PIPES – A Public Infrastructure for Processing and Exploring Streams

(Demonstration Proposal)

Jürgen Krämer
Department of Mathematics and Computer Science
Philipps University Marburg
kraemerj@mathematik.uni-marburg.de

Bernhard Seeger
Department of Mathematics and Computer Science
Philipps University Marburg
seeger@mathematik.uni-marburg.de

Abstract

PIPES is a flexible and extensible infrastructure offering fundamental building blocks that allow the construction of a fully functional prototype of a data stream management system (DSMS). It is seamlessly integrated into the Java library XXL [5, 7] for advanced query processing and extends its scope towards continuous data-driven query processing over autonomous data sources.

Our demonstration considers two realistic scenarios as example applications, namely traffic management and online auctions, whose relevance results from the fact that these represent a foundation for the development of benchmarks in stream processing. Subsequently, we will demonstrate how PIPES can be employed to these application domains. We will illustrate the construction and execution of query plans based on our generic operations. This is accomplished by a visual style of programming. Furthermore, we will point out our unique library approach by outlining its inherent publish-subscribe architecture and flexible metadata concept as well as our framework for operator scheduling and join processing.

Data Stream Processing

Autonomous data sources, e. g., sensors, satellites and click streams, produce potentially unbounded data streams with possibly unpredictable fluctuating rates. PIPES is a toolkit for developers establishing a basis for data stream management. In contrast to traditional database management systems (DBMS), the following issues have to be addressed:

- Typical queries are long-running and produce results in a continuous fashion. Due to limited system resources, nearly real-time requirements have to be considered as well as varying stream characteristics and approximate query answers.
- Stream elements have to be processed instantly at their arrival. An explicit materialization is mainly restricted to the case of historical queries. Therefore, concrete query plans consist of non-blocking data-driven operations. Since access to persistent data, such as relations, is still required in many applications, advanced mechanisms combining streams and relations are of particular importance in query processing.

A Brief Summary of PIPES

Contrary to existing work, we do not intend to build just another monolithic DSMS. Instead, we believe devoutly that it is nearly impractical to develop a general, performant and flexible DSMS coping with all issues arising in data stream management. A library approach, such as XXL and its extension PIPES, seems to be more adequate since it provides various fundamental and transparent building blocks. Based on these exchangeable components, we are able to create a fully functional prototype of a DSMS ensuring flexibility as well as extensibility.
Figure 1 illustrates how PIPES can be employed. We assume the dataflow being upward. At the bottom are heterogeneous data sources, e. g., sensors or relations, that are processed by operators belonging to query plans shown above. Different kinds of sinks, such as applications, PDAs or terminal users, appear at the top and continuously consume the results. The components at the right, namely the scheduler, memory manager, and query optimizer, represent the runtime environment and can be used either individually or in combination. These consequently enable a developer to create prototypical instances of a DSMS with various characteristics.

Query Plans

The infrastructure PIPES contains a separate framework that allows to construct directed acyclic query graphs. This is realized with the help of our inherent publish-subscribe architecture facilitating direct interoperability [6].
We differ between three different types of nodes within a query graph as illustrated by Figure 1:

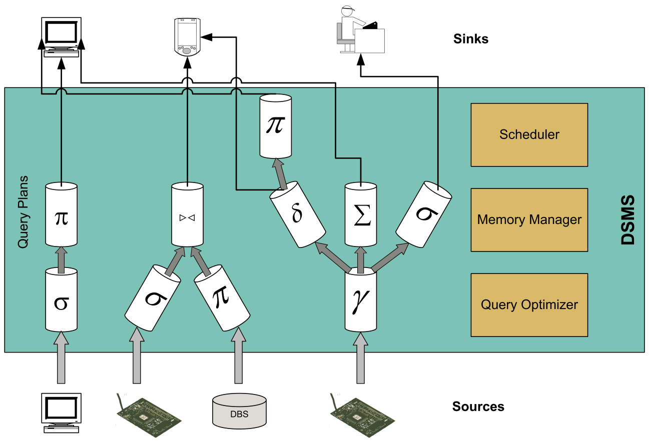


Figure 1: An architectural overview

1. A *source* transfers its elements to a set of subscribed sinks.
2. A *sink* can subscribe and unsubscribe, respectively, to multiple sources. It consumes all incoming elements from its sources as long as its subscription holds.
3. An *operator*, respectively *pipe*, combines the functionality of a sink and a source since it consumes an incoming element, processes it, and transfers its results to its subscribed sinks.

