Federated Stream Processing Support for Real-Time Business Intelligence Applications

Irina Botan[1], Younggoo Cho[2], Roozbeh Derakhshan[1], Nihal Dindar[1], Laura Haas[3], Kihong Kim[2], and Nesime Tatbul[1]

[1] ETH Zurich, Switzerland
{irina.botan,droozbeh,dindarn,tatbul}@inf.ethz.ch
[2] SAP Labs, Korea
{young.goo.cho,ki.kim}@sap.com
[3] IBM Almaden Research Center, USA
laura@almaden.ibm.com

Abstract. In this paper, we describe the MaxStream federated stream processing architecture to support real-time business intelligence applications. MaxStream builds on and extends the SAP MaxDB relational database system in order to provide a federator over multiple underlying stream processing engines and databases. We show preliminary results on usefulness and performance of the MaxStream architecture on the SAP Sales and Distribution Benchmark.

Key words: stream processing, federated databases, data integration, heterogeneity, business intelligence

1 Introduction

Business intelligence (BI) is a broad term that encompasses a wide range of skills, processes, tools, technologies, applications, and practices for gathering, storing, analyzing, modeling, integrating, providing access to, and presenting information about a business or industry. The main goal is to support better decision-making for a business by leveraging data, or facts, and relationships between these facts. In addition to factual information providing historic and current views of various business operations, business intelligence technologies may also provide predictive views to further facilitate complex decision-making. In order to automate such decision-making and reporting processes, the business intelligence software market provides a wide spectrum of technologies from databases and online analytical processing (OLAP), data mining and data warehousing, to business performance management and predictive analysis [1, 2].

With the emergence of new applications and advances in other relevant technologies, the list above has recently been extended with another critical functionality: real-time business intelligence. Well-known examples of real-time BI include fraud detection, real-time marketing, inventory management, and supply-chain optimization. All of these applications require real-time or near real-time response to relevant business events as they occur.

In a recent article, Agrawal listed the following as the main enabling technologies for real-time BI: online updates, stream analysis and management, automated data integration, new declarative query languages for analytical processing, and scalable parallel query processing on large data volumes [3]. Each of these technologies would help to address one or more of the requirements raised by Schneider [4], who argues that real-time BI raises two critical requirements: (i) reducing latency, and (ii) providing rich contextual data that is directly actionable. Schneider's list of possible architectural alternatives to address these needs includes enterprise data warehouses, database federation systems, stream processing engines, or custom systems. While warehouses and database federation are good at providing reliable access to detailed and aggregated contextual data (requirement (i)), stream processing and custom systems are better for low-latency processing of large volumes of events and alerts (requirement (ii)). It is well-known that custom solutions are costly, and none of the remaining alternatives can effectively address both critical requirements on its own. Hence, we are at a stage where the critical requirements of real-time business intelligence applications and their enabling technologies are well understood, and there is an increasing need for system architectures and platforms to build on this understanding.

We believe that the MaxStream federated stream processing architecture is one such platform. MaxStream is designed to provide the necessary support for real-time (or near real-time) business intelligence applications [5, 6]. MaxStream combines capabilities for data management, data federation, and stream processing in a single system. It therefore allows seamless exploitation of the strengths of each architecture. Continuous queries can be passed to stream processing engines for immediate analysis of live data, and both input and output streams can be persisted for further historic analysis.

In this paper, we demonstrate that a federated stream processing system can be a useful platform for real-time BI. We describe the architecture and implementation of MaxStream (Section 3) and discuss how it can be used in real-time analytics scenarios (Section 4). In Section 5, we show that a federated stream processing system is practical: it can handle a real business intelligence workload while providing for extended stream processing capabilities with minimal overhead. Section 6 concludes with thoughts on future work. We begin with a brief description of related work in Section 2 below.

2 Related Work

Our work lies at the intersection of federated databases and stream processing systems. It is clearly also related to data management systems developed for business intelligence applications. In this section, we place MaxStream in the context of earlier work in each of these three research areas.

Federated Databases. From one perspective, MaxStream is a data integration system. It builds a federation over multiple stream processing engines (SPEs)

and databases. As such, it closely relates to the work on traditional database federation. In federated databases, middleware (typically an extension of a relational database management system) provides uniform access to a number of heterogeneous data sources (e.g., [7, 8, 9]). Similarly, we have built MaxStream as a middle layer between clients and underlying systems, extending a relational engine infrastructure. However, MaxStream has a fundamental difference in focus from database federation. Traditional database federations are motivated by the need to leave data *in situ*, and yet provide a unified view of the data. Exploiting data locality for efficient query processing is key. While they do need to bridge some heterogeneity in terms of capability, powerful data models and standards provide a framework for integration. In the stream world, by contrast, functional heterogeneity is a major issue due to a lack of standards and agreement on core features and semantics. Exploiting data locality is not as critical, as the data is streaming to begin with, and thus can be routed to an appropriate SPE. Integration of data is not the major focus. Thus the work we are doing benefits from the work on data federation, but has unique challenges to address in dealing with stream processing and the functional heterogeneity of SPEs.

