Design Considerations of a Flexible Data Stream Processing Middleware

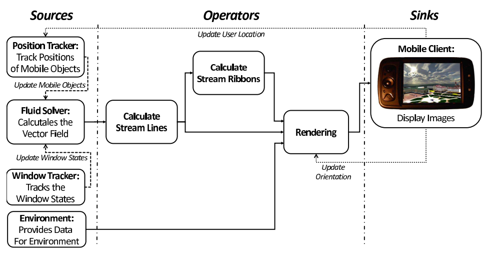
Nazario Cipriani[1], Matthias Grossmann[1], Harald Sanftmann[2], and Bernhard Mitschang[1]

[1] Universität Stuttgart, Institute of Parallel and Distributed Systems,
Universitässtraße 38, 70569 Stuttgart, Germany
{cipriani | grossmann | mitschang}@ipvs.uni-stuttgart.de
[2] Universität Stuttgart, Visualization Institute,
Universitätsstraße 38, 70569 Stuttgart, Germany
sanftmann@vis.uni-stuttgart.de

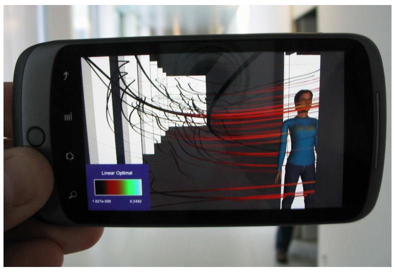
Abstract. Techniques for efficient and distributed processing of huge, unbound data streams have made some impact in the database community. Distributed data stream processing systems have emerged providing a distributed environment to process these potentially unbound streams of data by a set of processing nodes. A wide range of real-time applications process stream-based data. Sensors and data sources, such as position data of moving objects, continuously produce data that is consumed by, e.g., location-aware applications. Depending on the domain of interest, the processing of such data often depends on domain-specific functionality. For instance, an application which visualizes stream-based data has stringent timing constraints, or may even need a specific hardware environment to smoothly process the data. Furthermore, users may add additional constraints. E.g., for security reasons they may want to restrict the set of nodes that participates in processing.
In this paper we review context-aware applications which, despite their different application fields, share common data processing principles. We analyse these applications and extract common requirements which data stream processing systems must meet to support these applications. Finally, we show how such applications are implemented using NexusDS, our extensible stream processing middleware.

1 Introduction

Nowadays sensors exist producing a huge amount of data that is time-constrained and should thus be processed on-the-fly. This sensor data results in potentially unbound streams. Techniques for efficient and distributed processing of huge, unbound data streams have made some impact in the database community. In the past decade many studies have been conducted in the field of data stream processing systems. These studies ranged from architectural proposals to sophisticated processing techniques targeting the unbound nature of data streams. Nowadays distributed stream processing systems are state-of-the-art since they scale well with increasing workload and thus enable an efficient processing. These systems often provide a declarative query language and allow



(a) Visualization scenario realizing a flow visualization of airflows in buildings.



(b) Air flow visualized with stream ribbons in real-time.

Fig. 1. Real-time visualization and the respective visualization scenario.

to continuously process incoming data streams in a distributed fashion. The query is first mapped to operators and in a second step distributed to available computing nodes.

However, none of the systems reached a general acceptability for a huge number of applications and application domains, such as databases did. This is due to the fact that they do not consider the specific needs of real-time applications. Depending on the domain of interest (e.g., visualization) the processing of such data often depends on highly domain-specific functionality. For instance, an application which visualizes stream-based data has stringent timing constraints, or may need a specific hardware environment to smoothly process the data. Such a complex application scenario and the resulting visualization is depicted in Fig. 1(a) and Fig. 1(b) respectively. In this example, the air flow through a building is simulated and visualized as stream ribbons in real-time. Thereby the air flow adapts also to changing positions of objects moving in the building. The processing of such a scenery is a highly complex task and cannot be reasonably performed on mobile devices. Thus, dedicated hardware must be used. See [1] for a in-deep explanation of this example scenario.

