Real-Time Production Monitoring in Large Heterogeneous Environments

Arne Schramm, Bernhard Wolf, Raik Hartung, André Preußner

Real-time production monitoring in large heterogeneous environments

Arne Schramm, Bernhard Wolf, Raik Hartung, André Preußner

SAP AG, SAP Research Dresden, Chemnitzer Str. 48, 01187 Dresden

Abstract. Expensive production equipment requires continuous monitoring to gather data in real-time, e.g., to detect problems, to assess the quality of produced parts, to collect information about the machine states, and consequently to optimize production processes. However, the increasing amount of data – from sensors or systems – and its continuous processing are a big challenge for an IT infrastructure.
This paper presents a hybrid system, consisting of a distributed CEP (complex event processing) system to process data in real-time augmented with an in-memory database to extend the available memory as well as the processing capabilities of the overall system. Besides the description of the system architecture, details about the implementation of the concepts in a real production environment are given.

1 Introduction

Modern production facilities are being controlled and monitored by an IT back end consisting of various systems such as Enterprise Resource Planning (ERP), Manufacturing Execution Systems (MES), Supervisory Control and Data Acquisition (SCADA), and numerous analysis and decision support systems. The constant monitoring of production assets helps companies to save costs, predict downtimes, prevent the production of scrap, and dynamically adjust production processes. State-of-the-art production machinery often is equipped with numerous sensors, such as vibration, power or temperature sensors, giving information on the health and production state of the machine. In their effort to also optimize older equipments, manufacturers apply new sensors to existing machines to get more insight.

While more and more data is collected at the shop floor, analyzing this data has become a serious challenge. For instance, a mid size semi-conductor equipment can create up to 40.000 data values per second. Having an entire production line consisting of 500 to 1000 machines can easily generate more than one terabyte of data per day. This amount of data brings a number of challenges. The mere gathering of the data in a back end system can already be a problem for any network, given that it might not be used exclusively for this purpose. Analyzing this data in real-time is another non-trivial task that needs to be done in order to react in time to any event on the shop floor. In addition to scheduled analysis jobs like the extraction and continuous refinement of patterns

found in historic data, system engineers need to query historical and real-time production data ad-hoc, to fully understand the systems in their responsibility, to react in time, and to find possibilities for improvements.

We present a hybrid system combining distributed complex event processing (CEP) [1] and in-memory database technologies to collect and analyze large amounts of data in typical heterogeneous production environments in real-time. The system enables users to dynamically query streams of data and subscribe to results. The combination of both technologies allows users to analyze live and historical data in real-time. Furthermore, analyses can be adapted dynamically to explore the behavior of the production system instantly.

2 Related Work

In our work we combine knowledge from two domains: stream processing and business intelligence (BI). Stream processing and CEP mechanisms are used for processing production data in real-time. The results can be analyzed like in typical BI scenarios using data warehouse applications. In this section earlier work from both areas is discussed.

Originally, the purpose of data warehouses (DW) was to support strategic decisions [2]. With the increasing dynamics of markets the demand for tactical or even operational decision support emerged and approaches for "active data warehousing" [3] were introduced. In [2] different scenarios for real-time data warehousing are discussed. An architecture is described which processes events in real-time and instantly stores the results to the DW. Analyses are executed based on the persisted DW data automatically or on a user's request, i.e., representing a snapshot at this time. However, updating the data in DW does not automatically update the results of analyses.

Over the last decade several stream processing and CEP systems were proposed and commercialized [4]. These systems enable real-time processing directly in the main memory of the system. Queries are continuously executed on the streaming data providing results updated in real-time. To increase performance and fault tolerance distributed stream processing systems were developed, that allow for early data processing close to the data sources. The disadvantages compared to a DW are that data in the main memory are volatile and limited to the much smaller size of the main memory (gigabytes vs. terabytes).

To leverage the advantages of both data processing paradigms, hybrid systems , such as the federated stream processing architecture MaxStream [5], were introduced. MaxStream extends the SAP MaxDB Database Federation Engine with data agents, that act as an access point to stream processing engines. Two new operations to support stream queries were implemented: Streaming Input and Monitoring Select. The first enables the system to pass data streams to the federator and to persist them. The latter returns results once as soon as new data sets are found in the output table. The benefit of this system is the possibility to deploy distributed queries over stream processing engines and databases. Thus, historical and current data can be considered in one query. Since contin-

uous queries are not supported, results have to be polled from the federator by using the monitoring select statement, which is a major drawback for real-time processing.