Stream Processing. MaxStream acts as a stream processing system from the viewpoint of its client applications; however, it is not by itself a full-fledged SPE. Rather, the goal is to leverage the stream processing capabilities of the underlying SPEs as much as possible, while keeping the federation layer itself as lean as possible (i.e., with "just enough" streaming capability) in order to avoid complexity and overhead. Nevertheless, we can still compare MaxStream to existing work on stream processing in terms of the following two aspects. First, distributed SPEs such as Borealis [10] and System S [11] also employ multiple SPE nodes that share a continuous query processing workload; however, unlike MaxStream, autonomy and heterogeneity is not an issue for these systems as each processing node runs a controlled and identical SPE instance with the same capabilities and execution model semantics. Second, a few SPEs such as TelegraphCQ [12], Truviso [13], and DejaVu [14] have explored creating a streaming engine from a relational database engine. In doing so, these so-called "stream-relational" SPEs face many of the same issues and gain many of the same advantages as we do. However, our goals are different: MaxStream is focused on exploiting parts of relational technology that will facilitate our federation needs (e.g., streaming inputs and outputs, persistence, hybrid queries, model mapping, and so forth), rather than on building a complete SPE into the relational engine. In line with these goals, we have so far needed to introduce only two basic streaming capabilities into the SAP MaxDB relational engine - for being able to stream inputs and outputs through our federator, optionally with or without persistence (details are presented in Section 3.2). Stream-relational systems have similar mechanisms (and much more, such as sliding window processing, which we have not needed so far since we wanted to leverage the underlying SPEs for processing such advanced queries). For example, Truviso provides a "channel object" for explicitly creating a stream archive from a live stream. This is

similar to our persistent input streaming mechanism with one key difference. In MaxStream, if the persistence option is turned on, then the input stream is persisted and streamed as part of the same transaction, whereas in Truviso, these two parts seem to be decoupled from each other [13]. Also, to our knowledge, Truviso does not provide a special output streaming mechanism, whereas we have to do this in order to be able to continue supporting the pull-based client interfaces of SAP MaxDB.

Though not implemented on a relational database engine, we should also note that STREAM [15] bases its query model on relational semantics. In particular, our ISTREAM operator (Section 3.2) has been inspired by the relation-to-stream operator of the same name in STREAM [16], but there are some fundamental differences in its implementation and use, as we explain later in Section 3.2.

Business Intelligence. Business intelligence can be classified into three main types: strategic, tactical, and operational [17]. The first two deal with managing (long-term) business plans and goals based on historical data, while the last one focuses on managing and optimizing daily business operations. In operational BI, low-latency processing over business events as they happen is a critical need. "Real-time" or "right-time" BI also falls into this latter category. Though MaxStream can potentially serve all three forms of BI applications, we believe that with its emphasis on integrating SPEs and databases it can make a real contribution in the operational BI domain. Business intelligence is a key focus area for industry, with many of the recent advances being made by vendors. Industrial products such as SAP Business Objects [18] and IBM Cognos Now! [19] also employ some form of streaming dataflow engine to support operational BI. MaxStream's novelty lies in the way it can seamlessly integrate business events from multiple organizational units of an enterprise, each of which might be running a different specialized SPE for its local analytics tasks, as well as the tight integration with database processing. To the best of our knowledge, we are the first to explore this promising new direction of stream engine federation.

3 The MaxStream Federated Stream Processing System

MaxStream follows the typical architecture of a database federation system [9], providing a middleware layer between the application program and the data sources, and connecting to those sources via wrappers. Like a database federation engine, it provides a common programming interface and query language to the applications, and queries are translated into the languages supported by the underlying systems. MaxStream is built as an extension of a database federation engine, SAP MaxDB. We extend MaxStream with wrappers (called Data Agents in MaxDB) for stream processing engines, and two important operations (Section 3.2) that allow it to support stream queries. It inherits from SAP MaxDB the ability to federate additional databases, and it can also leverage MaxDB's local store.

