Transformation of Pervasive Raw Data Into Actionable Knowledge*

Sharma Chakravarthy
IT Laboratory and Department of Computer Science & Engineering
The University of Texas at Arlington, Arlington, TX 76019.
Email: sharma@cse.uta.edu

Abstract

This paper presents an architecture for addressing the transformation of large volume of data generated by pervasive and other sensor applications in a timely manner (or subject to quality of service or QoS constraints). As the size and rate at which raw data is generated increases, novel approaches for meaningful reduction and extraction of actionable information from the data are needed. In this short paper, an architecture, based on the synthesis of stream and complex event processing (CEP) approaches, is presented. The proposed architecture is flexible and accommodates the use of a number of alternative or complementary techniques/approaches.

1. Introduction

What is an effective architecture for situation monitoring applications? That is, applications in which very large amounts of raw data (presented in real time or near-real time) need to be meaningfully aggregated, correlated, and reduced to identify actionable knowledge and notified. How do we progress towards a flexible, composable architecture for analyzing large volumes of data? These questions certainly need to be answered as we move towards pervasive data management and just-in-time monitoring requirements. The amount of data that will be generated, and hence need to be processed is only going to increase dramatically as the technology improves. Application requirements will grow to include distributed data handling, handling of errors, privacy issues, security as well as collaboration/cooperation among applications. Filtering, fusion, aggregation, and correlation (to name a few) will become preferred mechanisms for dealing with vast amounts of disparate raw data to extract nuggets of meaningful and useful knowledge.

*This work was supported, in part, by NSF grant IIS 0534611

2. Looking Ahead

As an answer to the above questions, the general architecture of situation monitoring applications is depicted in Figure 1. Ability to convert very large amounts of raw data into actionable knowledge and in real-time (or near real-time) will form the cornerstones of these architectures. Figure 1 shows the components at the subsystem level. The inverted triangle is symbolic of the data reduction/aggregation/fusion/correlation process through a multi-stage process using which nuggets of actionable knowledge are extracted.

There may be several layers of data reduction and correlation. The initial reduction of raw data is likely to come from a computation-intensive process (whether it is stream processing, fusion, mining, or some other mechanism is not so critical) which gives rise to a number of high-level, domain/application-specific "interesting events". The results from this stage can be used directly or stored for later analysis. These "interesting events" are further composed as dictated by the application semantics to detect higher-level "situations". This process of generation of "interesting events" and correlation may be repeated several times until the desired level of abstraction is reached.

It is important to understand that stream processing [1, 2, 3, 4, 5, 6] is only one approach for converting large amounts of raw data into "interesting events" or results that are useful in their own right. This layer can change/evolve, in a number of ways including newer techniques, over time. The complex event processing [7, 8, 9, 10, 11, 12, 13, 14] layer is also likely to undergo changes (as we have already witnessed over the course of 20+ years). Together, the components indicated Figure 1, will provide the functionality needed for situation monitoring applications for a long time.

3. `MavEStream`: An Integrated Architecture

The proposed integrated architecture is elaborated in Figure 2. Our architecture consists of four stages:

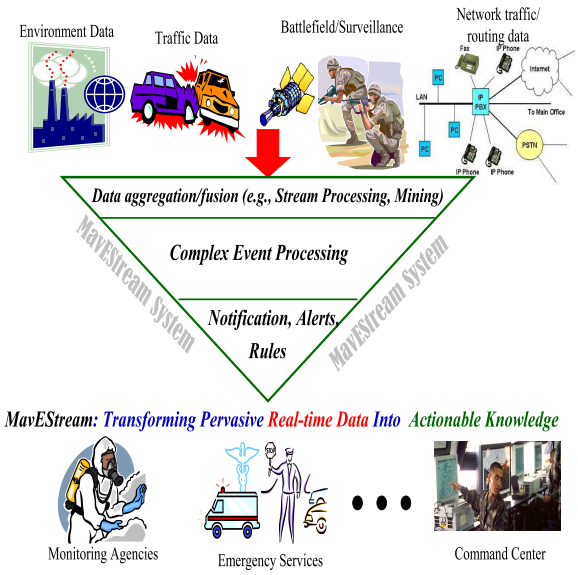

Figure 1. Architecture of Situation Monitoring Applications

Continuous Query (CQ) Processing (Stage 1): corresponds to the CQ processing of data streams. This stage represents the `MavStream` (or any other stream processing system) that accepts stream data as input, computes CQs, and produces output streams. In this discussion, we assume that CQs output data streams in the form of tuples. `MavStream` currently supports a number of stream operators: `select`, `project`, `join`, `aggregation operators` (`sum`, `count`, `max`, `min`, and `average`) and `group by`. `Having` functionality is implemented using a filter operator on the output of `group by`. Stream modifiers have been proposed as part of continuous queries. Output from a CQ can be consumed by an application; if needed, the output is also propagated to the event processing stage through the event generation interface.

Event Generation (Stage 2): generates events based on the association of `computed` events (with or without masks) with CQs. Evaluation of masks is also performed in this stage. In addition to the extensions to both systems (further elaborated below), **stage 2** has been added to facilitate seamless integration of the two systems. This stage allows for stream output to be split to generate different event types from the same CQ.

Complex Event Processing (Stage 3): represents the complex event processing component, where complex events are detected based on the definition of event expressions. `Computed` events generated by CQs act as primitive events. In the integrated architecture, `computed` events are raised by the event generation stage.

Rule Processing (Stage 4): is a component of CEP system which processes rules that are associated with events. When events are detected, conditions (specified as methods) are evaluated and if they evaluate to `true`, corresponding actions are performed. Events and rules can be added at run-time. Rules can raise events resulting in cascaded execution of event detection and rule execution.

3.1 Strengths of the Architecture

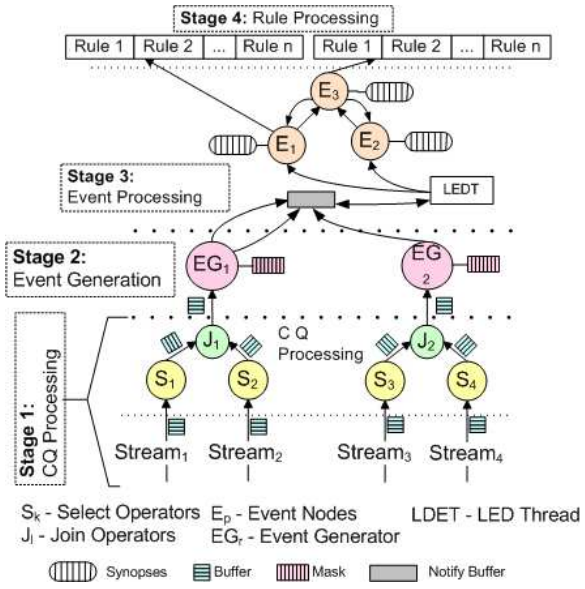

Figure 2. Multi-Stage Stream and Complex Event Integration Model

The architecture presented in Figure 2 is motivated by the separation of semantics and roles while preserving the expressiveness and efficiency of computation. Optimization of individual components will be much easier (and some work already exists in this direction) than optimizing the combined specification as a whole. Also, from the users' perspective, understandability and manageability of CQs and event specifications will be much easier than either a procedural approach or composing already overloaded constructs (or operators).

The seamless nature of our integrated architecture is due the compatibility of the chosen event processing model (i.e., an event detection graph) with that of the stream processing model used for stream processing.

We want to emphasize that the architecture shown above does not limit the applications to one stage of stream processing followed by one stage of complex event processing. We want to unequivocally indicate that there is no such arbitrary restriction in the proposed architecture. Incoming streams can be treated directly as

events, if necessary, by feeding streams to the event generation operator in **stage 2**. If there are no masks (or filters) associated with an event, the event generator node acts as a `no-op`. Analogously, as the output of a complex event is a stream, it can either act as an input to another CQ, an application, or propagate to detect higher-level complex events. In summary, seamless coupling of the two capabilities allows one to process large amounts of raw data in one or more stages of CQ and CEP combination to derive higher-level abstractions or knowledge. Furthermore, at any point in this process, rules associated with events (primitive or complex) can trigger actions and notifications. Arbitrary composition of stream and complex event processing is readily accommodated by the proposed architecture. Each subsystem can be used to its full potential without additional overhead and the functionalities can be combined as needed. It is even possible (in this architecture) to associate rules with CQ nodes if needed. All the functionality for supporting this capability already exists.

