DBMS meets DSMS
Towards a Federated Solution

Andreas Behrend[1], Dieter Gawlick[2] and Daniela Nicklas[3]
[1]*University of Bonn, Roemerstr. 164, 53117 Bonn, Germany*
[2]*Oracle Redwood City, 500 Oracle Parkway, Redwood City, CA 94065, U.S.A.*
[3]*Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany*

Keywords: Data Stream Management, Event Processing, Active Databases, Database Architecture, Query Processing.

Abstract: In this paper, we describe the requirements and benefits for integrating data stream processing with database management systems. Currently, these technologies focus on very different tasks; streams systems extract instances of patterns from streams of transient data, while database systems store, manage, provide access to, and analyze persistent data. Many applications, e.g., patient care, program trading, or flight supervision, however, depend on the functionality and operational characteristics of both types of systems. We discuss how to design a federated system which provides the benefits of both approaches.

1 INTRODUCTION

Traditional database management systems (DBMS) are capable of persistently storing and efficiently querying and analysing large amounts of 'static' data. These systems are not well-suited, however, for managing and analysing highly dynamic data—so-called data streams—which are generated, e.g., by sensor networks, financial tickers, or transaction loggers. Therefore, over the last 15 years, specialized systems have been developed, so-called data stream management systems (DSMS), which are explicitely designed for an online analysis of rapidly changing data. This systems, however, lack support for persistance. DSMS are so important that they are the subject of a variety of research project since many years and an integral part of the infra-structure of the major software vendors.

DSMS support long-running, persistent queries which continuously analyse ordered sequences of items. The performance gain over traditional DBMS is mainly achieved by avoiding the overhead for persitance and transactionality. Additionally, DSMS leverage the long livity of the continuous queries, share work between queries, and evaluate stream data incrementally, There are many applications that require the functionality of DSMS and DBMS, such as flight supervision and patient care—actually the need for pesistance and provenance is the overriding requirement. In these cases, the requirements for persistant data management are so important that the DSMS functionality is provided by the DBMS using triggers and/or persistant queries (Schüller et al., 2012; Guerra et al., 2011). However, there are major draw-backs: these systems are not scalable to large amount of incoming data and the ability for pattern recognition is way below the level typically found in DSMS.

The alternative is obviously the use of a DSMS complement by a DBMS. In this case the DSMS has to identify the information that needs persistance and the application has to store it in a DBMS. This approach requires a deep understanding of system design and it often not achiveable by applications programmers due to the many incompatibilities between these types of systems. Consequently, the development, test, and maintenance cost are often prohibative.

Consequently, there is a need for a system that includes both, the functionality of a DSMS and the functionality of a DBMS. We propose a federated solution that leverages the strength of both technologies and hides the differences as much as possible. This approach has the potential to broaden the application spectrum of stream processing systems considerably. The design of such a federated system, however, is not a simple task due to the heterogeneity of the underlying systems with respect to the supported data models, query languages and operational characteristics (Babcock

Behrend A., Gawlick D. and Nicklas D..
DBMS meets DSMS - Towards a Federated Solution.
DOI: 10.5220/0004122501570162
In *Proceedings of the International Conference on Data Technologies and Applications* (DATA-2012), pages 157-162
ISBN: 978-989-8565-18-1

et al., 2002). For example, DBMS do not only support relational data models, but also XML, RDF, multimedia, or text data and allow users to add domain and/or application specific data models. Although DSMS also support various data models, there remain differences in the structure and semantics of data which have to be bridged by a federated solution.

The same applies for the heterogeneity of query languages. Typically, DSMS provide variants of SQL such as CQL (Arasuet al., 2006) or StreamSQL (StreamBase Systems 2012), or complex event languages like Sase (Gyllstrom et al., 2007). Such languages suppport new stream-related concepts such as window expressions, sketches and approximate answers and differ in their operational characteristics in comparison to standard SQL statements. Although SQL also provides temporal support, history management (e.g. Oracle's total recall) and event processing, there are still intricate details which make the development of a common federated query language difficult (cf. Section 1.2). Beside declarative query interfaces, many DSMS even support functional stream processing languages, like Aurora's boxes and arrows (Abadi et al., 2003), or Infosphere Stream's Stream Programming Language SPL (Biem et al., 2010) which are hardly compatible with SQL. Beside pure DSMS, there are specialized systems like kdb+ (Kx Systems 2010) or DBToaster (Kennedy et al., 2011) that are optimized for processing real-time and historical data in main-memory.

