A Demonstration of the MaxStream Federated Stream Processing System

Irina Botan [1], Younggoo Cho [2], Roozbeh Derakhshan [1], Nihal Dindar [1], Ankush Gupta [1], Laura Haas [3], Kihong Kim [2], Chulwon Lee [2], Girish Mundada [2], Ming-Chien Shan [2], Nesime Tatbul [1], Ying Yan [2], Beomjin Yun [2], Jin Zhang [2]

[1]*ETH Zurich*
{irina.botan, droozbeh, dindarn, guptaan, tatbul}@inf.ethz.ch

[2]*SAP Labs*
{young.goo.cho, ki.kim, ch.lee, girish.mundada, ming-chien.shan, ying.yan, beom.jin.yun, gene.zhang}@sap.com

[3]*IBM Almaden Research Center*
laura@almaden.ibm.com

***Abstract*— MaxStream is a federated stream processing system that seamlessly integrates multiple autonomous and heterogeneous Stream Processing Engines (SPEs) and databases. In this paper, we propose to demonstrate the key features of MaxStream using two application scenarios, namely the Sales Map & Spikes business monitoring scenario and the Linear Road Benchmark, each with a different set of requirements. More specifically, we will show how the MaxStream Federator can translate and forward the application queries to two different commercial SPEs (Coral8 and StreamBase), as well as how it does so under various persistency requirements.**

I. Introduction

Stream processing has been widely used in an increasing number of application domains such as sensor-based monitoring, financial services, operational business intelligence (BI), and monitoring of computer systems and services. As the range of these applications widens, a variety of new requirements in terms of architectural design, functionality, and performance continues to emerge, which drives further research in this area.

Despite the availability of several academic and commercial data stream processing engines (SPEs) today, it remains hard to develop and maintain stream-based applications. We see two dominating reasons for this difficulty:

1. Heterogeneity: One major difficulty is the lack of standards, and the wide and changing variety of application requirements. Consequently, existing SPEs vary widely in data and query models, APIs, functionality, and optimization capabilities. This has led to some organizations using multiple SPEs, based on their application needs. The heterogeneous and continuously evolving nature of today's streaming landscape not only introduces complexities in choosing the right engine for a given application, but also makes application development and maintenance hard. The need for standardization has recently been acknowledged and a few initiatives in this direction have been started (e.g., [1]), but more work is needed before a full standard can emerge.

2. Stored-Streaming Divide: A second problem is that management of stored data and streaming data are still mostly treated as separate concerns, although applications increasingly require integrated access to both. Most SPEs recognize this need and provide connections to external DBMS engines. However, this type of SPE-DBMS integration came only as an afterthought, and therefore, is still supported at a somewhat artificial level by most of the SPEs except by a recent few (e.g., Truviso [2]).

We have also examined several real use case scenarios and benchmarks provided by SAP as well as commonly used by the research community, and have identified two different classes of streaming applications.

The first class includes business monitoring applications such as RFID-based supply-chain management [3] and the SAP Sales and Distribution benchmark [4]. Such business monitoring applications may deal with high data volumes, especially during peak periods; however, data rates and latency requirements are relatively relaxed. Processing delays on the order of (tens of) seconds can be tolerated. On the other hand, due to industry regulations, there may be strict requirements that no input and output events be lost, forcing all events to be stored persistently in a database, in addition to whatever live processing is required.

A second class of streaming applications has more demanding scalability constraints, such as the Linear Road Benchmark (LRB) for traffic monitoring [5]. The requirements of this class sharply contrast with the first class: input rates can be much higher; latency requirements are much stricter (at most a few seconds); but on the other hand, event persistence is typically not necessary.