Furthermore, users may add constraints. E.g., for security reasons they may want to restrict the set of nodes that participates in data processing. As explained in [2], there is a permanent adaptation necessity in today's stream processing systems. At a last consequence this means that each application domain has its dedicated processing schemes, although they rely on common processing techniques, i.e., data stream processing. Therefore, a common basis should be exploited to avoid redundant functionality and code as well as to reduce development time and errors due to usage of multiple different technologies.

In this paper we argue that many applications, although originating from different application domains, mostly share common processing principles. This calls for a data stream processing concept, that allows to express the particular characteristics and requirements of each application under concern. Hence, our data streaming approach, called NexusDS, has been designed to especially address the specifics of these domain-specific applications as well as the heterogeneous execution environment. This tight integration of applications and system considerably reduces development overhead and enhances infrastructure exploitation as well as overall performance.

The remainder of this paper is structured as follows: In Section 2 we present real world application scenarios from different application domains and extract their requirements in Section 3. In a next step we show in Section 4 why existing approaches do not suffice in supporting those applications. In Section 5 we introduce NexusDS, our stream processing middleware that supports the tight integration of such applications and provide an evaluation in Section 6. In Section 7 we conclude this paper with a short summary.

2 Application Scenarios

We will present three non-trivial applications: (A.) **Visualization application for mobile devices**, (B.) **storing moving objects' traces**, and (C.) **management support in smart factories**.

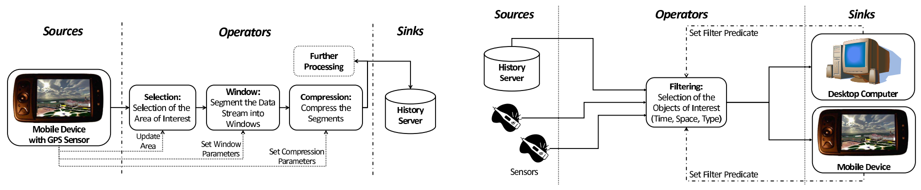
2.1 Visualization Application for Mobile Devices

A complex data stream scenario that goes beyond the current state of the art is an interactive and location-aware visualization application as depicted in Fig. 1(a) (the resulting visualization is shown in Fig. 1(b)). In this example, the air flow in a room is simulated and visualized. The *Environment* source provides room data. Objects moving in the room are tracked by the *Position Tracker* source whereas the status of the windows is tracked by the *Window Tracker* source. The *Fluid Solver* source simulates the velocity field which depends on the tracked objects. The *Calculate Stream Lines* operator seeds and calculates streamlines based on the velocity field. To visualize the twist induced by the velocity field, stream ribbons are calculated from the streamlines by the *Calculate Stream Ribbons* operator. The obtained geometry is rendered by the *Rendering* operator, which produces image output. This image output can then be displayed on a *Mobile Client* which do not have the capabilities to render complex sceneries themselves. Preferably, the rendering step is executed on specialized hardware having a graphics processing unit (GPU) which provides all the capabilities (GL extensions) required to deliver high quality renderings with good performance. User interaction, such as to rotate and pan the scene, can be modelled as parameter updates for the operators.

Thus, the system must support domain-specific operators, such as the Render operator and the corresponding constraints. E.g., for the effective execution of the domain-specific Render operator the presence of a GPU with certain capabilities is mandatory.

2.2 Storing Moving Objects' Traces

With the increasing use of sensor technology, the compression of sensor data streams is getting more and more important to reduce both the costs of further processing as well as the data volume for persistent storage. An example scenario is depicted in Fig. 2(a). A *Mobile Device with GPS Sensor* produces a stream of position updates, which is first processed by a *Selection* operator to reduce the stream to positions within a given area. The resulting stream is partitioned by the *Window* operator, which form the input for the


Sources
Operators
Sinks
Mobile Device with GPS Sensor
Selection: Selection of the Area of Interest
Window: Segment the Data Stream into Windows
Compression: Compress the Segments
Further Processing
History Server
Update Area
Set Window Parameters
Set Compression Parameters
Sources
Operators
Sinks
Set Filter Predicate
History Server
Filtering: Selection of the Objects of Interest (Time, Space, Type)
Desktop Computer
Mobile Device
Sensors
Set Filter Predicate

(a) Storing of moving objects' traces. (b) Management support in smart factories.