This segmentation of functionality is directly reflected by the implementation of PIPES. In addition to interfaces for each node type and abstract pre-implementations, a rich collection of ready-to-use components facilitates the implementation work of developers.
Moreover, our inherent publish-subscribe architecture allows to connect operators directly. This novel approach implies that no inter-operator queues are required for communication and, therefore, leads to a substantial overhead reduction.

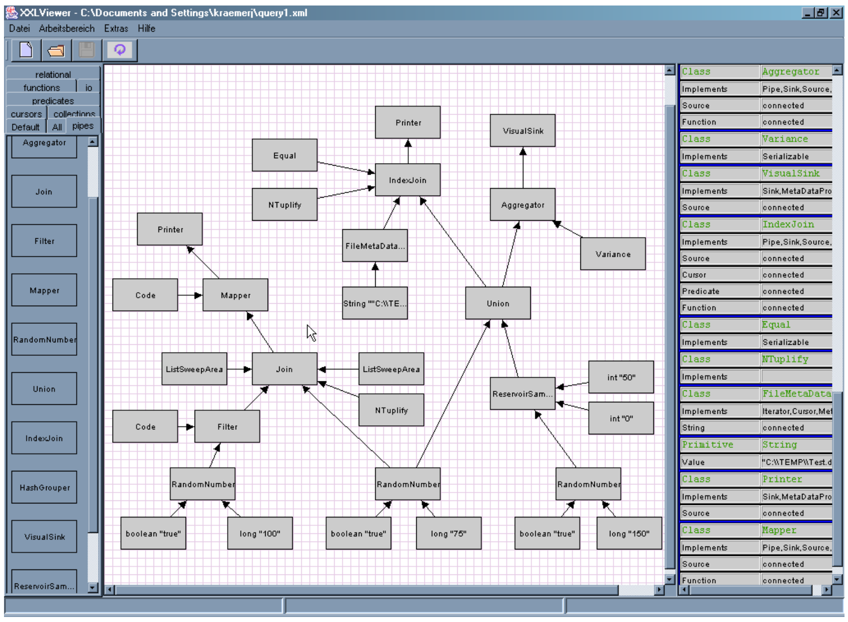
Operator algebra

PIPES provides a powerful and generic operator algebra for processing data streams including all operations known from the extended relational algebra. However, our algebra abstracts from relational schemas in so far that it handles arbitrary objects. We have already defined a precise operator semantics over time intervals for the data-driven realization of our operations [13]. Each operation is implemented in a non-blocking way leveraging sliding windows if necessary.
Our temporal operator algebra is absolutely conform to the Continuous Query Language (CQL) proposed by [2], but includes special mechanisms that substantially reduce stream rates. Comprisingly, we are able to build a physical query plan that produces snapshot-equivalent results in accordance to its logical counterpart derived from a CQL query.

Benefit from XXL and its library approach

Software design considerations play an important role in our library XXL [5], where PIPES is an integral part of. During our development, we have kept the following main goals in mind:

- *Applicability* Our library represents a toolkit consisting of easy-to-use and intuitively applicable building blocks. Single components should be usable without a complete knowledge of the library. Default implementations and simple use-cases help to create new query processing algorithms or even complete applications rapidly.
- *Reusability* Our generic and abstract approaches combined with design patterns cause the building blocks of our library being highly reusable due to better understandability. PIPES significantly benefits from XXL and its plenty of ready-to-use data structures, such as queues, heaps etc. As we will see in our demonstration, PIPES uses XXL's cursor algebra in many cases and thus gracefully combines data-driven and demand-driven query processing. Furthermore, we intend to leverage the index structures framework of XXL with its bulk operations to enable historical queries over streams.
- *Connectivity* In recent years, the application range of XXL has been widened continuously. XXL provides powerful mechanisms to connect its operators to databases, raw disks, files or even remote data sources. At present, some members of our team are working on efficient techniques for indexing and storing XML data and extending XXL's scope towards distributed query processing and web service integration. Note that all these features additionally widen the scope of PIPES.
- *Comparability* XXL represents an ideal testbed for algorithmic comparisons because its library design encourages exchangeable components. As a result of its research-oriented nature, a lot of algorithms have been implemented in XXL and are ready to be used for experimental evaluation. In the context of stream processing, this aspect becomes very important since to date no comparable approach exists. We will highlight this particular feature in our demonstration considering PIPES' scheduling and join frameworks as example.