3 The Hybrid Monitoring System

The systems architecture consists of four layers (Figure 1). Those layers are clearly separated by concerns. Starting from the data sources (shop floor) to the data sinks (users or systems) we defined the following layers:

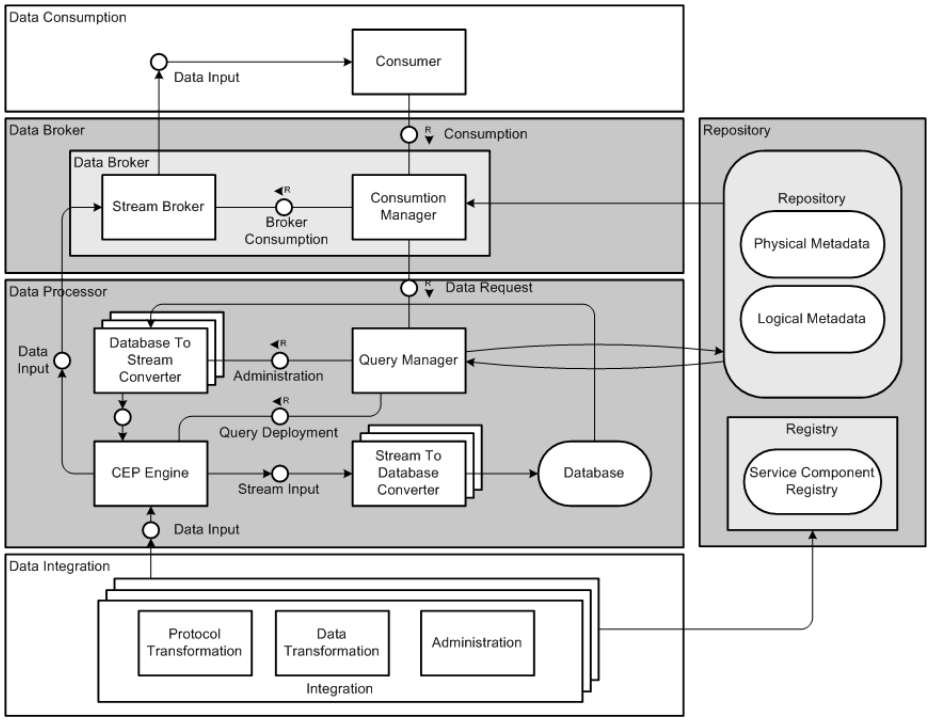

Fig. 1. Architecture Overview

The *Data Integration* layer guarantees the seamless integration of shop floor data from arbitrary source systems into our solution. Its main tasks and components are described in Section 3.1.

The *Data Processor* layer receives and processes real-time as well as historical data. Results are continuously forwarded to the next layer. A detailed description of the layer is given in Section 3.2.

The *Data Broker* layer acts as a single point of entry to the system for various data consumers. It takes care of distributing processing results, handling

subscriptions and providing system information. Section 3.3 depicts details of this layer.

All data of the system (raw data, processed data and metadata) can be accessed in the *Data Consumption* layer. Section 3.4 gives insight in tasks and requirements of this layer.

The management of the systems components as well as the provisioning of metadata is handled by a *Management* component (Section 3.5) spanning multiple layers.

3.1 Data Integration

Due to the fact that shop floor infrastructure is already defined in most application scenarios, our architecture needs to be integrated into it. This means it needs to connect to various systems, equipments and sensors that can have standardized or proprietary protocols and data format definitions. Data from the shop floor is sent to the *Data Processor* layer via the data input interface. Therefore, the *Data Integration* layer has to provide the following functionality:

- connectivity to sensors and systems,
- protocol and data transformation, and
- unified, push based data transfer to the *Data Processor* layer.

Since integration always needs to be tailored to the specific environment; there cannot be a generic solution to that. However, a common approach to minimize integration effort is the usage of standardized integration middleware. The integration solution in our architecture needs to fulfill the requirements listed above and support a distributed collection of all data. As stated in the introduction a single equipment can already produce thousands of values per second. Collecting all raw data of a whole production site centrally is not feasible. Therefore data has to be collected and integrated by a distributed integration solution.

3.2 Data Processor

The *Data Processor* layer contains the components needed to:

- query real-time data (e.g., calculate live performance indicators (PI)),
- query historical data (e.g., retrieve reference data),
- combine real-time and historical data (e.g., compare live and historical data),
- persist query results (e.g., for further reference), and
- manage running queries (create, deploy, start, stop, undeploy).

