QStream: Deterministic Querying of Data Streams

Sven Schmidt, Henrike Berthold and Wolfgang Lehner

Dresden University of Technology
Database Technology Group
dbgroup@mail.inf.tu-dresden.de

Abstract

Current developments in processing data streams are based on the best-effort principle and therefore not adequate for many application areas. When sensor data is gathered by interface hardware and is used for triggering data-dependent actions, the data has to be queried and processed not only in an efficient but also in a deterministic way. Our streaming system prototype embodies novel data processing techniques. It is based on an operator component model and runs on top of a real-time capable environment. This enables us to provide real Quality-of-Service for data stream queries.

1 Motivation

In recent years, the amount of data delivered by a variety of sensors has grown significantly. There are a lot of examples of sensors being installed to continuously measure physical properties and to send these values to back-end systems for further processing.

Once a sensor data stream is connected to a stream processing system, the continuous/standing queries have to be evaluated on that data. Based on the query results, fast reactions may be necessary. To give an example, think of a meteorological system where acquired data such as temperature, humidity, atmospheric pressure, brightness and rainfall etc. form the basis for statistical analyses on the one hand and for sophisticated forecast and reaction on the other hand. The latter includes the detection of abnormal situations identified by characteristic constellations of the sensor input values.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

In industrial application areas, sensors are used in production processes to observe the progress of single manufacturing steps. For example, the positional measures of a complex robotics form a data stream and are used for position tracking. Once the robotics reaches its final destination or is intended to stop immediately in case of an emergency, the data stream processing system has to react within a given time limit. Thus, based on the sensor data, the processes can be controlled and successfully completed.

Within a best-effort streaming system (e.g. [1, 2]), no one can give any guarantees for the exact evaluation of queries and for the amount of time elapsing between a characteristic constellation of input values and the reaction of the system. Within our real-time streaming project (QStream) we

- describe a set of stream operators plus their functional and non-functional properties,
- rely on a model for resource description in streaming environments,
- define Quality-of-Service parameters and enable the user to negotiate them, and
- implement this conceptual work on top of a real-time capable system environment.

The goal of QStream is to evaluate queries on data streams in a deterministic way: Once a query is registered and accepted by the system, it will be successfully completed independently from any system load or data stream load without compromising the negotiated quality constraints. These quality constraints mainly consist of result precision and time delay of the query evaluation system.

2 Operator Component Model

In QStream, after query parsing, a stream query is represented by a network of operators (figure 1). This network deals with a number of sensor originated input streams and produces one single output stream for each registered query.

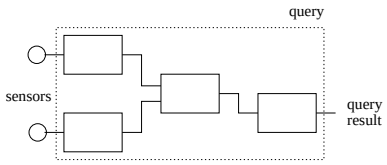

Figure 1: operator network

Thereby an operator is a small work unit which reads tuples from its input buffer, does some processing work, and writes the resulting tuples to an output buffer. The basic work includes *one* data read, *one* data processing and *one* data write. We will call it a *run*. The amount of data such an operator processes within a certain (amount of) time is reflected by the data rate. The initial data rate is determined by the sensor data sources. The relationship between input and output data rate of an operator belongs to the operator description.

Within QStream we do not contribute a new data model or completely new operators. Instead we propose a novel technology for their implementation. For sake of completeness, the operators currently supported are listed below:

- Filter: A filter drops tuples which do not conform to a given predicate. The resulting data rate depends on the input data rate and on the selectivity of the applied predicate, which has to be derived from statistical information, such as value distributions, about the input stream.
- Sample: The sample operator randomly discards tuples with a specified sampling rate. The data rate of the output stream can be calculated from the data rate of the input stream and the sampling rate.
- Aggregation: An aggregation operator is either a summation or the calculation of an average on a specified attribute. The output data rate decreases with a larger aggregation group size.
- Grouping and Aggregation: The grouping-and-aggregation operator performs a grouping on an ordering attribute (such as the minutes of a time attribute with precision of seconds) and an aggregation (SUM or AVG) on another attribute. As with aggregation, the output data rate depends on the input data rate and on the average group size.
- Map: The map operator applies a scalar function locally to a tuple. It calculates for example a new value from two tuple values and adds it to the tuple. The data rates of both streams are identical. Only the tuple size may change.
- Pairing: The pairing operator performs a join operation on an ordered attribute for example on a global time. It consists of two input streams and one output stream. The data rate of the output stream is identical to the data rates of the input streams.

Operator interfaces

One of our basic ideas is to run each operator of the operator network independently within the operating system environment. Thus, an operator has a control and a data exchange interface (figure 2). Through the control interface, each operator is parameterized before the data stream processing starts; it is continuously monitored; and (if necessary) it is adjusted during runtime. The data exchange interface is responsible for transferring the tuples from operator to operator.