In order to cope with these challenges, we propose semantic information layers which help to systematically identify challenges and solutions for designing a federated data management system for active data. But before these layers are presented in more detail, we discuss a use case in which various requirements of the proposed federated solution are examplarily illustrated.

1.1 Scenario

To illustrate the proposed architecture, we will follow a scenario from the health care domain. However, scenarios with similar characteristics can be found in other domains like traffic management, smart grids, or intelligent city infrastructures. Health care providers have to deal with an ever increasing amount of data that have to be acted upon by applying knowledge and regulations, both of which growing rapidly in amount and complexity.

Capturing of patient data as EMRs (Electronic Medical Records) is fast becoming common place. EMRs represent a wide variety of data; data have to be retained for an extended period of time and access to and analysis of these data has to be well supported while it is strictly controlled and has to be documented.

Doctors are typically unable to devote enough time to review all captured data (facts), therefore real time support is required to extract important information. This information has to be captured and brought into attention of doctors with the proper level of urgency and thus, facilitating situation-awarenes. This transformation from facts to information should be based on the codified knowledge of the medical community. The transformation has to take personal preferences into account and has to be fully auditable.

Patient data arrive with various delays and may go through revisions; vitals are immediately available while test depending on chemical reactions and cultures may take hours or even days. The real time support has to be able to deal with these widely varying delays. Auditing requires that it must be easy to understand which facts were available for which derived information.

Medical knowledge is evolving at a fast pace; if new — codified— knowledge becomes available existing facts have to be automatically reviewed in order to identify and act upon any relevant information that was not derived when the facts became available.

1.2 Challenges

From this scenario, we see the following four main challenges that should be addressed by a data management system:

Query Processing over Streaming and Persistant Data: to support applications like the one depicted in the scenario, the DSMS may need to reference DBMS data. The performance of DSMS can only be maintaned if relevant data from the DBMS are cached. Therefore much attention has be be given in deriving the right caching strategy from each continuous queries and also add a global optimization. Directives have to be given to the DBMS in order to notify the DSMS about changes of the cached data.

Heterogenous Data Models: DSMS are focused on temporal support for large incoming data streams; while temporal support for DBMS is only slowly gaining traction. Assuming bi-temporal support in the DBMS; it should be possible to represent any stream of data (events) as a DBMS object. Challenges are to verify this asumption and to

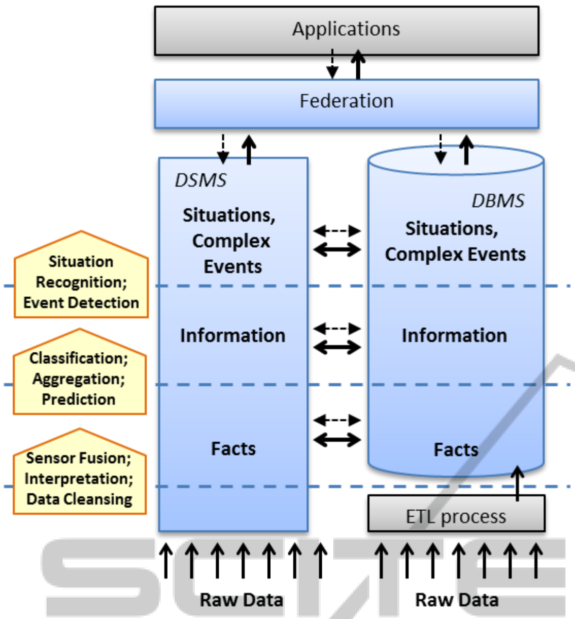
Applications
Federation
DSMS
Situations, Complex Events
Information
Facts
DBMS
Situations, Complex Events
Information
Facts
ETL process
Situation Recognition; Event Detection
Classification; Aggregation; Prediction
Sensor Fusion; Interpretation; Data Cleansing
Raw Data
Raw Data

Figure 1: Federated Architecture for DSMS and DBMS.

develop a mapping of any DSMS object to DBMS objects.

Complexity of queries: Realizing scenarios like the one discussed will lead to complex query plans. The reason is the joint focus on continuous queries in DSMS and on ad-hoc queries in DBMS. Even the event support in DBMS — whether realized through triggers and registered queries—does syntactically and semantically not match the DSMS support. This calls for a *common query language*. Secondly, for such answering queries, the raw data has to be interpreted, aggregated, classified, and maybe predicted; if the data management system wants to support this (to provide higher-level semantics for different applications), the resulting queries have to contain the logic for all these processing steps. Thus, the query language has to be directed towards common *semantic information layers* (similar to DBMS views).

