Heidelberg (2008)
35. Gorawski, M., Lorek, M., Gorawska, A.: CUDA Powered User-Defined Types and Aggregates. In: International Workshop on Engineering Object-Oriented Parallel Software (IEEE AINA_EOOPS-2013). IEEE CS (to appear, 2013)
36. Jestratjew, A., Kwiecień, A.: Using Cloud Storage in Production Monitoring Systems. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2010. CCIS, vol. 79, pp. 226–235. Springer, Heidelberg (2010)
37. Kwiecień, A., Sidzina, M.: Dual Bus as a Method for Data Interchange Transaction Acceleration in Distributed Real Time Systems. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2009. CCIS, vol. 39, pp. 252–263. Springer, Heidelberg (2009)
38. Kwiecień, A., Opielka, K.: Industrial Networks in Explosive Atmospheres. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2011. CCIS, vol. 160, pp. 367–378. Springer, Heidelberg (2011)
39. Skrzewski, M.: Analyzing Outbound Network Traffic. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2011. CCIS, vol. 160, pp. 204–213. Springer, Heidelberg (2011)