From Inception to Execution: Query Management for Complex Event Processing as a Service

Wilson A. Higashino, Cédric Eichler, Miriam Capretz, Thierry Monteil, Maria Beatriz F. de Toledo, Patricia Stolf

From Inception to Execution: Query Management for Complex Event Processing as a Service

Wilson A. Higashino*†, Cédric Eichler‡§¶,
Miriam A. M. Capretz*, Thierry Monteil‡¶, Maria Beatriz F. de Toledo† and Patricia Stolf§¶
‡CNRS; LAAS; F-31077 Toulouse, France. {eichler,monteil}@laas.fr §IRIT; F-31062 Toulouse, France. {eichler,stolf}@irit.fr
*Dept. of Electrical and Computer Engineering, Western University, London, ON, Canada. {whigashi, mcapretz}@uwo.ca
†Instituto de Computação, Univ. Estadual de Campinas, Campinas, Brazil. {wah, beatriz}@ic.unicamp.br
¶ Univ de Toulouse, UPS F-31400, INSA, F-31400, UTM F-31100, Toulouse, France.

***Abstract*—Complex Event Processing (CEP) is a set of tools and techniques that can be used to obtain insights from high-volume, high-velocity continuous streams of events. CEP-based systems have been adopted in many situations that require prompt establishment of system diagnostics and execution of reaction plans, such as in monitoring of complex systems. This article describes the Query Analyzer and Manager (QAM) module, a first effort toward the development of a CEP as a Service (CEPaaS) system. This module is responsible for analyzing user-defined CEP queries and for managing their execution in distributed cloud-based environments. Using a language-agnostic internal query representation, QAM has a modular design that enables its adoption by virtually any CEP system.**

Keywords*—*Complex Event Processing, Reconfigurable Systems, Automatic Query Optimization.

I. Introduction

Current trends in sensor and mobile technologies have been generating massive amounts of data that require innovative approaches to be processed and understood. This has motivated the creation of new technologies, of which Complex Event Processing (CEP) is one prominent example, to obtain insights from high-volume, high-velocity continuous streams of data.

Complex Event Processing is defined as a "*set of tools and techniques for analysing and controlling the complex series of interrelated events that drive modern distributed information systems*" [1]. CEP-based systems interpret input data as a stream of events and accept user definitions of queries (or rules) to derive semantically enriched "complex" events from a series of simpler events. These complex events can then be used to trigger actions, thereby enabling prompt establishment of system diagnostics and execution of reaction plans. Due to these characteristics, CEP has been adopted in many situations that require fast autonomous response, such as network monitoring, and smart building management.

This article is a first effort toward the development of a cloud-based CEP system designed to be consumed as a service. Such a *CEP as a Service (CEPaaS)* system has been conceptualized to leverage public cloud environments to implement elasticity and to provide low-latency processing of data coming from diverse physical locations. Users simply define CEP queries using a graphical environment, with the system transparently managing their execution.

This article introduces QAM (Query Analyzer and Manager), a module responsible for analyzing user-defined CEP queries and managing their execution. QAM uses a formalism based on attributed graphs and graph-rewriting rules to model queries and to implement four important tasks required by CEP systems: single-query optimization, multi-query optimization, operator placement, and autonomic runtime query management. Because of its modular design, QAM can be adopted by any CEP system as long as its query definitions can be transformed into the language-agnostic query representation used by QAM.

This paper is structured as follows: Section II offers an overview of related work. Section III discusses the notion of offering CEP in the service model. Section IV is dedicated to QAM and the four operations for which it is responsible. Finally, Section V presents conclusions and future research directions.

II. Related work

The basis of the CEP field was established by classical systems, such as Aurora [2] and STREAM [3], but their centralized architecture makes them inappropriate for the CEP as a Service scenario. Cloud environments have been targeted in more recent works such as TimeStream [4] and StreamCloud [5]. However, these are mostly based on clustered architectures, whereas this research envisions a CEP system distributed over wide-area networks. For further information, the survey by Cugola and Margara [6] provides a comprehensive review of modern and historical CEP systems.

CEP queries have usually been defined by means of specific proprietary languages such as CQL [7]. Despite standardization efforts [8], a huge variety of query description languages are still in use today. The adoption of a language-agnostic representation by QAM is motivated by this variety. The surveys by Eckert *et al*. [9] and by Cugola and Margara [6] provide a comprehensive review of CEP languages.

Most CEP systems deal with some of the aspects that should be handled by the QAM module, including query optimization, operator placement, multi-query optimization, and dynamic runtime management. For instance, Kalyvianaki *et al*. [10] described a query planner which was capable of executing multi-query optimization, and Abadi *et al*. [11] showed dynamic reconfiguration actions in the context of the Borealis system. Nevertheless, none of these studies has used a unified and integrated approach to handle all four tasks as proposed in QAM.

