Collecting Data Streams from a Distributed Radio-Based Measurement System

Marcin Gorawski, Pawel Marks, and Michal Gorawski

Silesian University of Technology,
Institute of Computer Science,
Akademicka 16,
44-100 Gliwice, Poland
{Marcin.Gorawski,Pawel.Marks,Michal.Gorawski}@polsl.pl

Abstract. Nowadays it becomes more and more popular to process rapid data streams representing real-time events, such as large scale financial transfers, road or network traffic, sensor data. Analysis of data streams enables new capabilities. It is possible to perform intrusion detection while it is happening, it is possible to predict road traffic basing on the analysis of the past and current vehicle flow. We addressed the problem of real-time analysis of the stream data from a radio-based measurement system. The system consists of large number of water, gas and electricity meters. Our work is focused on data delivery from meters to the stream data warehouse as quick as possible even if transmission failures occur. The system we designed is intended to increase significantly system reliability and availability. During this demonstration we want to present an example of the system capabilities.

Keywords: stream processing, fault-tolerance, sensor networks.

1 Introduction

These days it becomes more common to process continuous data streams. It may have application in many domains of our life such as: computer networks (e.g. intrusion detection), financial services, medical information systems (e.g. patient monitoring), civil engineering (e.g. highway monitoring) and more.

Thousands or even millions of energy meters located in households or factories can be sources of meter readings streams. Continuous analysis of power consumption may be crucial to efficient electricity production. Unlike other media such as water or gas, electricity is hard to store for further use. That is why a prediction of energy consumption may be very important. Real-time analysis of the media meter readings may help to manage the process of energy production in the most efficient way.

There are many systems for processing continuous data streams and they are still developed [1]. In [2] there is presented a fault tolerant Borealis system. This is a dedicated solution for applications where a low latency criterion is essential. Another system facing infinite data streams is described in [3]. Authors of

J.R. Haritsa, R. Kotagiri, and V. Pudi (Eds.): DASFAA 2008, LNCS 4947, pp. 702–705, 2008.

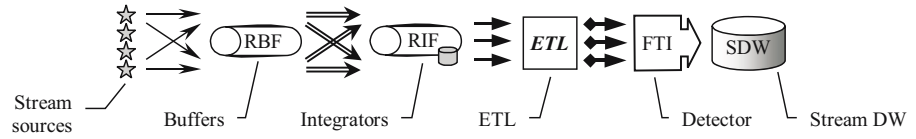

Fig. 1. Layered structure of the distributed telemetric system

the work deal with sensors producing data continuously, transferring the measured data asynchronously without having been explicitly asked for that. They proposed a *Framework in Java for Operators on Remote Data Streams* (Fjords).

In our research we have focused on processing data originating from a radio-based measurement system [4]. We carried research on efficient recovery of interrupted ETL jobs and proposed a few approaches [5,6] based on the Design-Resume algorithm [7].

Basing on the previous experience, we have focused on fault-tolerance and high availability in a distributed stream processing environment. In [8] we proposed a new set of modules increasing the probability that a failure of one or more modules will not interrupt the processing of endless data streams. Then we prepared a model [9,10] of data sources to estimate the amounts of data to be processed useful in the configuration of the environment. In this paper we want to present how the system works in practice and how it has been implemented.

2 Research System

Our research is based on a telemetric system [4] for remote and automatic reading of media consumption meters. It's main task is transferring data from particular meters using wireless communication to local collecting nodes (further called *data stream sources*). Next, the data streams need to be transferred to a *stream data warehouse* (SDW) in which the data can be processed analysed.

Our goal is to assure the continuity and correctness of the data streams being transmitted into a stream data warehouse. We want the process to run continuously, and we want it to be failure resistant. There are two goals to be achieved: (a) transferring all the data, (b) transferring data as fast as possible, minimizing the delay between measurement and loading into SDW. To achieve the above mentioned goals we introduce three additional layers to the system structure [8] (Fig. 1): RBF remote buffers, RIF persistent integrators, FTI detector.

Data sources (collecting nodes) are very simple devices comparable to LAN routers. Their task is to receive data from meters and send it further to the destination. The most important feature of the data sources is their inability to buffer data. If the outgoing connection is lost, the outgoing data is lost also. To avoid such situations each data source transmit data to many RBF buffering modules. If one connection is lost, the others remain.

RBF buffers are simple and low-cost buffering modules. Because they are cheap, they can be used in many copies. The more RBFs receive data from a data source, the lower is the probability of data loss. The RBFs not only

receive data from sources, they also transmit it to the subsequent system layer. They also support short-term buffering which avoids loss of data in case of short communication failures. Unfortunately, it does not protect the system against data loss during longer communication breaks.

We also introduce a layer of RIF persistent integrators. RIF modules extend the RBF functionality. They offer persistent buffering based on a persistent storage (e.g. hard disk). Another task of an RIF module is integration of stream parts from many RBFs into a single data stream. It is necessary when one RBF fails, and the RIF starts to receive data from another one. The data safely received can be processed now.