Provenance: When critical decisions are taken based on aggregated information, it is often crucial that the system is always able to tell how the aggregation was performed and what data has contributed to it. In an active data management szenario, it is also important to track which events where delivered to which application and when.This problem can be handled by assuming that any information that is considered to be critical is retained in the DBMS and can be accessed using the temporal DBMS support. If applications are using transactional support, any result can be audited by revieweing the — temporal — data, which version of the application was used, which requests were initiated by the application, and which authentification was used.

1.3 Contribution

In this position paper, we argue for the following:

- Many applications need management for both streaming and persistent data
- Many applications need timely analysis of data — event processing — as well as support for ad-hoc queries and provenance.
- A federation of a DSMS and a DBMS should be feasible to achieve this.
- Semantic processing layers help in designing such a federated system by making differences between DSMS and DBMS system transparent to applications.

Figure 1 guides us through the paper: we first introduce the semantic information layers “Facts”, “Information” and “Situations” in Section 2 and the overall federated architecture in Section 3. We identify relevant implementation techniques and challenges in Section 4. Finally, we conclude with Section 5 and show future directions for research and industry.

2 SEMANTIC INFORMATION LAYERS

As we can see from the applications, modern sensor-based applications need more than just pure fact-based data processing. Thus, in a full-fledge data processing architecture, we need to support different semantic layers of information. Such layers help in coping with the complexity of federated query plans, making the resulting architecture much more maintainable and flexible.

Fact Layer: Facts are statements, observation, or any other piece of data that is part of the basic information schema. There are various sources for facts: When raw data comes in from sensors, it has to be often pre-processed to extract features and facts. Similarly, when data is integrated from external data sources, ETL (“extract-transform-load”) processes are used to insert facts in the database.

Information Layer: Typically, applications need not only plain facts but some derived information. In classical fact-based data base management systems, this derivation is expressed by the SQL query that the application sends to the database. Various operations can lead form facts to information, step, like classification (e.g., mapping a blood pressure value to classes like "high", "normal", or "low) or aggregation. Prediction functions use application knowledge (pre-modeled or learned by data mining algorithms) and derive probable future states from present facts. Such trends and predictions are application-relevant information, too.

When information is only generated within the application layer, it can hardly be shared between applications. Thus, we propose to introduce a shared information layer in the data management system that can be directly queried by applications, similar to views in databases.

Situation Layer: On the top layer, we define situations as a relevant combination of facts and information that needs to be communicated to subscribed applications. Typically, only the change of state is communicated, i.e., in the moment when the situation occurs. Situations could be modeled as complex events (i.e., patterns evaluated on basic events) or as continuous queries (i.e., a query that is continuously evaluated). When archived, a situation would become part of the information layer.

The concept of semantic information layers is somehow similar to the well-known concept of views in data-bases and can thus help us in similar ways: First, since application-specific higher level information if explicitly modeled and expressed, it is easier for the application developer to communicate with the domain expert and to implement new queries. And secondly, the layers act as abstraction levels within the system design, so that the lower-level processing can be changed without changing higher-level processing and models.

However, there are also significant extensions to simple views: we see the need for a much richer set of operations to express the derivation of higher-level information, like prediction of future states, classification, or aggregation. Such operations also encode application-specific knowledge, represented in models that are either specified by the domain expert or are learned by data mining techniques over persistant data.

3 FEDERATED ARCHITECTURE

From the discussion of the scenario, we see that there is a need for data management systems that support both the efficient management of high volumes of stored data, and the processing support of high-performance streaming systems. To leverage the benefits of both systems, we propose a federated architecture. Note that in future data management systems, both sides may be integrated into one processing engine; however, for this, the challenges of a dual system have to be resolved, too.

Figure 1 shows an overview on the proposed architecture. Applications can issue continuous queries or define information models (needed for classification and aggregation) at the federation layer.

Here, these queries are transformed into executable query plans in the underlying systems, which are a DBMS and a DSMS. At each of the data processing layers, queries and data can be exchanged between the two systems.

Note that this is a streaming system; thus, the query plans are not executed just once, but deployed/registered to the underlying systems. Whenever new data arrives, the queries are executed again with this new data. If the query represents a continuous query, the new result set is communicated to the application. If it is a complex event pattern, the new data is treated as new basic events, and the systems check whether new complex event evaluates to true. Both cases are covered by the concept of "situations"; thus, the result every application query belongs to the upper most semantic information layer.