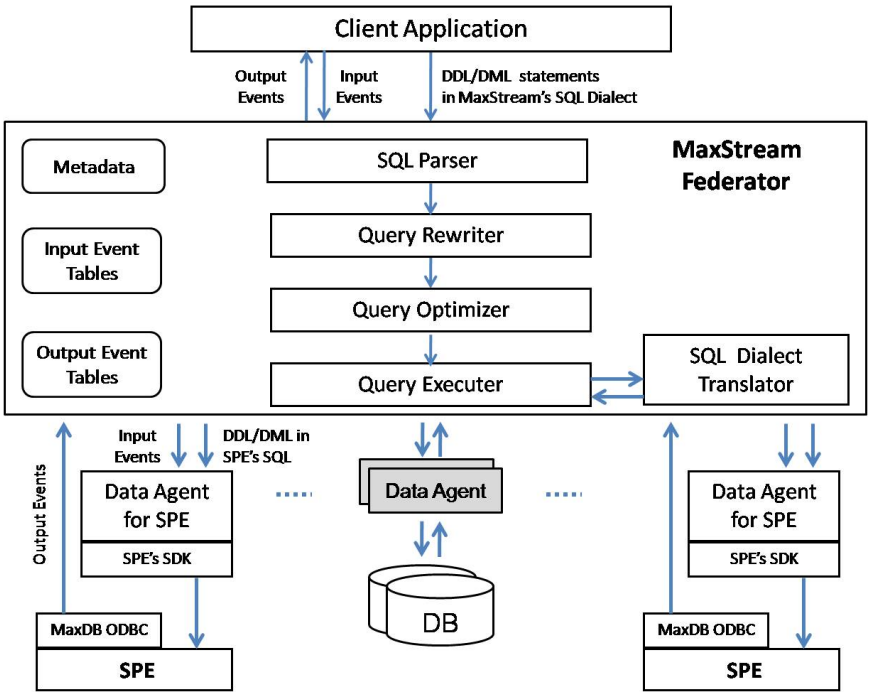

Fig. 1. MaxStream Architecture

In general, basing our stream federation engine on a database federator allows us to take advantage of all the basic relational functionality, including transactions, persistence, and the query processing architecture. Tools built for relational systems will work, and, because the system is SQL-based, existing relational applications can be easily ported to our platform and then extended as needed to work with streams. This is particularly convenient in a business intelligence environment where we want to leverage existing sophisticated tools [18, 20] to analyze data.

Today, MaxStream must push an entire stream query to an underlying SPE, though it can also take advantage of its local store for persistence, and to join the persisted data with streaming data. As we will show in more depth in Section 4, these are already useful capabilities for business intelligence applications. They provide for a seamless integration between the incoming streams and more static reference data, so that, for example, the streamed data can be easily persisted for regulatory compliance. In addition, they allow for an incoming stream of data to be enriched by joining it with local reference data before sending the stream on to an SPE for processing. However, we see much richer possibilities ahead; this minimal set of features is only a necessary first step that allows us to test the feasibility of our proposed architecture (Section 5).

3.1 Architecture

MaxStream's architecture is shown in Figure 1. It consists of a federation engine (MaxStream Federator) and a set of data agents (wrappers) for SPEs or databases. These interact as described below.

When an application submits a query to MaxStream, it is parsed, and, if there are no streaming features involved (*i.e.*, if the query is a standard relational query), it is passed to the usual rewrite engine and compiler. If it includes a stream query, however, that is routed today directly to the Query Executer and thence, to the SQL Dialect Translator. The translator creates an appropriate stream query in the language of the underlying engine, and returns it to the Executer, which sends it on to the Data Agent for the SPE. Data Agents handle registration of queries to the SPE; they also stream input events to the SPE for processing. The output stream from the SPE is written directly back to MaxStream through an ODBC connection from the SPE.

Using MaxStream in an application requires a few simple steps. We describe these briefly via a simple example below.

1. **Create the stream**. If the stream has not been previously created, it must be generated and also registered with MaxStream. MaxStream allows inserts on a base table to be streamed out to the SPE. This is needed where the business is receiving a lot of event data and wants to process it as it comes in, leveraging a stream engine. For example, suppose in a Call Center every call is captured with some metadata in a single table, `Calls`. The company realizes that they can provide better service if they can analyze the call patterns in real-time, so they decide to stream the data to an SPE. This would be done as follows:

```
CREATE STREAM CallStream ...;
INSERT INTO STREAM CallStream
    SELECT ...
    FROM   ISTREAM(Calls);
```

   The first statement tells MaxStream about the stream, and the second turns inserts into the existing table into a stream. We explain this mechanism in Section 3.2 below.
2. **Create a table for the output stream**. The SPE needs some place to put the results of its analysis, so we must create a table (which can be either persistent or transient) to store the output stream. This allows the SPE to use standard INSERT statements and an ODBC connection to get the output back to MaxStream. This can be done with a standard CREATE TABLE statement, e.g.:

```
CREATE TABLE CallAnalysis ...;
```

3. **Create a query to push to the SPE**. Now that there is a stream, and a place to put the output, we need to tell the SPE what to do. For example, we might issue a query such as:

```
INSERT INTO TABLE CallAnalysis
    SELECT Region, COUNT(*) as Cnt, AVG(WaitTime) as AvgWait,
           AVG(Duration) as CallLength
    FROM   CallStream
    GROUP BY Region
    KEEP 1 HOUR;
```

This continuous query tracks the number of calls, the average wait time for an agent, and the average length of calls by region, on an hourly basis, allowing the company to find hot spots where perhaps they need to assign more staff, or areas where the call center staff are not efficient enough. MaxStream will push this query to the SPE, after any necessary translations. The SPE will insert results back into CallAnalysis, using the ODBC connection that is set up when the Data Agent is instantiated.

4. **Set up monitoring for the output**. Finally, we will want to do something with the results of the analysis, so we need a way to find out when there is new data in the output table. MaxStream provides a way of monitoring changes to a base table. This mechanism is invoked by means of a hint on a SELECT statement, as follows:

```
SELECT *
FROM /*+Event*/  CallAnalysis
WHERE AvgLength > 5;
```

In this example, the predicate might be used to identify regions where the average call length was beyond some threshold (*i.e.*, where the call center is not being sufficiently efficient). The monitoring mechanism is also described in Section 3.2.

In summary, we have extended a typical database federation engine, SAP MaxDB Federator, with the concept of a stream, and the ability to push streaming queries to an SPE. MaxStream has the ability to create streams from inserts to a base table, and to monitor base tables which are receiving the results of analysis from an SPE. We explain how both of these novel features work in the next subsection.

