PAN – Distributed Real-Time Complex Event Detection in Multiple Data Streams

Lukas Probst, Ivan Giangreco, Heiko Schuldt

PAN – Distributed Real-Time Complex Event Detection in Multiple Data Streams

Lukas Probst Ivan Giangreco Heiko Schuldt

Databases and Information Systems Group
University of Basel, Switzerland
{lukas.probst|ivan.giangreco|heiko.schuldt}@unibas.ch

Abstract. In this paper, we present PAN, a generic middleware for distributed real-time complex event detection (CED) which is able to analyze multiple distributed data streams. In PAN, CED applications are defined as workflows and are executed by dedicated workers in a distributed way in a P2P network. In consequence, PAN is scalable in terms of the number of data streams and the complexity of the analyses. Evaluations based on an extended version of the ACM DEBS 2013 Grand Challenge scenario show the effectiveness and efficiency of PAN.

1 Introduction

The last decade has seen a vast proliferation of devices that sense their environment. As according to the IoT vision most of them are connected to the Internet, they are able to disseminate the data they measure in form of continuous data streams. Hence, the number of data streams and the volume of streamed data has increased enormously. Nevertheless, the analysis of a single or multiple of these Big Data streams in real-time is essential. In particular, the detection of complex events out of the raw streaming data in real-time is a major challenge and at the same time an important aspect in a variety of applications. As an example, consider a soccer team in which each player is equipped with several sensors which produce continuous data streams. These streams need to be analyzed to produce added value on a match for different stakeholders (clients). Hence, different events need to be detected out of all the incoming streams. This requires an infrastructure that i.) allows to implement, in a modular way, basic components for detecting simple events, that ii.) supports the combination of these components for complex event detection into workflows and that iii.) scales with the number of streams and with the complexity of the analyses.

The first generation of complex event detection (CED) systems, also called complex event processing (CEP) systems, has been built with a centralized architecture (e.g., [1, 8, 10]). This significantly limits their scalability, especially if complex events have to be detected in real-time. More recent approaches (e.g., [2, 4, 6]) use a distributed architecture and forward streams in a publish/subscribe style between workers that perform parts of the complex event detection.

In this paper, we introduce PAN (*P*2P *A*nalysis *N*etwork), a generic distributed real-time CED middleware which jointly addresses the challenges we

have listed above. PAN is able to analyze multiple distributed input data streams and to concurrently handle several analysis requests of different clients. In PAN, CED applications are defined as workflows on top of components implementing basic event detectors or other analysis operators such as aggregators. These workflows are executed in a distributed way in a P2P network based on publish/subscribe communication between workers. As a result, PAN has a high degree of scalability.

The contribution of this paper is twofold. First, we present PAN, a novel middleware architecture for distributed and scalable CED that seamlessly combines ideas from workflow management (definition of CED workflows) and P2P systems (distributed, scalable CED). Second, we provide the results of an evaluation of PAN's performance and scalability characteristics on the basis of a sports use case using an extended version of the ACM DEBS 2013 Grand Challenge scenario [7, 11]. The results show the effectiveness of the PAN approach.

The remainder of this paper is organized as follows: We introduce PAN in Sect. 2 and report on the evaluation of PAN in Sect. 3. Section 4 presents related work and Sect. 5 concludes.

2 PAN

In this section, we present and discuss the concepts of PAN. The main idea behind PAN is to obtain a high degree of scalability for real-time CED applications by distributing the workload across several peers in an unstructured P2P network and communicating via publish/subscribe.

2.1 CED Workflows in PAN

In PAN, CED applications are defined by means of *workflows*. They consist of so-called *workers* which provide basic functionality for CED and which are combined using a partial order that allows both sequential and parallel execution of workers, depending on the semantics of a concrete CED application. Figure 1 illustrates a sample workflow which generates the player as well as the team ball possession statistic streams for the soccer use case. This workflow includes several intermediate streams (e.g., *BALLHITS*), i.e., streams that are generated as output streams by some workers and consumed as input streams at other, subsequent workers. The *sensor devices* producing the initial input streams are sources of a workflow. Moreover, there are devices outside PAN called *clients* that consume the output streams of the workers. The clients are the sinks of a workflow and might join it only for a short time.

Each worker is hosted on a *peer*. Each peer, in turn, can host a single or multiple workers. With this design, PAN obtains a high degree of flexibility in terms of workflow distribution since the workflow can be either executed on a single machine or fully distributed onto a large number of peers hosting workers which compute only small subtasks.

To standardize the interaction between workers in PAN, they only share data via network communication, independent from their deployment.