Since the registered queries are typically long-running, query sharing plays a crucial part for optimizing the performance. For every new query, an ideal optimizer at the federation component would recognize which already running query plans could be re-used. However, since cross-platform optimizations over complex query plans might be too expensive or not possible, the semantic information layers provide another benefit: they already represent sharable query plans, since every modeled concept at the information comes with a query plan to derive it. If multiple queries use the same information concept, the system can re-use this query plan for both situations.

4 IMPLEMENTATION

In order to realize the federation layer depicted in the architecture from Figure 1, we can leverage existing techniques provided by the underlying systems. The arrows between the two subsystems indicate the data transfer that should be supported for each identified semantic layer. To this end, compatible operators and DB techniques have to be identified that allow for resuming the data processing task coming from the DSMS resp. DBMS subsystem.

4.1 Existing Techniques

The most relevant DBMS techniques that have a close relationship to stream processing are triggers, materialized views and registered (continuous) queries (e.g., Oracle's CQN). Triggers are active rules that allow for specifying the automated reaction to updates on tables or views. Due to their instance-oriented, push-based data processing, they directly correspond to incremental stream operators of DSMS. Materialized views and registered queries are techniques that allow for automatically refreshing query results as new data arrive. In this regard, they directly correspond to the concept of continuous queries in DSMS.

From the DSMS perspective, query plans, caching and batching are techniques which are related to DBMS concepts. Query plans are represented by operator trees in which operator subtrees coming from the DBMS query engine could be integrated. Caches of stream operators may even comprise persistent DBMS data which can then be jointly processed with the dynamic stream data. In this way, static domain knowledge — stored in the DBMS — can be used for a stream analysis, too. Finally, batching allows for combining stream data into a single update request that can be more efficiently handled by the set-oriented data processing strategy of a DBMS than individual ones.

All these techniques ought to be used in order to realize a federated query processing. The main challenge, however, is to find a meaningful and optimized combination of such techniques. The latter depend on the chosen query type which should be supported by the proposed federated system.

4.2 Federated Query Types

In Figure 2, we depict typical query types that can be realized within the federated architecture.

Pure DSMS Query Execution: Query (1) shows a pure DSMS query execution: without any interaction with the DBMS, the query processes raw data to facts, combines the facts to information, and if a relevant combination of information occurs, the result of this processing is sent to the application. One example for this query could be the continuous assessment of changes in temperature and cardiogram data for ICU patients. Such queries are well-suited for high volume data streams whose processing can be performed in main memory and where no reliable persistent storage of the data is needed.

Archiving Queries: If the facts derived from the

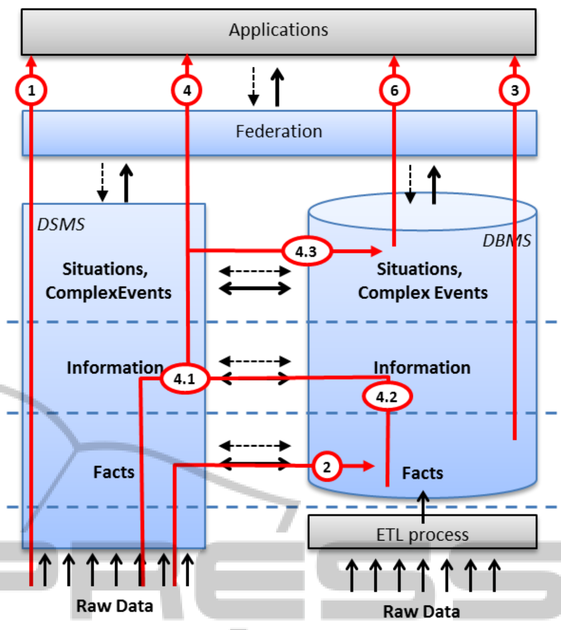

Figure 2: Federated Query Execution.

raw sensor data should be kept for future analysis or because of legal reasons, the persistent storage of the stream data using the DBMS would be desireable. Query (2) implements this functionality and is regsarded as archiving query*:* the DSMS performs a preprocessing of the raw data, but all facts are transferred and stored in the DBMS for further analysis. In order to cope with the frequency of streaming data, the DSMS could batch the induced inserts. Althought archiving queries are depicted for the facts layer, they are meaningful for all other semantic layers, too. An example for an archiving query is the relibale storage of patient data such as blood tests and medications. On the information layer, an archiving query storing very unusual cardigram data could be imagined allowing for legally justifying dangerous and heart-related drug treatments.