We have prototyped these features with the goal of understanding the feasibility of this approach. In particular, we wished to see whether we could provide sufficient performance to handle the demands of our envisioned business intelligence applications, despite the insertion of a federation layer between the application and the streaming engine(s). We also wanted to explore the usefulness of integrating persistence and streaming capabilities as described above. In pursuit of these two goals, we have focused to date on getting the architecture running, with a first Data Agent for a popular SPE, SPE X. We will show in Section 5 evidence that we have met these goals, and our future work will focus on extending our prototype with additional Data Agents and richer query processing features as needed for a broad range of business intelligence applications.

3.2 Two Key Building Blocks

In addition to changes to the SAP MaxDB query compiler to handle continuous queries and additional Data Agents to connect to SPEs, we made two substantive additions to the MaxDB core functionality to enable it to play the role of the federator in a streaming environment. We added capabilities for creating or passing streams through the federator to the SPE, so that we could provide stream inputs. We also added a *monitoring select* capability, to permit efficient monitoring of stream outputs. In the rest of this section, we describe these two capabilities.

Streaming Inputs

To have a continuous query on a stream, you must first have a stream. MaxStream can provide a stream to an SPE in two ways. If the data does not need to be persisted, an in-memory tuple queue mechanism is used to move streamed tuples quickly through the federator. In this case, a DDL statement would define the stream to the federator, and the application generating the stream can write directly to the stream. This mechanism is particularly useful if the application has relatively high data rates and/or relatively strict latency requirements. For example, to monitor and maintain service level agreements, an SPE needs reports on response times and latencies for a stream of jobs of different priorities. The application generating those reports could proceed as follows:

```
CREATE STREAM JobStream ...;
INSERT INTO STREAM JobStream
       VALUES(:job#, :timein, :timedone, ...);
```

If the streamed data needs to be persisted, it is first stored in a base table, and then a new operator, ISTREAM, turns inserts on that table into a stream for consumption by the SPE. The ISTREAM operator, first suggested in [16], is a relation-to-stream operator. At any point in time τ, the value of `ISTREAM(T)` on a table `T` is the set of records that are in `T` at time τ and that were not in `T` at time $\tau - 1$. If we were to persist the `JobStream` created above, we would need two INSERT statements, one to store the incoming tuples, and one to stream them out, as follows:

```
INSERT INTO Jobs VALUES (:job#, :timein, :timedone, ...);
INSERT INTO STREAM JobStream
    SELECT *
    FROM   ISTREAM(Jobs);
```

where `Jobs` is the base table created to hold the persistent tuples.

We would like to note that, although we are not the first ones to propose the ISTREAM operator, we have implemented and used it in a completely new way. In the STREAM system, ISTREAM is logically a relation-to-stream operator, but it was never implemented within the context of a relational database engine. Also, it was only used in the SELECT clause of a CQL query to indicate that the

result of the query should be of streaming type [16]. In MaxStream, ISTREAM is truly a relation-to-stream operator and it is used in the FROM clause to create a stream out of a given relational table. Furthermore, we have implemented ISTREAM directly within the context of the SAP MaxDB relational database.

Loosely speaking, our ISTREAM implementation works as follows. A transaction inserts tuples into a base table. When the transaction commits the new tuples are copied to an associated side-table, and a corresponding "stream job" is created and added to a queue of stream jobs, with a timestamp based on the logical commit time. A separate "stream thread" processes the side-table. The stream thread has a list of compiled query plans, each corresponding to one of the potentially multiple streaming inserts that select from ISTREAM on that table. It executes those plans on each stream job in the queue, sending them the relevant tuples out of the newly inserted set. For a more detailed description of our ISTREAM implementation, please see our technical report [5].

Monitoring Select

We have discussed how to get streamed data in from applications using tuple queues or the ISTREAM operator. Likewise, we need to be able to get result streams from the SPE, through MaxStream and out to the clients. However, MaxStream applications are fundamentally database applications, and database applications are pull-based: they ask queries and pull on a cursor to get results. SPEs, by contrast, are push-based. We explored several mechanisms for getting the results back to the clients, including continual polling, periodic selects, database triggers, and adding a subscription mechanism to the client interface, but found none of them adequate for our purposes [5]. All either scaled badly as the number of streams and queries on them grew, or were inefficient.

To better bridge between our pull-based clients and push-based streaming engines, we have added a new capability, *monitoring select*. Monitoring select is inspired by existing industry blocking selects [21, 22]. A monitoring SELECT statement is different than a typical select in that it only returns new rows, and when no new rows are found, it blocks until new rows appear. Thus, it saves the client from periodically polling. In essence, it emulates a subscription interface, without requiring substantial changes to the clients. Monitoring select is indicated by hints as illustrated in Section 3.1 above.

Under the covers this operator works a bit like ISTREAM, with a side-table for recently arrived tuples, and a waiting list or queue of jobs to be processed against them. The jobs for monitoring select are blocked queries, not stream inserts as with ISTREAM. Briefly, for each base table being monitored, there is a waiting list with the set of tasks monitoring that table, and a timestamp for each. As an application tries to "pull" data from the table, new rows that have come in since the last time it asked, if any, are returned, and if there are none, it is blocked. As new rows are inserted to the base table, they are also written to a side-table, and on commit the insert transaction wakes up any relevant select tasks in the waiting list. These tasks check whether the new rows

match their predicates (if any). If they do, the new rows are returned, otherwise the task is again suspended. Of course, there are several complexities, including possibilities for concurrency problems which must be handled; details are given in our technical report [5].