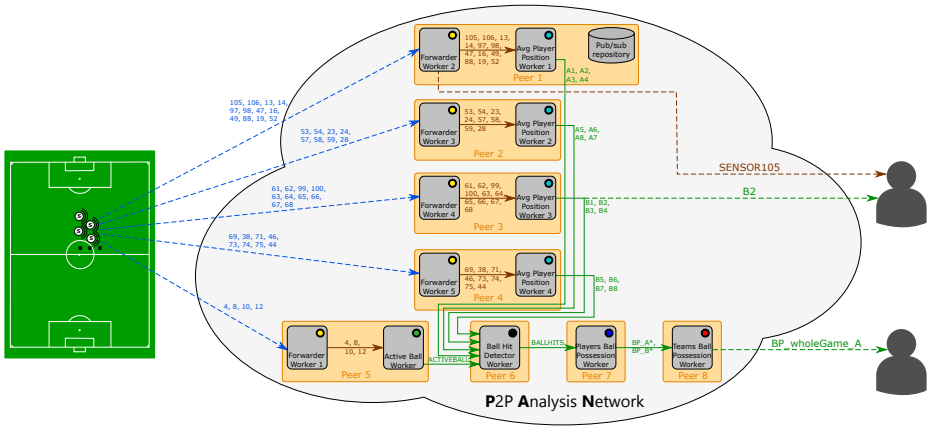

Fig. 1. Sample CED workflow in PAN for soccer game analysis. The initial input streams are taken from the ACM DEBS 2013 Grand Challenge [7, 11].[2]

2.2 Publish/Subscribe

Hard-wiring the communication between the workers would lead to a highly inflexible system, earmarked for a specific workflow. In contrast, the publish/subscribe style of interaction allows to decouple the sender of a data stream and its receiver. PAN uses a central publish/subscribe repository which stores a mapping from the stream identifier to a list of publishers. When a new worker is deployed, it has to publish all its output streams. Subsequently, all potential subscribers (clients or other workers) can use this information to identify the publisher and fetch data stream elements from there. Note that the repository has to be contacted only once, when the link between subscriber and publisher is established. Hence, the central repository does not become a bottleneck.

2.3 PAN Workers

A PAN worker is a building block for the CED workflows. At the interface, each worker generates one or several output stream(s) on the basis of one or several input stream(s) it consumes – either directly from a sensor or from other workers. The input streams are processed by one or several components inside the worker. The output streams contain the analysis results of these components.

PAN uses separate ring buffers to handle a worker's input and output streams and thus to connect two workers. A worker's input ring buffers contain the latest data stream elements of the input streams while the output ring buffers are filled with the data stream elements created by the worker's components. Each worker runs a server to enable downstream workers and clients to fetch the data stream

[2] Soccer field graphic: `https://de.wikipedia.org/wiki/Datei:Offsidelarge.svg`
Client icon: `http://www.flaticon.com/packs/humans-3`

elements stored in its output ring buffers. To fill its input ring buffers, a worker can fetch new data stream elements from a publisher periodically or on demand. In contrast, sensors always push their data to the first workers of a CED workflow which then forward by publishing the streams to all other workers and clients.

All generic and application-specific components of a worker run in parallel. A component can use all input data streams for its analysis task. However, a worker's components are strictly separated from each other. They neither share state, nor can they directly communicate with each other.

2.4 Scalability

Increasing the number of sensor data streams to be analyzed, the number of different analyses that have to be performed, or the complexity of these analyses results in an increased computational effort. PAN can handle this by distributing the overall workload across more workers which can then be deployed on peers with free capacities. In consequence, PAN scales w.r.t. the number of data streams and w.r.t. the complexity and the number of analyses.

3 Evaluation and Implementation

In our evaluation each peer is deployed in the Azure Cloud platform[3]. The ping between two peers is around 0.9 ms. Both clients and (simulated) sensors are deployed on a separate physical server[4] with a ping around 21 ms to the Cloud.

In order to create a CED application that runs on top of PAN one only has to implement the workers in Java and to specify the workflow in a JSON config file, similar to TechniBall's XML approach [9]. The config file is used to automatically deploy the CED system. The actual connection between the workers is done at start-up time using the publish/subscribe repository. Due to space limitations, we refer to [12] for further information on the implementation of PAN.

For evaluation purposes, we have implemented a soccer analysis application that generates ball possession streams using the dataset from the ACM DEBS 2013 Grand Challenge [7, 11] in multiple steps (see Fig. 1). The input data streams are created by sensors attached to the shin guards of the players and inside the ball and include position, timestamp, velocity and acceleration info. We have used the first 25 minutes of the soccer match for our evaluation.

PAN's performance is measured using the *query delay* that indicates how long the system needs to calculate and generate a certain output data stream. It is defined as the difference between the machine time when receiving an output data stream element at the client $MT(c)$ and the machine time $MT(s)$ at which the corresponding sensor data stream element has been emitted. Note that the query delay comprises also the time for sending the input stream to the first worker and for fetching the stream from the last worker of the CED workflow.

[3] Small VM instances (standard A1), 1 core 1.6 GHz CPU, 1.75 GB RAM
[4] Lenovo ThinkPad W530, Intel Core i7-3820QM CPU @ 2.70 GHz, 12 GB RAM

Table 1. Average query delay with increasing number of peers

Stream	3 peers	6 peers	8 peers	14 peers
SENSOR105	69.76 ms	66.61 ms	73.79 ms	88.96 ms
B2	2923.73 ms	123.99 ms	165.68 ms	114.72 ms
BP_wholeGame_A	1141.07 ms	1078.43 ms	948.44 ms	961.06 ms

We analyze PAN's performance by varying the number of peers hosting the workers of the ball possession workflow. More precisely, we use four different deployments with 3, 6, 8, and 14 peers. The client periodically (every 20 ms) fetches the latest data element of three streams, produced at different positions in the workflow: a forwarded sensor data stream (*SENSOR105*), an intermediate output data stream (*B2*) and a final output data stream (*BP_wholeGame_A*).