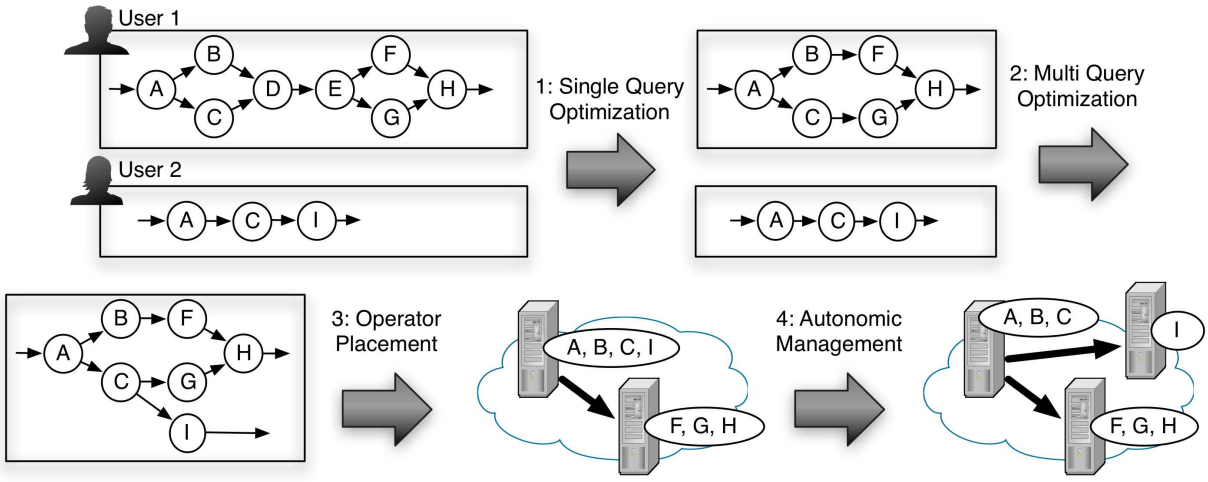

Fig. 1. Query Analyzer and Manager steps.

III. CEP as a Service

A *CEP as a Service* system aims to bring its users the functionalities of CEP associated with the many advantages of the service model, such as: (1) No up-front investment in hardware and software infrastructure; (2) Low maintenance cost, as the service model reduces the need for infrastructure monitoring, maintenance, and backup execution; (3) constant upgrades, mostly without interruption and at no charge.

Nevertheless, offering such a service involves many challenges, which is reflected in the limited number of similar services today. For instance, low latency is essential to many CEP use cases, but is difficult to achieve in a service environment because there is no control over the locations of the event sources and consumers. CEP use cases may also require a high-available service and may impose an unpredictable and variable load over it, requiring the implementation of innovative elasticity capabilities in the CEPaaS engine.

It is argued here that a distributed deployment over resources available in public cloud environments can be highly beneficial for such a *CEP as a Service* system. A distributed design improves system scalability because it enables allocation of user queries over many available servers. It also enhances the overall system availability because there is no single point of failure. In addition, use of public cloud resources also makes the system easily accessible by any client with Internet access. Finally, it also enables exploration of the inherent elasticity of cloud environments, facilitating fast allocation and de-allocation of computational resources according to system load.

IV. Query Analyzer and Manager

Implementation of a *CEP as a Service* system is a complex task involving the definition of many components and policies. This section introduces the *Query Analyzer and Manager* (QAM), which is the module responsible for analyzing user queries and for managing their execution.

Figure 1 informally illustrates the typical lifecycle of a QAM-managed query. The user initially defines a query using a CEP language, which is transformed into an attributed Directed Acyclic Graph (DAG) and sent to the QAM module. In the DAG format, the query goes through a pipeline of three tasks that results in mapping the query execution into a set of distributed components. Then QAM manages the runtime execution of the query and the evolution of the system deployment, responding to context changes such as hardware and software failures. The following subsections detail the QAM tasks.

A. Single-query optimization

Single-query optimization is the action of modifying a query to improve its efficiency (w.r.t. some optimization criterion such as CPU usage, network consumption, or processing latency) while keeping its functional properties unchanged. In Fig. 1, this process is illustrated by removal of the (redundant) operators D and E in the "User 1" query.

These transformations are executed at design time right after a new query is created and registered by the user. In consequence, this step assumes no *a priori* knowledge of available resources or of network and server states.

B. Multi-query optimization

Multi-query optimization consists in finding overlaps (common partial results) between queries and merging them into a single execution graph. This step usually aims to optimize the same criteria as the single-query optimization step. It can be executed as a separate step when a new query is created or as part of the operator placement step. In Fig. 1, the execution graphs are merged into a single one because both queries share the same processing represented by the operators A and C.

C. Operator placement

Operator placement is the task of mapping each query execution into the set of available computational resources. In the context of a cloud-based CEP system, this translates into deciding which datacentre will execute each query, the number

and types of server required in each datacentre, and how the query operators should be split among multiple processors. Note that this step implies choices at two different levels of granularity. First, a datacentre must be chosen, typically using aggregated metrics. Then placement is conducted within the selected datacentre.

This step is executed during initial system deployment, whenever a new query is registered, and in general, whenever a reconfiguration requires a placement decision (e.g., when an operator is duplicated to parallelize its execution, the placement routine is called to decide where the created operator instance should be deployed). In Fig. 1, execution of operators A, B, C, and I is mapped into one server, whereas the other operators are mapped to another server.