Class	Aggregator
Implements	Pipe,Sink,Source,
Source	connected
Function	connected
Class	Variance
Implements	Serializable
Class	VisualSink
Implements	Sink,MetaDataPro
Source	connected
Class	IndexJoin
Implements	Pipe,Sink,Source,
Source	connected
Cursor	connected
Predicate	connected
Function	connected
Class	Equal
Implements	Serializable
Class	NTuplify
Implements	
Class	FileMetaData
Implements	Iterator,Cursor,Met
String	connected
Primitive	String
Value	"C:\\TEMP\\Test.d
Class	Printer
Implements	Sink,MetaDataPro
Source	connected
Class	Mapper
Implements	Pipe,Sink,Source,
Source	connected
Function	connected

Figure 2: Development GUI for query plans

Runtime Environment

In order to build a prototypical DSMS, PIPES additionally includes the following frameworks for runtime components that interact with nodes in a query graph.
During runtime, each node collects secondary metadata, a kind of synopses, represented by iteratively computed inferential estimators similar to online aggregation [11], for instance, stream rate variance or selectivity. Our runtime components are consequently parameterized by a strategy that typically incorporates secondary metadata information.

Scheduler

The scheduling framework [6] integrated in PIPES provides a 3-layer architecture. In the first layer, multiple succeeding nodes in a query graph can be merged to a more complex virtual node. This novel approach significantly improves performance since no inter-operator queues are required for communication within such a virtual node. The second layer allows to schedule nodes within a single thread, whereas the third schedules the threads of the second layer. Our hybrid approach has the advantage of neither being restricted to a single thread for scheduling [14, 4, 9] nor assigning a separate thread per operator [3, 8, 15].

Memory Manager

PIPES offers a flexible framework for adaptive memory management. Operators requiring memory, for instance a join, are subscribed to a memory manager which globally assigns and redistributes available memory at runtime. Due to dynamic system load ensued by varying stream rates and query graph modifications, scalability possibly suffers from limited memory. To avoid this drawback, our framework even takes approximate query answers into account. Thus, each time an operator reaches its memory limit, its current memory usage is reduced by applying a user-defined load-shedding strategy similar to [8].

Query Optimizer

Our infrastructure additionally provides a rule-based query optimizer extending multi-query optimization proposed by [16] towards stream processing. Our optimizer takes a new query as input and heuristically produces a set of snapshot-equivalent query plans as output. Each query plan is probed against the currently running query graph and, according to a cost function, the best matching plan is determined. Then, the accessory nodes are integrated into the running query graph via our publish-subscribe architecture. Currently, we use our query optimizer solely for static optimization but we intend to widen its applicability to the dynamic case.

Demonstration Details

In our demonstration, we will motivate three important features of PIPES: its broad applicability, its easy and flexible connectivity, and its algorithmic testbed character.

Application Domains

We will show how PIPES can be utilized to execute continuous queries in two completely different application domains. Both scenarios are plausible since benchmarks are presently developed referring to these applications.

Traffic management

Our first application scenario is based on extensive real data streams consisting of loop detector data collected by the Freeway Service Patrol Project (FSP) in 1993. At this, the traffic flow was measured twice along a ten-mile section on highway I-880, near Hayward, California. Each measurement lasted for approximately four weeks and collected data of 100 loop detectors at five lanes, including a separate lane for high occupancy vehicles (HOV). A stream element contains the following information: detector position, lane, timestamp at which a vehicle passed the sensor, velocity of

the vehicle and its length.
Similar to the Linear Road Benchmark [17], we will formulate and execute a few continuous queries exemplarily analyzing such traffic data streams. Typical queries might be: What has been the average speed of HOVs driving in direction Oakland within the last hour? At which sections of the highway is the average speed below a certain threshold constantly for 15 minutes? The latter might be an indicator for incidents and traffic congestion.