In a recent publication, we have proposed the MaxStream federated stream processing architecture as a solution approach to support both of the above classes of applications in the face of the heterogeneity and stored-streaming challenges [6]. MaxStream introduces a federation layer that sits between client applications and a collection of underlying SPEs and database engines. The federator acts as a common gateway over these engines. As such, it hides the potential differences of the underlying engines from the application by presenting

a common API and query execution model. It also facilitates porting the application to another SPE, or extending the application to meet new requirements, since a different or additional SPE with the requisite functionality can be added. Our federation layer further can bridge the stored-streaming divide as it treats both SPEs and database engines as part of the federation, and includes its own persistent storage. Persistence is optional, however, to allow the system to support the higher scalability requirements of applications such as LRB.

We currently have a working prototype of MaxStream with several key features that we would like to demonstrate through two applications scenarios, one from each class described above. These features include the ability to: (i) accept and push continuous queries down to a selected SPE, (ii) pass input data feeds through to the SPE, (iii) make the result streams received from the SPE available to multiple applications, (iv) provide optional persistence mechanisms for input and output streams within the federator, and (v) process stream-table joins within the federator. It is important to note that all of these features are supported in a transparent manner, hiding the underlying system heterogeneity. For example, we can currently plug in two different commercial SPEs under MaxStream (Coral8 [7] and StreamBase [8]), behind a common client interface.

In this demonstration proposal, we first present an architectural overview of our MaxStream system, highlighting the features to be presented during the demonstration. Then we summarize two application scenarios, namely Sales Map & Spikes and the Linear Road Benchmark, with details on how they will be used to showcase the major technical features of MaxStream, and we conclude.

II. MaxStream Overview

In this section, we will provide an overview of the MaxStream federated stream processing architecture, focusing on its key design decisions and features. We refer the interested reader to our technical report for further system implementation details [6].

The MaxStream Federator has been designed as a middle layer between the client applications and a collection of streaming engines and databases. The key high-level design ideas are that (i) MaxStream provides its users with a uniform interface for queries and data, shielding them from the details of multiple data processing systems that may be running underneath; (ii) the federator has been designed as an extension to the relational database infrastructure, enabling the reuse of the existing robust support for SQL, transactions, persistence, as well as any federation features that may already be available; and (iii) MaxStream follows a lean design that provides "just enough" streaming functionality within the federator layer so as to leverage the capabilities of the underlying SPEs and to avoid overhead.

Though our design is general enough to be implemented on any relational database platform, we have chosen to build the federator on top of the SAP MaxDB relational database by extending its existing federation capabilities [9]. We leverage the basic architecture of the SAP MaxDB federator, while

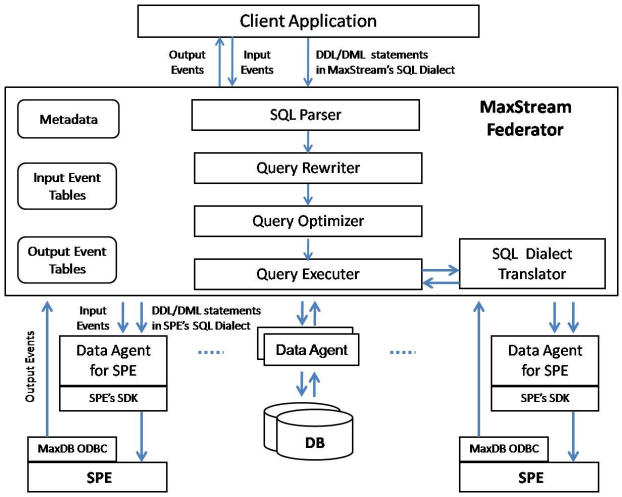

Fig. 1. MaxStream Federator Architecture

adding the required extensions for input and output data streaming mechanisms, and for query language parsing and translation support for continuous queries.

Figure 1 shows the main architectural components of MaxStream. In our initial implementation, we focused on getting the basic MaxStream architecture working and not on advanced features like query optimization. Hence, we currently reuse the Query Rewriter and the Query Optimizer modules without much change. On the other hand, we made major extensions to the rest of the components as explained next.