Table 1 lists the results of this evaluation. While the average query delays of the *SENSOR105* and the *BP_wholeGame_A* streams are rather constant, the query delay of the *B2* stream is approximately 20 times higher in the three peers setting than in the other settings. With only three peers, the two peers that are supposed to be responsible for generating the average player position streams are not capable of doing so. However, the evaluation shows that PAN can solve such computational bottlenecks by distributing the workflow onto more peers.

4 Related Work

Similar to PAN, also RACED [4] distributes the detection components in a P2P network and links these components using publish/subscribe. However, since in RACED the data stream requested by a client has to be generated along its shortest path tree (SPT), clients cannot share a workflow if they do not have the same SPT while in PAN no duplication is needed. In [6] the authors of RACED propose a single-tree deployment strategy for their T-REX middleware [5] that allows workflows to be shared by clients such that the same output stream does not have to be generated multiple times. However, while PAN is a worker-based CED middleware, RACED and T-REX are language-based CED middleware approaches that suffer from some limitations as the client can neither define complex calculations nor small programs that have to be performed in order to detect a complex event or to generate the corresponding output data stream element. The same is true for all other language-based CED middleware systems such as, for instance, Amit [1] or Cayuga [8]. Worker-based CED middleware systems like PAN or OSIRIS-SE [2], in contrast, facilitate the implementation of arbitrary workers in a modular way and thus do not suffer from such limitations.

In reply to the ACM DEBS 2013 Grand Challenge [7, 11], six systems have been proposed [3]. While the workers and CED workflows of PAN are based on the requirements of the challenge and thus have some similarity with all these systems, PAN's architecture is mainly influenced by the approach of Jergler et al. [10] that proposes a workflow-based architecture for CED in which

different workers are connected with non-blocking ring buffers. While [10] states that the concept can in general be implemented in a distributed way using publish/subscribe, only a centralized implementation is presented. Hence, PAN fills a void as it promotes these concepts to a distributed and thus scalable system.

5 Conclusion

In this paper, we have introduced the distributed real-time CED middleware PAN. It uses workflows to define CED applications and distributes the workload onto multiple workers hosted by peers in a P2P network. The worker-based approach allows to implement parts of CED workflows in a modular way, ranging from simple stream forwarding to highly sophisticated analyses. Evaluations have shown that PAN is able to eliminate computational bottlenecks by distributing the workflow on more peers. In our future work, we plan to organize the workers in a structured P2P network to store the mapping of streams to publishers in a distributed and reliable way. Moreover, we plan to further analyze, evaluate and compare different approaches to publish/subscribe-based communication, in particular a pull-based vs. a push-based communication approach.

References

1. Adi, A., Etzion, O.: Amit - the situation manager. VLDB Journal 13(2) (2004)
2. Brettlecker, G., Schuldt, H.: Reliable distributed data stream management in mobile environments. Information Systems 36(3) (2011)
3. Chakravarthy, S., et al. (eds.): The 7th ACM International Conference on Distributed Event-Based Systems, DEBS '13, Arlington, TX, USA. ACM (2013)
4. Cugola, G., Margara, A.: Raced: an Adaptive Middleware for Complex Event Detection. In: Proc. ARM'09. Urbana Champaign, IL, USA (2009)
5. Cugola, G., Margara, A.: Complex event processing with t-rex. Journal of Systems and Software 85(8) (2012)
6. Cugola, G., Margara, A.: Deployment strategies for distributed complex event processing. Computing 95(2) (2013)
7. ACM DEBS 2013 Grand Challenge Description. `http://www.orgs.ttu.edu/debs2013/index.php?goto=cfchallengedetails`
8. Demers, A.J., et al.: Cayuga: A General Purpose Event Monitoring System. In: Proc. CIDR'07. Asilomar, CA, USA (2007)
9. Gal, A., et al.: Grand Challenge: The TechniBall System. In: Proc. DEBS'13. ACM, Arlington, TX, USA (2013)
10. Jergler, M., et al.: Grand Challenge: Real-time Soccer Analytics Leveraging Low-latency Complex Event Processing. In: Proc. DEBS'13. Arlington, TX, USA (2013)
11. Mutschler, C., Ziekow, H., Jerzak, Z.: The DEBS 2013 Grand Challenge. In: Proc. DEBS'13. Arlington, USA (2013)
12. Probst, L.: PAN – A P2P Approach for Scalable Complex Event Detection in Distributed Data Streams. Master's thesis, University of Basel (2014), `http://dbis.cs.unibas.ch/downloads/theses/MSc_Thesis_Lukas_Probst.pdf/at_download/file`