Continuous DBMS Query: Query (3) represents a continuous query solely realized within the DBMS. In this way, the advantages of both system types are directly supported. This query type, however, is solely applicable for particular stream frequencies and volumes which are certainly lower than the ones manageable by a DSMS. An example for this query type is the continuous analysis of patient records of a hospital for operative reasons.

Complex DSMS/DBMS Query: Query type (4) represents complex DSMS/DBMS queries with various data flows between the two subsystems. A classic example is the continuous determination of diagnoses based on medical domain knowledge and current patient data. In this query type, raw stream data is first pre-processed to facts (step 4.1) quite similar to the query type (1). In order to decide about the present situation, however, additional information is needed which is provided by the DBMS. Thus, a join between stream data from the DSMS and DBMS data is needed (of course, by the help of DSMS caching techniques). Otherwise, the DBMS has to provide the relevant information (e.g., using a materialized view) by applying domain knowledge on its stored facts.

Based on this combined information, situations can be derived which are highly relevant for the monitoring application. It might be necessary to make the derived situations available directly within the DBMS (selection point 4.3) in order to facilitate a statistcial analysis of this temporal data. This kind of queries (6) could be used, e.g., to analyse the consequences of changed medical knowledge on derived diagnoses or therapies retrospectively.

4.3 Implementing the Federation Layer

In the introduction we have already indicated the principle differences between DBMS and DSMS which makes the realization of the federation layer a complex task. The most important goal is the development of a unified query languages based on a common temporal data model and an integrated algebra. For optimizing complex DBMS/DSMS queries, new cost models have to be developed which allow for controlling the data flows between the two subsystems. Additionally, new query rewriting strategies have to be investigated for an algebraic-based optimization of mixed query types. These techniques form the basis for an automated and transparent distribution of (parts) of queries to the underlying subsystems.

A federated query processing ought to be able to adapt to new stream characteristics which, e.g., require a continuous DBMS query to become a DSMS one due to an increased update frequency. To this end, cost-based migration techniques have to be developed. Finally, the storage of temporal data and the needed history access by the DBMS call for new compression and efficient access structures such as Oracle's total recall.

5 CONCLUSIONS

This paper argued for a federated approach to managing persistent and active data in a uniform way. We discussed general challenges which will be encountered when designing such a system, i.e.., technological requirements needed for controlling its operational characteristics. Additionally, we showed how the combination of DBMS and DSMS technologies may lead to new functionalities such as novel (federated) query types. In order to systematically develop this functionality, we propose semantic information layers that additionally help to design and use a federated system in a methodological way.

REFERENCES

Abadi, Daniel J., Don Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. 2003. “Aurora: A New Model and Architecture for Data Stream Management.” *The VLDB Journal* 12 (2).

Arasu, A., S. Babu, and J. Widom. 2006. “The CQL Continuous Query Language: Semantic Foundations and Query Execution.” *The VLDB Journal* 15 (2)

Babcock, Brian, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. “Models and Issues in Data Stream Systems.” In *PODS'02*, 1. Madison, Wisconsin.

Biem, Alain, Eric Bouillet, Hanhua Feng, Anand Ranganathan, Anton Riabov, Olivier Verscheure, Haris Koutsopoulos, and Carlos Moran. 2010. “IBM Infosphere Streams for Scalable, Real-time, Intelligent Transportation Services.” *In SIGMOD'10*. ACM.

Guerra, Diogo, Ute Gawlick, Pedro Bizarro, and Dieter Gawlick. 2011.“ An Integrated Data Management Approach to Manage Health Care Data.” In *BTW*, 596–605.

Gyllstrom, D., E. Wu, H. J Chae, Y. Diao, P. Stahlberg, and G. Anderson. 2007. “Sase: Complex Event Processing over Streams.” In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research, Asilomar*.

Kennedy, Oliver, Yanif Ahmad, and Christoph Koch. 2011. “DBToaster: Agile Views for a Dynamic Data Management System.” In *Cidr*, 284–295.

Kx Systems. 2010. “Kdb+ White Paper.” http://kx.com/papers/Kdb+_Whitepaper-2010-1005.pdf.

Schüller, Gereon, Roman Saul, and Andreas Behrend. 2012. “AIMS: A Tool for the View-Based Analysis of Streams of Flight Data.” In *SSDBM (accepted)*.

StreamBase Systems. 2012. “StreamSQL Guide, StreamSQL Online Documentation.” http://streambase.com/developers/docs/latest/streamsql/index.html.