Note that monitoring select can be used with either permanent or transient, in-memory tables. The use of transient tables gets the stream output to the application for handling. Use of a permanent base table also persists a copy of the streamed result. While somewhat higher overhead, this can be particularly useful for business intelligence applications that want to save the results of a stream query for further analysis. For example, given an operational application that monitors service level agreements, we might persist the results of the stream query that detects violations, for further analysis into root cause, trends, and so on.

3.3 Hybrid Queries: Using Persistence with Streams

So far, we have shown how MaxStream can handle streams and queries over streams. But with business intelligence scenarios that will rarely be sufficient. Warehouses typically have fact tables and dimension tables. Fact tables tend to change more frequently than the dimension tables, and are usually much larger. For example, the fact table might capture sales transactions, while the dimension tables add information about the products sold, the store at which the transaction occurred, etc. Clearly, new transactions occur much more frequently than the addition of a new product or a new store. Business intelligence applications combine the fact and dimension tables to explore the data in different ways. Similarly, to do business intelligence analytics over a stream, the rapidly changing data in the stream must be combined with descriptive data in other tables. For many scenarios, in fact, the stream can be viewed as playing the role of the fact table. Fortunately, MaxStream provides the ability to join this frequently changing data with the more static dimension or descriptive data that might be persisted either in the local store or in one of the federated databases.

Returning to our call center scenario, remember that metadata about every call was being recorded, and streamed to the SPE, which was returning some statistics for various regions. We can easily imagine how persistent data might supplement both the input stream (the raw calls information) and the output stream (the grouped statistical information) for different purposes. For example, suppose the actual incoming data from the call center has an operator code, an arrival time for the call, a start time at which the operator began serving the customer, and an end time for the call. In other words, refining the example from Section 3.1 above, suppose the `Calls` table is defined as:

```
CREATE TABLE Calls
    (Opcode integer, ArrivalTime Timestamp,
     StartTime Timestamp, EndTime Timestamp);
```

Note that there is no region field in the table. Hence, to be able to do the aggregation on region, our SPE will need some additional information, namely,

the mapping from operator codes to regions. That can be supplied by joining the ISTREAM'ed `Calls` table to a persistent table, `OperatorsbyRegion` as follows:

```
INSERT INTO STREAM CallStream
    SELECT o.RegionNm as Region,
           c.StartTime-c.ArrivalTime as WaitTime,
           c.EndTime-c.StartTime as Duration
    FROM   ISTREAM(Calls) c, OperatorsbyRegion o
    WHERE  c.Opcode = o.Operator;
```

This join injects the additional information needed into the stream passed to the SPE, so that the query asked in Section 3.1 can be processed by the SPE. As a reminder, the query is repeated here:

```
INSERT INTO TABLE CallAnalysis
    SELECT Region, COUNT(*) as Cnt, AVG(WaitTime) as AvgWait,
           AVG(Duration) as CallLength
    FROM   CallStream
    GROUP BY Region
    KEEP 1 HOUR;
```

This example demonstrates both the usefulness and the simplicity of being able to join an input stream to persistent data. Note that since MaxStream is also a database federation engine by heritage, the table `OperatorsbyRegion` could be stored locally, or, equally easily, could be stored at any federated database.

In a similar fashion, it is possible, and useful, to join an output stream with additional data. This might allow further analytics on the stream, or just create richer context for a report. For example, perhaps an analyst might want to understand why certain regions have longer wait times than others. The `Regions` table might have important information about each region such as the number of operators, training method for operators, hours of training, number of supervisors – and so on. Rather than just getting back the rolling statistics by region delivered by the SPE above, a more interesting query for the analyst might be the following:

```
SELECT a.Region, a.AvgWait, a.AvgDuration, r.NOps, r.Training, ...
FROM /* +Event */ CallAnalysis a, Regions r
WHERE AvgWait > 10
  AND a.Region = r.RegName;
```

This lets the analyst see the information about average call waiting time and duration in the context of additional information about that region, for those regions which have long wait times (10 minutes or more!), potentially helping them diagnose the problem at these centers. Again, the query is specified as a simple join, and clearly, more complex queries with additional grouping and analysis would also be possible. Again, the static tables could be local or in a federated database.

We would like to note that both forms of hybrid queries illustrated above (i.e., enriching the input stream with static information as well as doing so for

the output stream) merely leverage our two key building blocks, ISTREAM and monitoring select. We did not have to add any other sophisticated stream processing capability into the federator. This fits well with our design goal about keeping the federator lean [1].

This section has described the overall architecture of MaxStream, and demonstrated features of MaxStream that make it ideally suited for use in business intelligence scenarios. MaxStream allows us to feed standard business intelligence tools and applications with real-time information, and complement or extend that information with information from static tables. It also allows us to leverage the power of a stream processing engine to do on-the-fly analysis of rapidly growing event streams, persisting as desired both input and output streams for further, richer analytics. We believe these are critical features for real-time business intelligence applications, and we elaborate on this point in the next two sections.

4 Using MaxStream in Real-Time BI Scenarios

Section 3 provided an overview of MaxStream's architecture, illustrating the concepts with some small examples with an operational BI feel. In this section, we describe in more depth a few concrete application scenarios that are inspired by real business intelligence use cases. Our goal is to show the role MaxStream might play in supporting these diverse business scenarios.