A client application submits queries in our working MaxStream Federator Language, which is an extension of SQL with continuous querying and windowing constructs. It essentially allows the application to read and write from streams, and permits joins between a stream and one or more static tables. Note that our language syntax has been evolving as we make progress in the formal query model that we have been creating [10]. The SQL parser was properly extended to recognize our current language extensions. After by-passing the Query Rewriter and the Query Optimizer, the Query Executer passes the complete continuous query plan directly to the SQL Dialect Translator, which in turn translates the plan into the streaming SQL dialect of the external SPE that will actually execute the query. Thus, the SQL Dialect Translator has also been extended. The Query Executer receives the translated query back and sends it to the associated SPE through a Data Agent. As shown in Figure 1, the Data Agent is the bridge for all control messages exchanged and all input streams forwarded to the SPE. The output streams from the SPE are written directly into SAP MaxDB tables through an ODBC connection from the SPE into the SAP MaxDB engine, to be presented back to the client application.

There are two alternative ways of streaming inputs in MaxStream: (i) If the input events are to be persisted, they are first materialized in a relational table in MaxStream before they are streamed into the SPE using the ISTREAM operator

[11]. (ii) If there is no persistence requirement, a high-speed in-memory queue is used to pass data through to the SPE.

A symmetric functionality in the federator is the ability to pass outputs received from the underlying SPEs to the corresponding client applications. This can also be done in two alternative ways: (i) If the output stream is to be persisted within the federator, it is first inserted into a special "event table" inside SAP MaxDB. Then the client application can continuously receive results from the event table through a novel mechanism that we have implemented called the "Monitoring Select". (ii) If there is no persistence requirement, the event table is simply created as a SAP MaxDB in-memory table, on which the monitoring select mechanism is used as described above.

Finally, the MaxStream Federator can also conveniently support hybrid queries over streams and tables, building on the existing relational joining mechanisms and the input/output streaming mechanisms that we have added on top.

III. Demonstration Details

In order to demonstrate the key features of the MaxStream system, we will use two application scenarios: Sales Map & Spikes, and the Linear Road Benchmark. The former is a business monitoring application that is inspired by the SAP Sales and Distribution Benchmark (SD) [1] and several other operational BI applications that we have studied [12], and the latter is a road traffic monitoring benchmark developed and commonly used by the academic community for complexity and performance measurements [5]. As such, these two applications match very well with the two typical classes of applications that we have described in Section I.

A. Scenario #1: Sales Map & Spikes

Sales Map & Spikes depicts the scenario of a large international company with many locations worldwide, each with multiple sources of raw operational data. In particular, each location is constantly dealing with new orders, creating invoices, and scheduling deliveries in its geography. An SPE installed at each site keeps track of local aggregate sales volumes and inventory on a minute-by-minute basis, while the corporate headquarters needs to monitor the overall business by maintaining a map of its sites to track the total hourly sales by product and region. In a potentially heterogeneous environment, it is a real challenge to serve this application. We will show that our MaxStream Federator can address the challenge.

For example, let us focus on the site in Rome. Using our federator, the map application poses a continuous query against the Rome site's orders; the results of that query would update Rome on the map. A second application checks for spikes in the sales volume, notifying an executive in Rome when hourly sales exceed a certain limit. Note that the only functionality needed in the federator for this scenario is the ability to push a continuous query down to an SPE, to pass the input data feed through to the SPE, and to make the result stream received from the SPE available to multiple applications. Headquarters may also want to keep a permanent record of the orders (for regulatory reasons) or of the results (for trending). In such cases, the federator also needs to be able to persist the input or output streams.