Fig. 2. Real-time visualization and the respective visualization scenario.

Compression operator. The compressed position information can be stored in a position *History Server* for later analysis or can be further processed by subsequent operators.

This allows to move the generic part of the functionality out of the compression operator and to implement it as an additional operator. This increases the performance, as the different operators can be deployed on different nodes and also enhances reusability of the generic parts [3].

2.3 Management Support in Smart Factories

A lot of influencing factors can cause disturbances in production processes in today's factories [4]. Unsteadiness of the demand for a product, changes of orders of a customer, delayed delivery of raw materials, failures of machines or decreasing quality of the products require quick adaptation of the production process. To perform such quick adaptation, it is necessary that the responsible persons, e.g., production managers or maintenance staff, can get information on the current state of the production facilities, failures or required actions at any time. The NexusExplorer is a tool which communicates with the back-end systems via an interface which is tailored to consume dynamic data. This interface constitutes a sink for data streams which may run on a *Desktop Computer* or *Mobile Device*. Selections based on time, type and location can easily be implemented as a *Filtering* operator in a data stream system, as shown in Figure 2(b). In such scenarios, the data stream system can also be used to propagate measured values from the *Sensors* to a *History Server*, making the data available for retrospective failure analysis.

Data required by the NexusExplorer is often a business secret of the company owning the factory. Thus it is important that applications such as the NexusExplorer can restrict the set of nodes for data processing to nodes owned or controlled by the company.

3 Requirements of Stream-based Applications

The presented sample applications illustrate that many different requirements exist at different levels of query processing as well as integration process. As an example, we refer to the *Render* operator which typically requires a GPU. Another example is the compression of moving object traces. Here, the integration of specific compression algorithms within the data stream processing pipeline is needed. An alternative case, where we do not need dedicated processing capabilities but an adaptation at infrastructure level, is given by the scenario *management support in smart factories* in Section 2.3. Here it is crucial, that sensitive factory data remains within certain predefined boundaries and is not propagated arbitrarily. All applications share the usage of streamed and static context data to adapt the actual processing according to their surroundings. The single steps of the applications map nicely to complex operators of a stream processing system, thus using such systems seems reasonable. In summary we have identified the following main requirements of these applications:

A. **Custom data-processing**: Applications often require functionality to tailor the system behaviour as well as actual data processing to their specific needs. An example is the *Rendering* operator of the visualization application from Section 2.1. In a large and distributed computing environment it is essential that new operators can be added to the stream processing system online, to make the operator available fast and easily.
B. **Structured and unstructured data support**: New application domains may introduce new types of data, such as images or video streams. This means that the system must allow to implement new operators that go beyond those provided by state-of-the-art data stream processing systems.
C. **Deployment and execution specifications**: Operators as well as applications may impose certain constraints to the operator deployment and execution. E.g., operators may only be deployed on specific hardware, may require a certain amount of memory at runtime, or may even be allowed to be exclusively executed in a certain (secure) environment, as for the smart factory example from Section 2.3. For this reason, the operator model of a data stream processing system must provide a way to describe operators with their respective deployment and runtime constraints. These constraints are defined by operator developers and application developers.
D. **Deal with heterogeneous system topology**: Certain tasks may be computationally expensive making the usage of dedicated hardware a necessity [5]. Recent research, such as [6], shows that creating dedicated operators running on FPGA chips is beneficial. Data streaming systems must support integration of such highly domain-specific processing logic to exploit this potential. This results in a broad variety of participating computing nodes, which must be managed by the stream processing system.
E. **Exploit mobile devices as data source and execution nodes**: In the scenarios described beforehand, mobile devices consume data but also provide data relevant for correct processing. E.g., for the visualization pipeline scenario the client receives a video stream of the rendered scene but must also provide the current position and viewing direction to set the viewport correctly. In the trajectory compression scenario the mobile client provides data of the current position.