The *Data Integration* layer pushes all shop floor data to the CEP component in the *Data Processor* layer. There it can be processed, persisted, and forwarded to the next layer. The processing of data is done by CEP queries. These queries define logic that is executed on all or only selected data arriving. A typical CEP query could, e.g., collect all power consumption data of a machine for a given

time and continuously calculate the moving average on it[1]. The results of queries can be input for further queries, persisted in the Database or forwarded to the Stream Broker (Section 3.3) for distribution.

Streams can also be persisted in the Database using the Stream To Database Converter. Previously persisted raw data or query results can be used for complex analyses like pattern detection or for combination with real-time data. The persisted data of a machine could, e.g., be used as a reference for the machine's currently collected data to detect changes in the machine behavior and predict failures or problems. To retrieve data from the Database, the Database To Stream Converter creates streams from database tables and passes them to the CEP Engine component.

The Query Manager component handles multiple CEP nodes which are located close to the integration nodes in the network. The management of these nodes comprises the starting and stopping of nodes as well as the deployment and distribution of queries.

3.3 Data Broker

The purpose of this layer is to provide a single point of access for data consumers. It consists basically of two components - the Stream Broker and the Consumption Manager. While the Stream Broker has the typical tasks of a broker in a Message-Oriented-Middleware (MOM) like JMS [7] or WS-Notification [8], the Consumption Manager provides the interface to the upper layer. By using the Consumption Manager, consumers can read all relevant data of the system, run queries and subscribe to results.

The benefit of this separation is increased scalability: Classical MOM approaches like JMS lack scalability because of the centralized broker. This limits the maximum number of messages the system can handle in a certain time frame. In our approach, this problem is solved by running multiple stream broker instances. Moreover, this is completely transparent for consumers, since they are only communicating with the Consumption Manager. The Consumption Manager stores, which result streams are registered at which broker instance, and forwards subscriptions accordingly.

3.4 Data Consumption

The *Data Consumption* layer at the top level groups all potential data consumers. A Consumer component can be a graphical dashboard or other IT systems like ERP, MES or SCADA. Dashboards can support engineers in various tasks such as monitoring production equipment or exploring the system and finding possibilities for improvements. IT systems can subscribe to streams to adapt production processes or trigger actions based on events in the shop floor.

[1] Details of a reference model for performance indicators developed in the KAP project are described in [6].

Any Consumer in our architecture implements the Data Input interface to receive (processed) stream data from the Stream Broker. To retrieve metadata, to send subscriptions, and to deploy new queries, the Consumer uses the Consumption interface implemented by the Consumption Manager in the *Data Broker* layer.

3.5 Management

The Management component spans the *Data Processor* layer and the *Data Broker* layer. It contains two sub-components – the Registry and the Repository.

The Registry component is used by all other components for service discovery inside the system. Every component registers itself at the Registry when started. Other components can then find the registered components via the Registry, so that the system does not need to be configured manually when starting components on different nodes in the network. The Repository contains all metadata describing the system and the data available in it. We distinguish between physical metadata and logical metadata.

Physical Metadata Repository (PMR): The PMR provides data helping the user understanding the real-world system landscape and mapping it to data in the streaming system. It also contains data needed by the Query Manager and Consumption Manager to manage queries in the system. The PMR holds a hierarchy of physical components which is used to model the real-world environment. A physical component is meant to be any kind of shop floor device, e.g., a sensor. The hierarchy of physical components represents the structure at the shop floor.

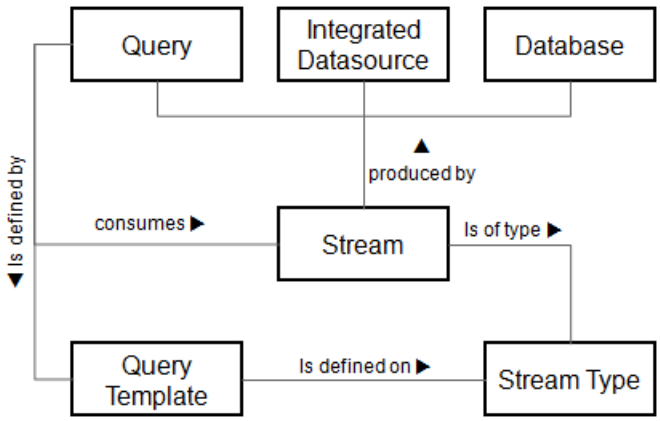

Fig. 2. Simplified Repository Structure

Logical Metadata Repository (LMR): The LMR contains information about streams and queries available in the system. A simplified structure of the LMR

meta-model is depicted in Figure 2. The central entity is the stream. A stream can have different sources: it can either be produced from a database table, it can be retrieved directly from a physical component, be the result of a query, or a combination of these. A stream is an instance of a defined stream type. This classification allows to check, whether a stream can be processed in a certain query or not. We envision a query as the combination of a query template with concrete input streams and parameters. A query template defines the logic of a query that can be applied to streams of a certain stream type. The concept of streams, stream types, queries, and query templates allows the definition of complex logic by an expert, which can then be re-used by consumers of the system to analyze actual data from concrete streams.