The separation of stream and CEP, and their coupling has a number of additional advantages. Common computations can be performed by a continuous query and the output can be split into different types of events of interest for the purpose of event processing. For example, *slowing down of a car* is a common computation that can be performed by a CQ and the output can be split into multiple event types based on the lane characteristic (regular or HOV). This facilitates modularization and reduces the number of continuous queries (or subqueries) in the system there by improving QoS. Similarly, composite event detection can be added at any time to the output of CQs in the system. Output of CQs can also be input selectively to the event processing subsystem. For example, "if detecting the slowing down of cars and accidents during rush hours is more important", this can be easily accommodated by defining masks on an event for the same CQ.

4. Summary

We have implemented the above architecture using existing components to demonstrate the flexibility of the proposed approach. Other approaches/technologies can be added, integrated, or replaced, relatively easily, to suit the needs of the application and data source requirements.

References

[1] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. B. Zdonik, "Monitoring Streams - A New Class of Data Management Applications," in *Proceedings of the International Conference on Very Large Data Bases*, August 2002, pp. 215–226.

[2] S. Madden and M. J. Franklin, "Fjording the stream: An architecture for queries over streaming sensor data," in *Proceedings of International Conference on Data Engineering*, April 2002, pp. 05–55.

[3] Q. Jiang and S. Chakravarthy, "Data stream management system for MavHome," in *Proceedings, Annual ACM SIG Symposium On Applied Computing*, 2004, pp. 654–655.

[4] Q. Jiang and S. Chakravarthy, "Scheduling Strategies for Processing Continuous Queries over Streams," in *Proceedings of the Annual British National Conference on Databases*, 2004, pp. 16–30.

[5] J. Chen, D. Dewitt, F. Tian, and Y. Wang, "Niagaracq: A scalable continuous query system for internet databases," in *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, August 2000, pp. 379–390.

[6] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, approximation, and resource management in a data stream management system." in *Conference on Innovations in Database Research*, 2003.

[7] D. L. Lieuwen, N. H. Gehani, and R. Arlein, "The Ode Active Database: Trigger Semantics and Implementation," in *Proceedings of International Conference on Data Engineering*, March 1996, pp. 412–420.

[8] S. Gatziu and K. R. Dittrich, "Events in an Object-Oriented Database System," in *Proceedings of Rules in Database Systems*, September 1993, pp. 23–39.

[9] H. Branding, A. P. Buchmann, T. Kudrass, and J. Zimmermann, "Rules in an Open System: The REACH Rule System," in *Rules in Database Systems*, 1993, pp. 111–126.

[10] S. Chakravarthy *et al.*, "Design of Sentinel: An Object-Oriented DBMS with Event-Based Rules," *Information and Software Technology*, vol. 36, no. 9, pp. 559–568, 1994.

[11] A. Dinn, M. H. Williams, and N. W. Paton, "ROCK & ROLL: A Deductive Object-Oriented Database with Active and Spatial Extensions," in *Proceedings of International Conference of Data Engineering*, 1997, pp. 491–502.

[12] S. Chakravarthy and D. Mishra, "Snoop: An expressive event specification language for active databases," *Transactions of Data Knowledge and Engineering*, vol. 14, no. 1, pp. 1–26, 1994.

[13] C. Roncancio, "Toward Duration-Based, Constrained and Dynamic Event Types," in *Active, Real-Time, and Temporal Database Systems*, 1997, pp. 176–193.

[14] R. Adaikkalavan and S. Chakravarthy, "SnoopIB: Interval-based event specification and detection for active databases," *Transactions of Data Knowledge and Engineering*, vol. 59, no. 1, pp. 139–165, 2006.