4 Related Work

Complex data stream scenarios, as discussed in Sections 2 and 3, raise requirements that go beyond state-of-the-art data stream processing systems. To support domain-

	Custom Data-Processing	**Structured / Unstruct. Data**	**Deployment and Execution**	**Heterogeneous System Topology**	**Mobile Devices Support**
Stream Globe	provided by query	relational / -	-	desktop computers	data source
PLACE*	-	relational / -	-	desktop computers	data source
Borealis	local op.	relational / -	-	desktop computers	-
SystemS	local op.	relational / arbitrary	deploym. & runt. (comp.)	supercomp., mainframes	-
NexusDS	global op.	objects / arbitrary	deploym. & runt. (part.)	deskt. comp., mobile dev.	data source & execution

Table 1. Comparison of data stream processing systems

specific requirements and to attract a wide range of streaming applications, stream processing systems should support the efficient integration of domain-specific functionality. All systems considered in this paper allow to access and process data in a distributed fashion, which is a key feature to efficiently process data. Table 1 compares state-of-the-art distributed data stream systems according to the requirements from Section 3.

StreamGlobe [7] provides a fixed set of operators at each computing node. Custom operators must be included with each query that is submitted to the system, thus limiting the usability of that operator for this particular query. Later queries have to send the custom operator again in order to use it. StreamGlobe does not support the processing of unstructured data and does not provide a way to define custom deployment and execution rules.

PLACE* [8] is a spatio-temporal distributed stream processing system for moving objects. The integration of custom operators is not considered and unstructured data is not supported. The whole execution of queries is performed in backbone servers that also track the moving objects as they move along. Therefore, mobile devices are exploited as data source but not integrated with the actual query execution.

In Borealis [9], a system developer must install operators manually at each processing site which makes them locally available. Only structured data is supported limiting the usage for the scenarios sketched. Borealis neither supports heterogeneous system topologies, nor operator constraints.

SystemS [10] allows to add custom operators locally at each processing site, provided they do not require third party libraries. Otherwise, a system administrator must configure and install additional software packages before the operator can be used. To the best of our knowledge, beside NexusDS, SystemS is the only data stream processing system that is capable of handling structured as well as unstructured data. SystemS is designed for clusters of computing nodes and thus assumes homogeneous computing nodes according to our classification.

NexusDS shares a few concepts with the discussed systems, but differs in several ways: Arbitrary custom operators can be added to NexusDS and made available at a global scope by publishing them via a repository site managed by NexusDS. Furthermore, the operator model of NexusDS supports structured as well as unstructured data and the operators can be deployed and executed according to application-specific constraints specified within the query graph. NexusDS supports a variety of devices ranging from simple mobile devices with a reduced set of capabilities to desktop computers with dedicated hardware installed.

5 NexusDS

NexusDS [11] is a distributed stream processing middleware targeting the requirements from Section 3, thus providing enhanced support for complex application scenarios demanding for specialized techniques. In the following we describe how NexusDS satisfies each single requirement.

5.1 Custom Data Processing Logic

Applications formulate their data processing scheme by defining a query graph that represents the data sources as well as the data processing. The operator set of NexusDS is extensible, i.e., an application developer can integrate even highly specialized and domain-specific operators. For this, developers of such operators enrich the actual operator implementation by descriptors which are attached to the operator as *meta data describing the operator*. The operator meta data include characteristics such as the *accepted and delivered data types*, the number of *inputs and outputs*, the operator *execution requirements* specifying special software and hardware requirements, or *presets* allowing to specify commonly used settings for the operator parameters. When interconnecting operators, only inputs and outputs of the same data format can be combined. By building on this flexible meta data concept arbitrary operators can be integrated into NexusDS.

5.2 Support Structured as well as Unstructured Data

NexusDS uses the Augmented World Model (AWM) [12] as the basic structured context-data format. The AWM is an object-oriented, extensible data model tailored to the needs of location-based applications. Like common object-oriented data models, the AWM supports (multi-) inheritance. In contrast to those, AWM objects do not have a fixed structure, but are sets of attributes, where the type of the object is just an additional attribute. An object can even contain multiple instances of the same attribute, in which additional meta data can be used to distinguish the instances.