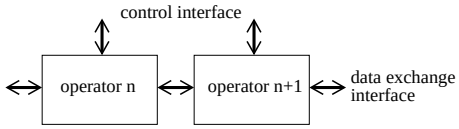

Figure 2: operator interfaces

3 Controlling the Operator Network

The QStream control application first builds an operator network from a stream query. Thereafter, it calculates memory requirements and processing times for each single operator based on the data stream characteristics and on the users negotiated quality constraints. When the network construction and resource calculation process is complete, the control application tries to schedule the values with the operating system's resource manager and, thus, performs an admission control. This implies that the number of accepted stream queries depends on the characteristics of the data stream (data rate, bursts) and on the complexity of the queries, and that it is limited by the overall resources of the QStream system. It is important to notice that once a query has passed the admission control, a successful evaluation of the data streams within the negotiated quality constraints is guaranteed.

Scheduling

Scheduling covers issues about how often and how long an operator is supposed to perform its work. We propose the concept of an 'optimal processing speed' of the operator network: If the operators run too slow (perform a run very rarely), the quality constraints regarding query delay and precision may not be fulfilled. On the other hand it is inadequate to speed up the operator network (perform the runs very frequently) or pass through the sensors high data rates if the user is only interested in approximate values and

if a time limit for answering the queries does not exist. This would result in slowing down concurrently running processes.

For the deterministic behavior regarding the processing times we rely on a real-time operating system (RTOS) environment. Firstly, this enables us to schedule precisely, and secondly, we are able to encapsulate the QStream operators from applications with no real-time requirements.

All of the QStream operators are periodic processes. Thus, an operator periodically performs some work. Moreover, the RTOS signals the operator the start of the next run.

Periods

Based on the data rate of the sensors and based on the data rate requirements/quality constraints at the output of the operator network, the period length P_O of each operator is determined. Therefore, the number of tuples which are to be produced during a single run (given by the operator description) is taken into account. The resulting periods are scheduled for the QStream operators. The RTOS has the responsibility to signal each period start to the specific operator. The operator itself has to ensure that it completes its work during the operator processing time t_O, whereas $t_O << P_O$.

The processing time t_O is the time required by a virtually isolated operator for a single operation. The simplest method for obtaining t_O is to measure it on a reference system with only one operator running at a time.

In general, multiple operators O_1, ..., O_n form an operator network and, thus, work in parallel (all in real-time mode). The time needed to run all n operators is the sum of the single operators' processing times. Therefore, the operator processing times t_O have to be normalized ($\hat{t}_O$) to a common period length $\hat{P}_O$ for summation: $T = \sum_{k=1}^{n} (\hat{t}_O)$. The n operations may be considered as 'serialized', even though the RTOS scheduler will break down each t_O into a number of CPU time slices and execute them in an interleaved way.

Thus, for a well-scheduled system we have to ensure that $T < \hat{P}_O$. Otherwise, there would be not enough processing time to let all the operators work in parallel.

Data Exchange Model

Since operators are independently scheduled real-time processes within our prototype, they cannot make blocking function calls to the previous or to the next operator for handing over tuples. Obviously, an intermediate buffer between each pair of consecutive operators is established. One of the operators takes the role of a data producer, the other has the role of a consumer regarding the intermediate buffer (figure 3).

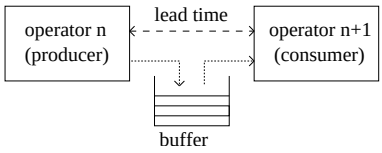

Figure 3: producer-consumer-relationship

Each time the producer completes a run, it puts the output tuples into the intermediate buffer for the consumer to pick them up at the start of its next working period. There are two conditions that must hold for the non-blocking data exchange: The producer must not try to write into a full buffer and the consumer must not try to read from an empty one. Thus, the minimum size of the buffer as well as the starting point of the different operators are of interest. In our prototype, we use the model of jitter-constrained periodic streams (JCP, [4]). This model is applied to a single producer-consumer relationship and allows the calculation of the minimum intermediate buffer size B and the lead time t_{lead} of the consumer. Relying on the model, we have to provide

- the average number of tuples the producer writes during one run
- the number of tuples the consumer reads during one run
- the period lengths P_O of both producer and consumer
- the producers' processing time t_O for one run

Additionally, this model takes into account the effects of jitter within the data stream and within the operators behavior: A size jitter allows the operator to produce a varying number of output tuples (depending on the stream data) per operation and a time jitter reflects the fact that the processing time for a run t_O may jitter in certain constraints. The general observation is: the larger the different jitters are, the higher the resulting intermediate buffer must be.