Figure 2 illustrates the scenario for the Sales Map & Spikes application. All the DDL and DML statements are submitted to MaxStream as a single module. Once MaxStream receives a module it should identify which set of queries must be run inside MaxStream and which ones must be pushed to the underlying SPE. In addition to any regular database tables (Products), it first creates the tables (OrdersTable, TotalPOrdersTable, and TotalSalesTable) in which input and output streams will be kept persistently inside MaxStream. All the CREATE INPUT / OUTPUT STREAM statements and the continuous queries have to be pushed to the SPE after being translated into the SPE's SQL dialect by our SQL Dialect Translator. The INSERT INTO statement is used to populate the input streams (OrdersStream, POrdersStream), which will be feed to the SPE using the ISTREAM operator over the corresponding MaxStream tables. Continuous queries which calculate aggregates over the sales orders for the last hour will be sent to the SPE, and their results (TotalSalesStream, TotalPOrdersStream) will be streamed back into the corresponding event tables (TotalSalesTable, TotalPOrdersTable). In order to notify the client application of the resulting sales events, Monitoring Select operators are used.

One of the main features of MaxStream is the ability to support hybrid continuous queries that require joins between streams and database tables. This is also illustrated in our example scenario above as part of the second continuous query, where we continuously calculate the hourly sum of sales orders for each product sold at a location. For this query to work, the input stream which is persisted in the OrdersTable, must be first ISTREAM'ed and joined with the Products dimension table that is also kept inside MaxStream. Then the resulting POrdersStream in the SPE will be populated using the INSERT INTO statement. After the corresponding continuous query is run on POrdersStream in the SPE, the resulting TotalPOrdersStream can be written into the TotalPOrdersTable event table within MaxStream, from where the client can receive the results via a Monitoring Select. Though not shown in the figure, a similar hybrid join could also be easily performed between a MaxStream output stream and a static table for enriching the output before presenting to the client.

B. Scenario #2: Linear Road Benchmark

Linear road is a benchmark for data stream management systems [5]. It describes a traffic management scenario in which the tolls for a highway system are dynamically computed based on the utilization of those highways and the occurrences of accidents. The input data for LRB consists of car position reports (each car reporting every 30 sec-

[1]The SAP SD Benchmark is one of the most widely accepted server benchmarks. For example, in 2008, 83 SAP SD Benchmark results were certified, while 17 TPC-C results and 14 TPC-E results were certified [4].

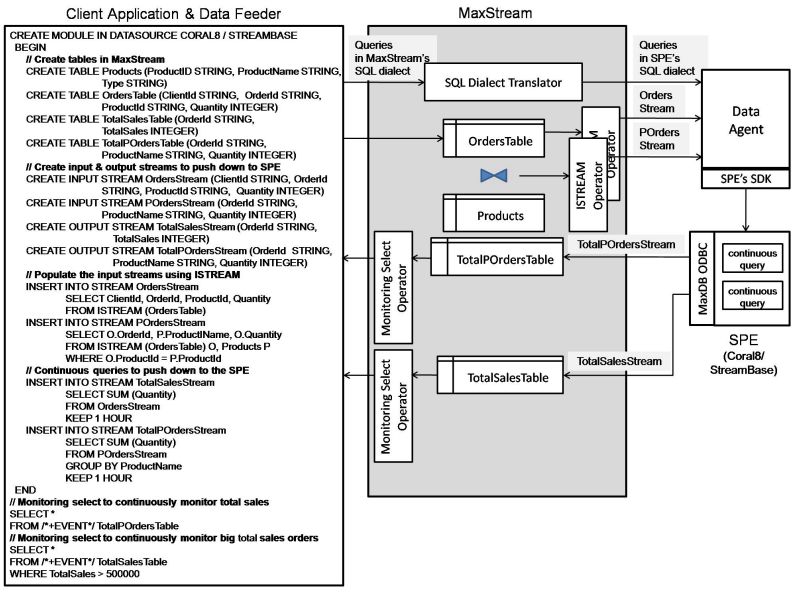

Fig. 2. Sales Map & Spikes Demonstration Scenario

onds) and query requests of the following types: (i) Accident Notification, (ii) Toll Notification, (iii) Balance Query, (iv) Daily Expenditure Query, and (v) Travel Time Estimation. The measure of this benchmark is given by Load, representing the number of highways that can be handled by the stream processing engine. A certain load level is considered to be achieved when all the queries involved in the benchmark are answered correctly within at most 5 seconds after the corresponding query request has been entered into the system.