Behind the RIF layer we placed the layer of ETL modules. ETL stands for *Extraction, Transformation and Loading*. In this layer the data streams received from RIF modules are processed (filtered, transformed, recalculated, joined, aggregated). The ETL process is described using *Directed Acyclic Graph*. Graph nodes are responsible for data processing, whereas graph edges define tuple flow directions. Graph nodes belong to one of three node classes: extractors responsible to retrieving data to be processed, transformations in which tuples are processed, and inserters which save data in a destination. The ETL process supports three resumption algorithms: Design-Resume (DR) [7], hybrid DR-based resumption [5] and checkpoint-based algorithm [6]. In stream processing only checkpointing is applicable.

The FTI layer is responsible for merging redundant data stream from replicated ETL processes. It also checks the correctness of received data comparing all received stream copies. Finally the data is stored in the data warehouse.

The use of checkpointing enables replication and migration of ETL process. Saved ETL process state can be easily copied and used to restart ETL on another machine or network node. The checkpointing algorithm uses filtration known from DR algorithm. It enables synchronization with streams incoming from RIFs and delivered to the last FTI layer.

3 Technical Details

For research purposes the system has been implemented in Java. We have built the stream data generator which works similarly to the real collecting nodes. RBF and RIF modules are also implemented in Java, although in real world RBFs should be a small and simple devices based on microcontrollers e.g. AVR or ARM equipped with necessary communication interfaces. The RIF module can be built similarly to RBF using microcontroller; however, it is not to be used in so many copies as RBFs, so its functionality can be realized in a PC running necessary software.

ETL module is a set of Java classes implementing ETL process, communication with RIF and FTI layers and resumption algorithms. Each instance of an ETL process is started in a separate *Java Virtual Machine* (JVM).

Each system module (data generator, RBFs, RIFs, ETLs, FTI) runs on a separate instance of JVM. This references the complete independence of modules

in real world. The communication between layers uses RMI and TCP/IP connections. RMI interfaces of modules are required to locate needed services and initialize the communication. When the modules agree for the connection options after exchange of host IPs and port number, they exchange data via TCP/IP connections, which are much more efficient than RMI.

During experiments failures are simulated in two ways: by "unexpected" terminating of random system modules or by breaking the connections between system modules. In both cases the system is expected to handle the failure correctly and continue the processing without any data loss.

Our demonstration is going to be preceded by a short slideshow clarifying how the system works. During the demonstration we want to show the working system using the software simulator running on a PC-class machine. We are going to simulate module failures and observe the system reaction.

References

1. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Motwani, R., Nishizawa, I., Srivastava, U., Thomas, D., Varma, R., Widom, J.: Stream: The stanford stream data manager. IEEE Data Eng. Bull. 26(1), 19–26 (2003)
2. Balazinska, M., Balakrishnan, H., Madden, S., Stonebraker, M.: Fault-Tolerance in the Borealis Distributed Stream Processing System. In: ACM SIGMOD Conf., Baltimore, MD (2005)
3. Madden, S., Franklin, M.J.: Fjording the stream: An architecture for queries over streaming sensor data. In: ICDE, pp. 555–566. IEEE Computer Society, Los Alamitos (2002)
4. Gorawski, M., Malczok, R.: Distributed spatial data warehouse indexed with virtual memory aggregation tree. In: Sander, J., Nascimento, M.A. (eds.) STDBM, pp. 25–32 (2004)
5. Gorawski, M., Marks, P.: High efficiency of hybrid resumption in distributed data warehouses. In: DEXA Workshops, pp. 323–327. IEEE Computer Society, Los Alamitos (2005)
6. Gorawski, M., Marks, P.: Checkpoint-based resumption in data warehouses. In: Socha, K. (ed.) IFIP International Federation for Information Processing, Warsaw. Software Engineering Techniques: Design for Quality, vol. 227, pp. 313–323 (2006)
7. Labio, W., Wiener, J.L., Garcia-Molina, H., Gorelik, V.: Efficient resumption of interrupted warehouse loads. In: Chen, W., Naughton, J.F., Bernstein, P.A. (eds.) SIGMOD Conference, pp. 46–57. ACM, New York (2000)
8. Gorawski, M., Marks, P.: Fault-tolerant distributed stream processing system. In: DEXA Workshops, pp. 395–399. IEEE Computer Society, Los Alamitos (2006)
9. Gorawski, M., Marks, P.: Towards reliability and fault-tolerance of distributed stream processing system. In: DepCoS-RELCOMEX, pp. 246–253. IEEE Computer Society, Los Alamitos (2007)
10. Gorawski, M., Marks, P.: Distributed stream processing analysis in high availability context. In: ARES 2007: Proceedings of the The Second International Conference on Availability, Reliability and Security, Washington, DC, USA, pp. 61–68. IEEE Computer Society, Los Alamitos (2007)