The QAM also uses attributed graphs to represent the available cloud servers and the network topology that interconnects them. Therefore, operator placement can be performed using classical approaches that map graphs into a substrate graph (e.g., [12]).

D. Autonomic management

Autonomic management refers to the self-managed evolution of the system in runtime. Here, queries are reconfigured in response to context changes such as violations of monitored parameters, hardware and software failures, and evolving client requests. These transformations are executed in runtime and may require performing a complex set of actions. In Fig. 1, a new server is provisioned during this step, to which the execution of the operator I is migrated.

E. Realization

Being both formal and visual, graphs have been successfully used to model the structural constraints and properties of a vast range of systems in multiple fields, including software architectures. Remarkably, graphs rewriting techniques can be used to design correct-by-construction frameworks for the specification of system transformations [13]. Such techniques have been used in particular for autonomic management of various systems such as machine-to-machine scenarios[14].

Note that both single and multi-query optimization as well as autonomic management implies system transformations. Each of these can be enforced using similar techniques, specifically by defining transformations stemming from an existing graph-based framework. Use of such a representation is supported by the fact that several CEP languages, such as CQL [7], can easily be translated into directed attributed graphs.

V. Conclusions

This work has presented the Query Analyzer and Manager, a generic CEP module that can be used to analyze user-defined queries and manage their execution. QAM represents queries and the deployment environment as directed attributed graphs and uses graph transformations to express reconfigurations and other dynamic aspects of the system. In particular, QAM is responsible for completion of four important CEP-related tasks: single-query optimization, multi-query optimization, operator placement, and autonomic runtime query management.

In the future, the authors plan to develop, implement, and evaluate each step of the *QAM* using graphs and graph transformations. Note that the mechanisms implied in the various transformational steps are very similar to each other. In particular, the authors believe that transformations and policies can be built in an iterative fashion, in the sense that single-query optimization transformations (e.g., operator duplication) can still be relevant in multi-query optimization that and in turn can be used in the dynamic (autonomic) management task.

References

[1] D. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*, 1st ed. Addison-Wesley Professional, 2002.

[2] D. J. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *The International Journal on Very Large Data Bases*, vol. 12, no. 2, pp. 120–139, Aug. 2003.

[3] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "STREAM: The Stanford Data Stream Management System," Stanford InfoLab, Technical Report 2004-20, 2004. [Online]. Available: http://ilpubs.stanford.edu:8090/641/

[4] Z. Qian, Y. He, C. Su, Z. Wu, H. Zhu, T. Zhang, L. Zhou, Y. Yu, and Z. Zhang, "TimeStream: Reliable Stream Computation in the Cloud," in *Proceedings of the 8th ACM European Conference on Computer Systems*. New York, New York, USA: ACM Press, 2013, p. 1.

[5] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez, "StreamCloud: A Large Scale Data Streaming System," in *2010 IEEE 30th International Conference on Distributed Computing Systems*. Ieee, 2010, pp. 126–137.

[6] G. Cugola and A. Margara, "Processing flows of information: from data stream to complex event processing," *ACM Computing Surveys*, vol. 44, no. 3, pp. 1–62, Jun. 2012.

[7] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121–142, Jul. 2005.

[8] N. Jain, S. Mishra, A. Srinivasan, J. Gehrke, J. Widom, H. Balakrishnan, U. Çetintemel, M. Cherniack, R. Tibbetts, and S. Zdonik, "Towards a Streaming SQL Standard," *Proc. VLDB Endow.*, vol. 1, no. 2, pp. 1379–1390, Aug. 2008.

[9] M. Eckert, F. Bry, S. Brodt, O. Poppe, and S. Hausmann, "A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed," in *Reasoning in Event-Based Distributed Systems SE - 3*, ser. Studies in Computational Intelligence, S. Helmer, A. Poulovassilis, and F. Xhafa, Eds. Springer Berlin Heidelberg, 2011, vol. 347, pp. 47–70.

[10] E. Kalyvianaki, W. Wiesemann, Q. H. Vu, D. Kuhn, and P. Pietzuch, "SQPR: Stream query planning with reuse," in *2011 IEEE 27th International Conference on Data Engineering*. Ieee, Apr. 2011, pp. 840–851.

[11] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J.-H. Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the Borealis stream processing engine," in *Proceedings of the Second Biennial Conference on Innovative Data Systems Research (CIDR'05)*, 2005, pp. 277–289.

[12] I. Houidi and D. Zeghlache, "Exact adaptive virtual network embedding in cloud environments," in *2013 IEEE 22nd International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2013, pp. 319–323.

[13] D. Hirsch and U. Montanari, "Consistent transformations for software architecture styles of distributed systems," *Electronic Notes in Theoretical Computer Science*, vol. 28, no. 0, pp. pp.4–25, 2000.

[14] C. Eichler, G. Gharbi, N. Guermouche, T. Monteil, and P. Stolf, "Graph-based formalism for machine-to-machine self-managed communications," in *2013 IEEE 22nd International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, 2013, pp. 74–79.