Online auctions

NEXMark [18] is an XML-based benchmark in progress designed to measure the effectiveness of DSMSs. A configurable stream generator produces synthetic XML data sets representing the necessary files and streams for an online auction scenario, where the following events occur in a continuously manner: new people are registered, items are submitted for auctions which are opened and closed, and bids arrive for items.
Since the NEXMark generator has been available just recently, we will show how PIPES, respectively XXL, can be utilized to gracefully combine data-driven and demand-driven query processing by joining streams and persistent data. Moreover, we will illustrate how to map some of the CQL queries presented in [18] to our physical operator algebra including even more complex queries, e. g., with a time-based, fixed window group-by clause such as, "Return every 10 minutes the highest bid in the recent 10 minutes.".

In order to make use of PIPES, a bidirectional connection between an application and our infrastructure has to be established. For that reason, a concrete application must provide at least the following:

- An adapter wrapping a raw input stream to a *source* within a query graph.
- A set of *continuous queries* specifying the operations to be performed on the input streams.
- Purpose-built *sinks* presenting, storing or transferring the streaming query results.

GUI for Query Plans

XXL provides an absolutely novel and generic visual style of programming (see Figure 2) based on the reflection mechanism built in Java. We will show how this tool allows a simple construction of complex query plans with respect to the above mentioned application scenarios. In addition to storing these query plans in XML files, the user has the option to generate and run the corresponding Java source code. Moreover, it is possible to enter Java code directly. We exposed this feature to developers since programming with XXL heavily profits by Java's concept of anonymous classes, e. g., to specify user-defined functions and predicates.
During our presentation, we will point out the inherent publish-subscribe architecture of PIPES as well as the generic design of our operator algebra parameterized by functions and predicates.

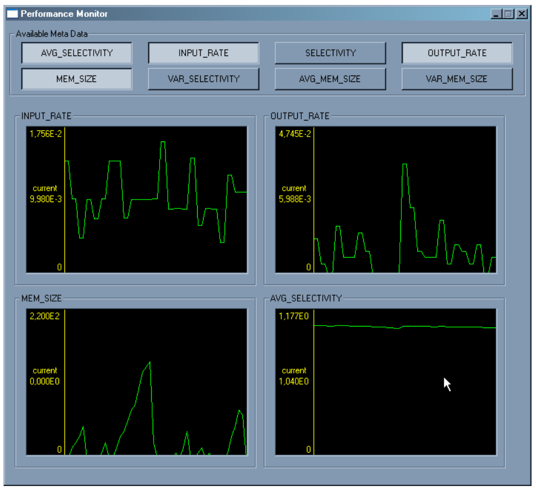


Figure 3: Monitoring secondary metadata

Performance Monitoring Tool

Since our runtime components incorporate secondary metadata such as runtime statistics to control and adapt system behavior, PIPES provides a configurable factory that decorates arbitrary nodes in a query graph with the desired metadata information. Hence, an administrator or developer gets the facility for specifying which metadata is required and updated during runtime. We support a widespread range of secondary metadata, for instance, current input and output rates, selectivity, number of subscribed sinks, actual memory size, or averages and variances of the previously listed items. Furthermore, it is possible to alter secondary metadata composition at runtime.
Our performance monitor (see Figure 3) is useful to visualize the available secondary metadata of a node in a running query graph. This tool will be part of our demonstration in so far that we will highlight the varying stream and operator characteristics arising in our applications scenarios, for instance, the effect of fluctuating stream rates on internal buffers.

Code Reusability

Subsequently to the demonstration of PIPES' applicability and connectivity, we will outline the advantage of code reusability resulting from our library approach.
Based on data flow translation operators [10], PIPES provides mechanisms to interact with cursors. We will point out how developers benefit from XXL's cursor algebra as well as its powerful index structures and IO capabilities. Another example for code reuse is given by our broad package of online aggregation functions which are designed independently from the underlying kind of processing, i. e., demand- or data-driven.

Algorithmic Testbed

We will finally demonstrate how our library approach suits for algorithmic comparisons.
First, we will shortly explain the sophisticated scheduling architecture integrated in PIPES that is powerful enough to compare most of the recent scheduling techniques in stream processing [4, 9, 8, 15, 14, 3, 6] within a uniform framework.

Second, we will give a brief overview of our generic and flexible join framework for streams based on a generalized ripple join technique [11]. A join is parameterized by exchangeable status-aware data structures, so called SweepAreas [7], providing efficient support for insertion, retrieval and reorganization. PIPES offers a variety of appropriately tailored SweepAreas coping different types of joins including [12, 19].