For the Nexus system, this concept has two main advantages. Firstly, it greatly facilitates the integration of data coming from different providers. Different representations of the same object can be integrated by unifying the two sets, which even works, when the two data providers disagree about the type of the objects. Resolving such inconsistencies can be either done by the system in an additional step, or can be left to the application. Secondly, the concept of multi-attributes allows to represent dynamic attributes like the position of a mobile object. In this case, the object contains multiple instances of the position attribute, where each instance contains an additional meta data item representing the temporal validity of this instance.

In addition to AWM objects, NexusDS can also handle application-specific data streams, which allows, e.g., operators generating a video stream (unstructured data). For this, application developers have to develop the specific operators processing the application-specific data. Developers must also provide the respective serialization and deserialization operators to support distributed processing.

5.3 Definition of the Actual Deployment and Execution

Data processing schemes in NexusDS are formulated as query graphs. The *Nexus Plan Graph Model* and *Nexus Execution Graph Model* (*NPGM* and *NEGM*) arrange the operators used for data processing and support the definition of deployment and runtime constraints. The difference between *plan graph* and *execution graph* is that the execution graph specifies the whole deployment (physical operators, execution environments, etc.) whereas the plan graph is a *hybrid graph model* to orchestrate data-flow graphs composed of boxes. Boxes are an abstraction and can either be sources, sinks, or operators. Hybrid graph model means NPGM allows to define properties of the query graph by *deployment* and *runtime constraints*. The annotation of the query graph by constraints allows to influence the actual deployment process and furthermore defines the runtime behaviour of the boxes. NPGM query graphs are not directly deployable as there may exist boxes that are not mapped to a concrete physical operator (logical boxes) and the distribution of the physical operators is still unknown. Before execution, NPGM graphs must be mapped to an executable representation (NEGM) which in the next step can be deployed and executed on the available infrastructure.

To create a NEGM graph the NPGM graph is fragmented into subgraphs according to annotated constraints. These fragments are deployed and executed on different heterogeneous and distributed nodes. Query graph fragmentation is a highly complex task. We adopt a meta-heuristic approach that allows us to efficiently find a suitable query graph fragmentation. By deploying and executing the fragments with their respective boxes on different computing nodes, NexusDS can efficiently process complex tasks, such as the streamline calculation scenario.

5.4 Deal with Heterogeneous System Topology

Operators may require specialized hardware, such as a GPU. To find suitable processing nodes, NexusDS operators must be annotated with constraints describing the requirements in terms of hardware and software resources [2]. This meta data is used during the transformation from NPGM to NEGM to constrain the selection of suitable nodes for the specific operator and to guarantee a valid deployment decision. Consequently, the execution environments also must be annotated with the same kind of constraints. This information is used to match operators to concrete execution environments satisfying the operator requirements.

5.5 Exploit Mobile Devices as Data Source and Execution Nodes

Nowadays mobile devices have multiple sensors that collect data of the mobile device's context. As shown in Section 2.1, this data is often important in order to make a stream query graph work properly, e.g., for setting the area of interest according to the current mobile device's position. Processing capabilities of modern mobile devices have increased in the past decade but are still not suited for execution of complex operators, such as the streamline calculation and postponed rendering of complex sceneries. In NexusDS, mobile devices can be integrated as data sources as well as processing nodes executing certain tasks, e.g., filtering data elements before sending them to subsequent processing nodes.

6 Performance Evaluation

We have evaluated our work by implementing the scenario presented in Section 2.1 in NexusDS. Different domain-specific operators have been implemented that exploit the capabilities NexusDS offers. The resulting application is shown in Fig. 1(b). The test platform consisted of up to four commodity PCs with Q6600 CPUs, 4 GB DDR2 main memory, and GBit Ethernet interconnection. The mobile platform receiving the rendered images was a HTC Nexus One. All operators were implemented in C++ (to fully exploit GPU capabilities) and embedded in NexusDS which is written in Java. The native visualization application (without the overhead of NexusDS) reached a runtime of approx. 49.4 seconds on a single node for each rendered image. The same application, integrated in the NexusDS system, performed the same task on the same node in approx. 51.1 seconds, resulting in a small overhead due to the NexusDS framework. When exploiting the distribution capabilities of NexusDS we achieve a speed-up factor for the query graph of approx. 5.9 for a distributed configuration with 16-way parallelism of performance-critical operators[3] using four nodes. If we consider only performance-critical operations for this configuration even a speed-up factor of approx. 8.4 is measured. This indicates NexusDS scales well with increasing parallelism, showing the scalability of our system w.r.t the parallelism of visualization operators that rely on the flexibility of NexusDS [1].