4.1 Reducing Latency in Event-Driven Business Intelligence

Consider a global mail delivery company (e.g., FedEx, UPS, DHL, etc.) with company locations in many cities in various countries. Packages to be delivered must first be received. This can be done in several ways, for example, via staffed drop-off stores, self-service drop boxes, or arranged pick-up from the customer's address. Next, the collected packages must be distributed towards their destinations in a multi-hop fashion, travelling through a series of company facilities, and finally being delivered to their destination by local staff members driving trucks. Each package to be shipped comes with a service option and at a corresponding cost. For example, same-day delivery is the fastest delivery option (guaranteed delivery within the same day that the package is received) and therefore is the most expensive. This company must continuously monitor its package deliveries all around the globe to make sure that the delivery guarantees are met, as service violations are not acceptable and may cause financial penalties and customer dissatisfaction.

The whole distribution process is quite complex and certainly requires carefully creating an optimized delivery schedule in advance. However, due to situations that cannot be foreseen in advance, companies also like to monitor what

[1] The underlying mechanism that facilitates this is the fact that the stream is materialized in a table (on disk or in memory) before the join operation takes place.

happens in reality. Thus, during the actual distribution, many business events indicating changing package locations are generated as part of daily operations in this company. Furthermore, processing these events with low-latency is key to ensuring that timely package delivery can be achieved. For example, one of the business analysts that work for our mail delivery company would like to create analytic queries to watch the deliveries in the last 15 minutes and see how they compare to the target delivery times. S/he can then use this information to make on-the-fly business decisions such as routing dispatch units in different ways to avoid potential problems or to help solve existing problems.

In this scenario, reducing latency of continuous analytic queries is key. It would take too long to dump all delivery events into a data warehouse. Instead, they should be processed in a streaming fashion. Furthermore, comparing live delivery time values to target values requires a hybrid query between new event streams and static reference data. Last but not least, the company may be using several different SPE instances at different locations, whose results must be aggregated to be able to make more informed decisions about dispatching resources. This scenario is similar in flavor to the Service Level Agreement monitoring example used in Section 3.2. MaxStream can support each of these needs, and in addition, could persist the results of the operational analysis so that further analytics could be applied later – to detect repeated trouble spots, for example, and do more strategic levels of planning.

4.2 Persistent Events in Supply-Chain Monitoring

Supply-chain monitoring involves keeping track of where products appear in time as they move from initial producers to final consumers. Product locations are typically kept track of using auto-id technologies (e.g., RFID). Each event indicates a product-id, event-type, product-class, event-location, and event-time. Events can be of two types: issue or receipt. One might like to monitor various complex events over these RFID events:

- For each product class, what is the average cycle time (i.e., the difference between time of receipt and time of issue)?
- Show the products that arrived late, together with the name of the carrier company and from/to location.
- Out of stock: Continuously check the available quantity of every sold item in the back stock and raise an alert whenever the sum of the quantity drops below the predefined threshold.

In this application, latency is not a major issue, as the sort of analysis to be done is mostly strategic or tactical. On the other hand, events are arriving frequently, and all events must be persisted for durability. Other scenarios with this flavor include the Call Center application used as an example in Section 3.1, where the main goal was to understand past performance to make tactical or strategic improvements. As we showed in that section, these types of applications are easily supported using MaxStream's features.

4.3 Other Real-time BI Applications

There are many other BI scenarios that require real-time event processing. Most of them fall into one or the other of the two patterns sketched above. For example, fraud detection is a common desire for financial and credit card companies. In this case, reducing latency is critical, to be able to catch cases of fraud as they happen. The event information will likely be the basic information about the transaction, e.g., for credit cards, who is charging what amount for what items to which card. Additional information about the customer and their buying patterns will need to be included for the analysis. Again, MaxStream supports this case by allowing the transaction information to be joined with the contextual information as it streams through the system. On the other hand, quality management for manufacturing has more of a feel of the supply chain management application: many events flow through the system as the statistics on production flow in. Some are analyzed in real-time (for instance, to find real outliers), but for the most part the goal is to capture the information for later strategic analysis of yield, repair records, returns, and so on.

With large organizations, these scenarios become further complicated by the likelihood that there is not a single stream processing engine nor a single source of data for correlation. Instead, these organizations often have several databases, and we may expect, several heterogeneous SPEs (e.g., to be able to exploit specialized capabilities of different SPEs). In this case, MaxStream's ability to bridge across heterogeneous SPEs and data sources will be invaluable. In [5] we give an example of a simple sales order scenario in which the corporate headquarters gathers statistics for a map leveraging the SPEs at each of the worldwide sites. We expect that as more businesses adopt stream processing engines, this case will become increasingly common, and MaxStream is prepared to handle it.

5 Feasibility Study

We have discussed MaxSteam's architecture, and illustrated how it might be used in real-time business intelligence scenarios. In this section we will show that in fact, it is feasible to use MaxStream in these ways. In particular, we will demonstrate that performance is not an issue; the overheads introduced by MaxStream are negligible for business intelligence applications. We will also show that the types of statements described in Section 3 above really do work, and can, in fact, complement and extend an existing business applications.