We have chosen LRB as a demonstration scenario since it is a challenging benchmark in terms of query semantics and complexity, and because of its strict requirements in terms of correctness and performance. Our implementation of LRB is composed of 23 DML statements for the set of queries required by the benchmark. Most of these queries use different combinations of windowing strategies like: time- and count-based sliding or tumbling windows on partitioned or non-partitioned input data. In both scenarios, client application submits a set of DDL statements and continuous queries as well as input events to MaxStream, then MaxStream acts as the gateway to the underlying SPEs. In a recent paper, we have shown that MaxStream using a commercial SPE can support a load factor of 4.0, while the same SPE on its own can also support up to 4.0 [6]. This shows that MaxStream's overhead is acceptable.

In this demo, however, our goal is not to show our LRB performance. We will rather use the benchmark to show the degree of query complexity that we can support in our federator. Also, different from the business scenario, LRB does not require event persistence; thus this scenario will be ideal to showcase the transient input/output streaming features of MaxStream. Even though we have implemented LRB in its entirety [6], in this demonstration for practical reasons, we intend to show a smaller subset of it (a few minutes worth of input data, fixed load factor, Toll Notifications query). We are omitting the scenario figure for LRB due to space limitations; the general setup is similar to that of Figure 2.

Acknowledgements. This work has been supported in part by the following grants: Swiss NSF NCCR MICS 5005-67322, Swiss NSF ProDoc PDFMP2-122971/1, and ETH Zurich Enterprise Computing Center (ECC) SAP industrial partner grant DE-2008-022.

REFERENCES

[1] S. Zdonik, N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, M. Cherniack, U. Cetintemel, and R. Tibbetts, "Towards a Streaming SQL Standard," in *VLDB Conference*, 2008.

[2] M. J. Franklin, S. Krishnamurthy, N. Conway, A. Li, A. Russakovsky, and N. Thombre, "Continuous Analytics: Rethinking Query Processing in a Network-Effect World," in *CIDR Conference*, 2009.

[3] "SAP Real-World Awareness Forum," http://www.sap.com/about/company/research/irf/2009/endtoend/index.epx.

[4] "SAP Sales and Distribution (SD) Benchmark," http://www.sap.com/solutions/benchmark/sd.epx.

[5] A. Arasu, M. Cherniack, E. F. Galvez, D. Maier, A. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, "Linear Road: A Stream Data Management Benchmark," in *VLDB Conference*, 2004.

[6] I. Botan and et al., "Design and Implementation of the MaxStream Federated Stream Processing Architecture," ETH Zurich, Tech. Rep., June 2009, http://www.systems.ethz.ch/research/projects/maxstream/maxstream-federator-tr.pdf.

[7] "Coral8, Inc." http://www.coral8.com/.

[8] "StreamBase Systems, Inc." http://www.streambase.com/.

[9] "SAP MaxDB - The SAP Database," http://maxdb.sap.com/.

[10] I. Botan and et al., "Explaining the Execution Semantics of Sliding Window Queries over Data Streams: A Work in Progress Report," ETH Zurich, Tech. Rep., June 2009, http://www.systems.ethz.ch/research/projects/maxstream/maxstream-model-tr.pdf.

[11] A. Arasu, S. Babu, and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *VLDB Journal*, vol. 15, no. 2, 2006.

[12] I. Botan, Y. Cho, R. Derakhshan, N. Dindar, L. Haas, K. Kim, and N. Tatbul, "Federated Stream Processing Support for Real-Time Business Intelligence Applications," in *VLDB BIRTE Workshop*, 2009.