7 Conclusion

As demonstrated by the complex examples presented in Section 2, it is mandatory to provide an adaptable execution environment for streaming applications. In this paper we have discussed requirements of such applications. These applications, although originating from different application domains and having different processing constraints, share a common processing principle, i.e., data stream processing. They are often embedded in a utterly heterogeneous and distributed infrastructure ranging from desktop computers to mobile devices. Therefore a data stream processing system needs to supply adequate techniques to provide a tight integration of standard and non-standard

[3] Performance-critical operators are operators requiring a huge amount of computation time compared to the other operators in the query graph.

processing schemes. These techniques reach from the integration of specific operators to the manipulation of the actual graph deployment. NexusDS is especially tailored to meet these requirements. By the unique techniques provided by NexusDS it is possible to realize applications relying on highly domain-specific concepts.

These features raise new challenges including the development of a query engine that is aware of the domain-specific constraints and that exploits them when searching for a query graph distribution.

References

1. H. Sanftmann, N. Cipriani, and D. Weiskopf, "Distributed context-aware visualization," in *IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, 2011, perWare 2011.
2. N. Cipriani, C. Lübbe, and A. Moosbrugger, "Exploiting constraints to build a flexible and extensible data stream processing middleware," in *The Third International Workshop on Scalable Stream Processing Systems*, Atlanta, USA, 4 2010.
3. N. Hönle, M. Großmann, D. Nicklas, and B. Mitschang, "Preprocessing position data of mobile objects," in *MDM*, X. Meng, H. Lei, S. Grumbach, and H. V. Leong, Eds. IEEE, 2008.
4. D. Lucke, C. Constantinescu, and E. Westkämper, "Smart factory - a step towards the next generation of manufacturing," in *Manufacturing Systems and Technologies for the New Frontier*, M. Mitsuishi, K. Ueda, and F. Kimura, Eds. Springer London, 2008.
5. J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. J. Purcell, "A survey of general-purpose computation on graphics hardware," in *Eurographics 2005, State of the Art Reports*, Aug. 2005.
6. J. Teubner and L. Woods, "Snowfall: Hardware stream analysis made easy," in *BTW*, ser. LNI, T. Härder, W. Lehner, B. Mitschang, H. Schöning, and H. Schwarz, Eds., vol. 180. GI, 2011, pp. 738–741.
7. R. Kuntschke, B. Stegmaier, A. Kemper, and A. Reiser, "StreamGlobe: processing and sharing data streams in grid-based P2P infrastructures," in *VLDB '05: Proceedings of the 31st international conference on Very large data bases*. VLDB Endowment, 2005.
8. X. Xiong, H. G. Elmongui, X. Chai, and W. G. Aref, "Place: A distributed spatio-temporal data stream management system for moving objects," in *Mobile Data Management, 2007 International Conference on*, 2007.
9. D. J. Abadi, Y. Ahmad, M. Balazinska, U. Cetintemel, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *Second Biennial Conference on Innovative Data Systems Research (CIDR 2005)*, Asilomar, CA, January 2005.
10. L. Amini, H. Andrade, R. Bhagwan, F. Eskesen, R. King, P. Selo, Y. Park, and C. Venkatramani, "SPC: a distributed, scalable platform for data mining," in *DMSSP '06: Proceedings of the 4th international workshop on Data mining standards, services and platforms*. New York, NY, USA: ACM, 2006.
11. N. Cipriani, M. Eissele, A. Brodt, M. Grossmann, and B. Mitschang, "NexusDS: A Flexible and Extensible Middleware for Distributed Stream Processing," in *IDEAS '09: Proceedings of the 2008 international symposium on Database engineering & applications*. New York, NY, USA: ACM, 2009.
12. D. Nicklas and B. Mitschang, "On building location aware applications using an open platform based on the NEXUS augmented world model," *Software and System Modeling*, vol. 3, 2004.