To accomplish these goals, we use a simplified scenario from the SAP SD Benchmark, a standard server benchmark that simulates a typical business application [23]. The benchmark includes a variety of user transactions to create orders and delivery documents, check on orders, issue an invoice, and look at customer order history. Each transaction consists of several dialog steps with ten seconds of think-time for each. The benchmark measures throughput in the number of processed dialog steps per minute (SAPs).

SAP SD is not a stream processing benchmark, but it does represent a typical operational scenario that could be enhanced by the ability to do some real-time business intelligence on the data. Note that for this application, all events (orders, deliveries, etc) must be persistently stored, so large data volumes will accumulate over time. We added to this scenario the ability to do analytics over the incoming data. In particular, we compared the original SD benchmark with two variations. In the first variation, we streamed the incoming orders to the SPE for some real-time analysis (SD + ISTREAM). In the second, we monitored sales, looking for unusually large orders (SD + Monitoring Select). The processing time at the SPE was not measured; we only compare the times in MaxStream to see the effects of our new features (i.e., ISTREAM and monitoring select).

Our system was configured with the MaxStream server on a 4-way quad-core Intel Tigerton 2.93 GHz with 128GB memory. Sixteen application server blades were used; each blade was a 2-way quad-core Intel Clovertown 2.33GHz with 16GB memory. The stream processing engine (SPE X) ran on a 4-way dual-core AMD Opteron 2.2GHz with 64GB memory. All systems were running Linux. This configuration enabled us to handle 16,000 users.

For the SD + ISTREAM scenario, we used the following continuous INSERT statement to forward all orders to the SPE for processing:

```
INSERT INTO STREAM OrderStream
    SELECT A.MANDT, A.VBELN, A.NETWR,
           B.POSNR, B.MATNR, B.ZMENG
    FROM   ISTREAM(VBAK) A, VBAP B
    WHERE  A.MANDT = B.MANDT
      AND  A.VBELN = B.VBELN;
```

As new orders are inserted into the VBAK table, they are joined with VBAP and added to OrderStream for processing by the SPE. VBAK stores order data, and VBAP holds the line item information for each order (so these tables are analogous to ORDERS and LINEITEMS in TPC-H [24]). With this scenario we can observe the overhead in sending the operational data to the SPE for analytic processing. In the SD + Monitoring Select scenario, we monitor for big sales orders (totalling over 95 items). The query looks as follows:

```
SELECT A.MANDT, A.VBELN, B.KWMENG
FROM   /*+ EVENT */ VBAK A, VBAP B
WHERE  A.NETWR > 95
  AND  A.MANDT=B.MANDT
  AND  A.VBELN=B.VBELN;
```

The input data was modified so that about 1% of queries were big sales orders.

The results of our experiment are shown in Table 1. The maximum throughput possible with 16,000 users is 96,000 SAPs (dialog steps/minute). In all three configurations, we are able to achieve close to the maximum, showing that there is little overhead for forwarding the orders to the SPE or for monitoring orders with a monitoring select. Since the users order, in aggregate, 533 line items per second, 533 events are passed to the SPE every second, and only 0.8% more

	SDB	SDB + ISTREAM	SDB + Monitoring Select
# of SD users	16,000	16,000	16,000
Throughput (SAPs)	95,910	95,910	95,846
Dialog response time (ms)	13	13	13
% DB server CPU utilization	49.8%	50.6%	50.1%

Table 1. MaxStream Performance Results on SAP SD Benchmark

CPU is used than in the original benchmark. Monitoring Select does reduce the throughput slightly, but the penalty is small. Thus, using MaxStream is a realistic approach to adding business intelligence capabilities in an operational scenario.

Note that the main purpose of this feasibility study was to show the utility of MaxStream in handling realistic operational BI scenarios as well as to see how much performance overhead was introduced by using MaxStream's basic streaming building blocks in exchange for that benefit. We have also done an experimental study that directly focuses on how MaxStream scales with increasing input load based on the Linear Road Benchmark [25]. The results of that study can be found in our technical report [5].

6 Conclusions and Future Directions

Real-time business intelligence is becoming a pressing need for many companies. Real-time BI offers businesses the ability to optimize their processes, and to get just a little ahead of the competition, by leveraging their valuable information assets. But delivering real-time BI requires new platforms which offer the low latencies of stream processing, the support for analytics of data warehouses, and the flexible, dynamic access to data of data federation engines.

In this paper, we have described MaxStream, a stream federation engine that provides a promising approach to this challenge. MaxStream provides access to heterogeneous stream processing engines, seamlessly integrated with a persistent database and data federation capability. We described the MaxStream architecture, including the extensions we made to allow data to be streamed through the engine to an SPE, and back through MaxStream to a client. Through several scenarios, we illustrated how these features could be leveraged for real-time business intelligence applications. Finally, we demonstrated that it is feasible to leverage these extensions to give operational applications a real-time BI capability, without incurring significant penalties in performance.