4 Implementation

The concepts were implemented using SAP's CEP and in-memory solutions. However, other CEP or in-memory systems can be integrated in a simple way by utilizing the defined interfaces. The implementation is currently running in a real production facility processing PIs [6] in real-time.

A performance analysis showed the real-time capabilities of our system. For that, we distinguish between two message types: control and data messages. Control messages are used to manage the system and are not critical w.r.t. performance. In our architecture control messages are sent via web services usually taking less than 300ms. In contrast, data messages contain real-time stream data and are sent directly via UDP. The overall performance depends on the CEP, the Stream Broker, the networks bandwidth and the message size. With a typical message size of 250 Byte the Stream Broker can send and receive up to 40000 messages per second in a 100 MBit network. In a 1 GBit network the value increases to 90000 messages per second with a latency of 0.6 ms.

The results do not include a CEP. The performance of the CEP depends on the specific implementation and the complexity of the queries. There are currently no standardized CEP benchmarks, however, a first impression of the performance can be found in [9]. Compared to the broker's performance the throughput is similar, thus, a slowdown of the system is not expected.

5 Conclusions

A main challenge of today's production environments is real-time analysis of large volumes of data collected at the shop floor. The heterogeneous IT infrastructure, changing requirements and different standards are additional issues to be solved in order to achieve an end-to-end data integration.

In this paper we presented an approach for real-time production monitoring in large, heterogeneous environments. Our solution combines emerging data processing technologies, i.e., a distributed CEP system with in-memory databases. By this combination, we compensate the limitations of the CEP system in terms of available memory, and further integrate persisted data without having the

update latency of a database. To further increase the performance of the data processing, stream brokers are utilized to separate the data processing from result distribution.

The application of this hybrid system to the manufacturing domain enables us to continuously (pre-)process data of the shop floor assets in real-time and close to its source. Consequently, network load is reduced whereas scalability is increased. Our system allows to create and deploy new queries dynamically so that running analyses can be adapted to changed requirements.

6 Acknowledgements

The paper presents results of the KAP project (260111), which is co-funded by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7).

References

1. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Professional (May 2002)
2. König, S.: Erfolgsfaktoren in Business Intelligence Projekten. Technical Report 03-2009, Fachhochschule Hannover, Fakultät IV – Wirtschaft und Informatik (2009)
3. Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.E.: Supporting Streaming Updates in an Active Data Warehouse. In: Proceeding of the IEEE 23rd International Conference on Data Engineering 2007 (ICDE 2007), IEEE (2007) 476–485
4. Cugola, G., Margara, A.: Processing Flows of Information: From Data Stream to Complex Event Processing. ACM Computing Surveys **44**(3) (June 2012) 15:1–15:62
5. Botan, I., Cho, Y., Derakhshan, R., Dindar, N., Haas, L.M., Kim, K., Tatbul, N.: Federated Stream Processing Support for Real-Time Business Intelligence Applications. In: Enabling Real-Time Business Intelligence Third International Workshop, BIRTE 2009, Held at the 35th International Conference on Very Large Databases, VLDB 2009. Volume 41 of LNBIP., Berlin Heidelberg, Springer (2010) 14–31
6. Hesse, S., Wolf, B., Rosjat, M., Nadoveza, D., Pintzos, G.: Reference model concept for structuring and representing performance indicators in manufacturing. In Emmanouilidis, C., Taisch, M., Kiritsis, D., eds.: Competitive Manufacturing for Innovative Products and Services: Proceedings of the APMS 2012 Conference, Advances in Production Management Systems. IFIP AICT (2012)
7. Sun Microsystems Inc.: Java Message Service Specification – Version 1.1 (April 2002)
8. Graham, S., Niblett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Tuecke, S., Vambenepe, W., Weihl, B.: Web Services Notification (WS-Notification) – Version 1.0 (January 2004)
9. Sybase Inc.: Evaluating Sybase CEP Performance. Technical white paper, Sybase Inc. (October 2010)