Conclusion and Acknowledgements

The main contribution of our work is to provide a sophisticated foundation for advanced query processing over data streams by developing powerful generic frameworks transparently integrated into a public available library. This approach turns out to be absolutely novel in the stream community since comparable projects chiefly focus on the development of a monolithic DSMS. Instead, we believe devoutly that it is nearly impractical to develop a general-purpose DSMS coping with all issues arising in data stream management.
XXL with its library approach has been established in the database community in recent years and we hope to successfully enlarge the number of its users as well as its application domains because such a library offers a lot of advantages compared to a monolithic system.

This project has been supported by the German Research Society (DFG), under grant no. SE 553/4-1. In addition, we want to thank all recent and former members of our database research group for their comprehensive contributions [1].

REFERENCES

[1] The XXL / PIPES Project. http://www.mathematik.uni-marburg.de/DBS/xxl/.

[2] A. Arasu, S. Babu, and J. Widom. An Abstract Semantics and Concrete Language for Continuous Queries over Streams and Relations. In *Proc. of the 9th Intl. Conf. on Data Base Programming Languages (DBPL)*, 2003.

[3] R. Avnur and J. M. Hellerstein. Eddies: Continuously Adaptive Query Processing. In *Proc. of the ACM SIGMOD*, pages 261–272. ACM Press, 2000.

[4] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain: Operator Scheduling for Memory Minimization in Data Stream Systems. In *Proc. of the ACM SIGMOD*, pages 253–264, 2003.

[5] J. Bercken, B. Blohsfeld, J.-P. Dittrich, J. Krämer, T. Schäfer, M. Schneider, and B. Seeger. XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In *Proc. of the Conf. on Very Large Databases (VLDB)*, pages 39–48, 2001.

[6] M. Cammert, C. Heinz, J. Krämer, A. Markowetz, and B. Seeger. PIPES - A Multi-Threaded Publish-Subscribe Architecture for Continuous Queries over Streaming Data Sources. Technical report, University of Marburg, 2003. CS-32.

[7] M. Cammert, C. Heinz, J. Krämer, M. Schneider, and B. Seeger. A Status Report on XXL - a Software Infrastructure for Efficient Query Processing. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 26(2):12–18, 2003.

[8] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Monitoring Streams: A New Class of Data Management Applications. In *Proc. of the Conf. on Very Large Databases (VLDB)*, pages 215–226, 2002.

[9] D. Carney, U. Cetintemel, S. Zdonik, A. Rasin, M. Cerniak, and M. Stonebraker. Operator Scheduling in a Data Stream Manager. In *Proc. of the Conf. on Very Large Databases (VLDB)*, pages 838–849, 2003.

[10] G. Graefe. Query Evaluation Techniques for Large Databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[11] P. J. Haas and J. M. Hellerstein. Ripple Joins for Online Aggregation. In *Proc. of the ACM SIGMOD*, pages 287–298. ACM Press, 1999.

[12] J. Kang, J. Naughton, and S. Viglas. Evaluating Window Joins over Unbounded Streams. In *Proc. of the IEEE Conference on Data Engineering*, 2003.

[13] J. Krämer and B. Seeger. Operations on Data Streams – Temporal Semantics and Data-driven Implementation Concepts. (ready for submission).

[14] S. Madden and M. J. Franklin. Fjording the Stream: An Architecture for Queries Over Streaming Sensor Data. In *Proc. of the IEEE Conference on Data Engineering*, 2002.

[15] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olsten, J. Rosenstein, and R. Varma. Query Processing, Resource Management, and Approximation in a Data Stream Management System. In *Proc. of the Conf. on Innovative Data Systems Research (CIDR)*, 2003.

[16] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and Extensible Algorithms for Multi Query Optimization. In *Proc. of the ACM SIGMOD*, pages 249–260, 2000.

[17] R. Tibbetts and M. Stonebraker. The Linear Road Benchmark. http://www.cs.brown.edu/research/aurora, 2003.

[18] P. Tucker, K. Tufte, V. Papadimos, and D. Maier. NEXMark – A Benchmark for Queries over Data Streams. http://www.cse.ogi.edu/dot/niagara/NEXMark/, 2003.

[19] S. D. Viglas, J. F. Naughton, and J. Burger. Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. In *Proc. of the Conf. on Very Large Databases (VLDB)*, pages 285–296, 2003.