MaxStream is currently in its infancy. As we look ahead, several important areas of research loom. As we build more data agents for MaxStream, we will need to leverage a common model and language for the federation layer, but with the heterogeneity of current SPEs, choosing or creating such a model is a substantial challenge. Work is underway on this front, and our initial results are documented in [26]. Today, MaxStream only pushes down entire queries to the SPEs; we plan to relax this requirement. This will open up many avenues for research. For what

workloads and what types of processing will splitting work across multiple SPEs make sense? How can we take best advantage of underlying SPEs' capabilities? Is there a sensible notion of optimization, and what are the choices and metrics we need to consider? We plan to build a real business intelligence scenario to understand what features are most necessary and to prove their value. We believe that Maxstream can already play an important role in the next generation of business intelligence systems, and are eager to see how much it can achieve.

Acknowledgements. We would like to thank Chan Young Kwon for his help with the SAP SD Benchmark measurements, and MaxStream team members for their contributions. 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. Bussler, C., Castellanos, M., Dayal, U., Navathe, S.B., eds.: BIRTE'06, Seoul, Korea, Springer (September 2006)
2. Castellanos, M., Dayal, U., Sellis, T., eds.: BIRTE'08, Auckland, New Zealand (August 2008)
3. Agrawal, D.: The Reality of Real-time Business Intelligence. In: BIRTE'08, Auckland, New Zealand (August 2008)
4. Schneider, D.A.: Practical Considerations for Real-Time Business Intelligence. In: VLDB/BIRTE Workshop, Seoul, Korea (September 2006)
5. Botan, I., Cho, Y., Derakhshan, R., Dindar, N., Haas, L., Kim, K., Lee, C., Mundada, G., Shan, M.C., Tatbul, N., Yan, Y., Yun, B., Zhang, J.: Design and Implementation of the MaxStream Federated Stream Processing Architecture. Technical report, ETH Zurich, Computer Science (June 2009) http://www.systems.ethz.ch/research/projects/maxstream/maxstream-federator-tr.pdf.
6. Botan, I., Cho, Y., Derakhshan, R., Dindar, N., Haas, L., Kim, K., Lee, C., Mundada, G., Shan, M., Tatbul, N., Yan, Y., Yun, B., Zhang, J.: A Demonstration of the MaxStream Federated Stream Processing Architecture (Demonstration). In: IEEE International Conference on Data Engineering (ICDE'10), Long Beach, CA (March 2010)
7. Heimbigner, D., McLeod, D.: A federated architecture for information management. ACM Transactions on Information Systems **3**(3) (1985)
8. Sheth, A.P., Larsen, J.A.: Federated database systems for managing distributed, heterogeneous, and autonomous databases. ACM Computing Surveys **22**(3) (1990)
9. Haas, L.M., Lin, E.T., Roth, M.T.: Data Integration Through Database Federation. IBM Systems Journal **41**(4) (2002)
10. Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.: The Design of the Borealis Stream Processing Engine. In: CIDR Conference, Asilomar, CA (January 2005)

11. Amini, L., Andrade, H., Eskesen, F., King, R., Park, Y., Selo, P., Venkatramani, C.: The Stream Processing Core. Technical Report RSC 23798, IBM T. J. Watson Research Center (November 2005)
12. Chandrasekaran, S., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: CIDR Conference, Asilomar, CA (January 2003)
13. Franklin, M.J., Krishnamurthy, S., Conway, N., Li, A., Russakovsky, A., Thombre, N.: Continuous Analytics: Rethinking Query Processing in a Network-Effect World. In: CIDR Conference, Asilomar, CA (January 2009)
14. Dindar, N., Güç, B., Lau, P., Özal, A., Soner, M., Tatbul, N.: DejaVu: Declarative Pattern Matching over Live and Archived Streams of Events (Demonstration). In: ACM SIGMOD, Providence, RI (June 2009)
15. Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G., Olston, C., Rosenstein, J., Varma, R.: Query Processing, Approximation, and Resource Management in a Data Stream Management System. In: CIDR Conference, Asilomar, CA (January 2003)
16. Arasu, A., Babu, S., Widom, J.: The CQL Continuous Query Language: Semantic Foundations and Query Execution. VLDB Journal **15**(2) (2006)
17. Azvine, B., Cui, Z., Nauck, D.D.: Towards Real-Time Business Intelligence. BT Technology Journal **23**(3) (2005)
18. Business Objects: SAP Business Objects http://www.sap.com/solutions/ sapbusinessobjects/index.epx.
19. Cognos: IBM Cognos Now! http://www.cognos.com/products/now/.
20. Cognos: IBM Cognos http://www.cognos.com/.
21. ANT: ANTs Data Server V 3.60 - Programmer's Guide http://www.ants.com/.
22. Oracle: Oracle8 Application Developer's Guide Release 8.0 http://download.oracle.com/docs/cd/A58617_01/server.804/a58241/ch12.htm.
23. SAP: SAP Sales and Distribution Benchmark http://www.sap.com/solutions/benchmark/sd.epx.
24. TPC: TPC-H Benchmark http://www.tpc.org/tpch/.
25. Arasu, A., Cherniack, M., Galvez, E.F., Maier, D., Maskey, A., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear Road: A Stream Data Management Benchmark. In: VLDB Conference, Toronto, Canada (September 2004)
26. Botan, I., Derakhshan, R., Dindar, N., Haas, L., Miller, R., Tatbul, N.: Explaining the Execution Semantics of Sliding Window Queries over Data Streams: A Work in Progress Report. Technical report, ETH Zurich, Computer Science (June 2009) http://www.systems.ethz.ch/research/projects/maxstream/maxstream-model-tr.pdf.