4 Prototype

We selected the Realtime Application Interface (RTAI, [3]) as operating system environment for QStream because, first, RTAI supports hard real-time as well as soft real-time. Second, real-time code may not only run as a kernel module but also as a user program. And third, a lot of data exchange mechanisms such as FIFOs, mailboxes and shared memory are originally provided by RTAI.

Within RTAI, our operators run in real-time mode while other processes (e.g. other Linux applications) have only non-real-time privileges. Non-real-time processes get the remaining processing time after all operators have finished the work within their periods.

Figure 4 illustrates measured times t_O of a filter operator running as a real-time process and the same operator running as non-real-time. In both scenarios we produce heavy load in the background.

The real-time graph of figure 4 shows that the background load does not significantly influence our periodic filter operator because the background process runs as non-real-time Linux process.

The measured run times for the filter operator in non-real-time mode are permanently higher than in real-time mode because the filter process only gets the same priority as concurrently running Linux processes.

Much more important, the filter's t_O increases by multiples of ten due to the varying load of the background process. Thus, for deterministic operator behavior we *must* rely on a real-time capable environment.

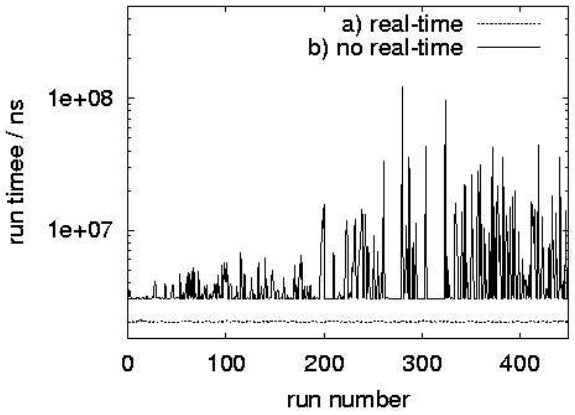

Figure 4: filter operator with background load

Controlling

The control application is executed in five steps. First, the control application traverses the user-given operator network and calculates the data rates between each couple of operators. This is done from the data sources (sensors) down to the data sink (last operator). If the resulting data rate at the data sink is higher than the user requirements (QoS), an initial sample operator is inserted in front of the first operator to reduce the amount of processed data.

Second, the control application calculates the operator period lengths with the operators' input and output data rate in mind. The model of jitter-constrained periodic streams is subsequently applied to each producer-consumer relationship of operators. This provides the intermediate buffer sizes and the lead time of an operator in producer role.

Third, the total memory requirement is obtained as the sum of all buffer sizes plus the static amount of operator memory. The absolute starting points of the single operators are derived from the relative values (lead time).

In a fourth step, the operator processes are initialized with the period length P_O and queued to start exactly at the pre-calculated point in time.

Finally, the operators continuously work and periodically start a new run after the specified period has elapsed.

Data Exchange

We exploit the features of the RTAI FIFOs for data exchange. An arbitrary number of applications is allowed to connect to such a FIFO, either as a FIFO writer (producer) or as a FIFO reader (consumer). In QStream, two FIFOs are established between consecutive operators: one for exchanging the tuples (variable tuple size) and one for handing over the packet control information.

The synchronization of the FIFO access is entirely managed by RTAI. From the operators' point of view, the FIFO is created with a size parameter which is equal to the buffer size B calculated in the control application. Relying on the scheduling and on the deterministic real-time behavior, none of our operators is blocked when reading from or writing to a FIFO.

The QStream demonstration will show how to negotiate QoS parameters for a given operator network and how the QStream operators run on top of the real-time environment.

References

[1] Don Carney, Ugur Çetintemel, Alex Rasin, Stan Zdonik, Mitch Cherniack, and Michael Stonebraker. Operator scheduling in a data stream manager. In *Proc. of 29th International Conference on Very Large Databases, September 9-12, 2003, Berlin, Germany*, pages 838–849, 2003.

[2] Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data, June 9-12, 2003, San Diego, CA, USA*, pages 647–651, 2003.

[3] Lorenzo Dozio and Paolo Mantegazza. Real time distributed control systems using rtai. In *In Proc. of the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, May 14-16, 2003, Hakodate, Hokkaido, Japan*, pages 11–18, 2003.

[4] Claude-Joachim Hamann. On the quantitative specification of jitter constrained periodic streams. In *Proc. of the Fifth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, January 12-15, 1997 Haifa, Israel*, pages 171–176, 1997.