Semantically Partitioned Peer to Peer Complex Event Processing

Filip Nguyen, Daniel Tovarňák, and Tomáš Pitner

Abstract. Scaling Complex Event Processing applications is inherently problematic. Many state of the art techniques for scaling use filtering on producers, vertical scaling, or stratification of an Event Processing Network. The solutions usually aren't distributed and require centralized coordination. In this paper, we are introducing a technique for scaling Complex Event Processing in a distributed fashion and by taking semantic information of events into account. We are introducing two CEP models for scaling CEP architectures, providing core algorithms, and evaluating their performance.

1 Introduction

In this paper, we are concerned with a new way of scaling Complex Event Processing (CEP) applications. This section introduces CEP, presents the motivation for our work, and introduces our contribution.

1.1 Complex Event Processing

CEP is both a theoretical and a practical research area that studies events and event processing in current computing systems and businesses. Examples of such events may be:

- A payment using a credit card. This may be regarded as relatively infrequent event.
- A barcode reading of a product code. This may be regarded as a frequent event.
- A motion sensor on electronic doors to a supermarket.

Filip Nguyen · Daniel Tovarňák · Tomáš Pitner
Faculty of Informatics, Masaryk University,
Brno, Czech Republic
e-mail: {xnguyen,xtovarn,tomp}@fi.muni.cz

F. Zavoral et al. (eds.), *Intelligent Distributed Computing VII,*
Studies in Computational Intelligence 511,
DOI: 10.1007/978-3-319-01571-2_8,

All of these events may be correlated together and may thusly cross both technological and domain boundaries. The process of correlating is referred to as *pattern matching*. Such a pattern might be: two payments made by the same credit card in different supermarkets within a time frame of 1 hour.

Another possible view of Complex Event Processing is an upside down version of standard data processing. Instead of data sitting in a database, waiting for queries to be submitted, CEP may be viewed as queries sitting and waiting for data to be submitted. That is why CEP is used as a monitoring tool ([4], [5]). An interesting method of using CEP as a monitoring was introduced in [2], where it is usable even for data mining.

Motivation for CEP nicely correlates with the current advent of Big Data, which is a data centric approach to extracting meaningful data. The problem here is not to store such data, but to retrieve it and to extract meaningful information [1]. CEP is able to carry out extractions of data regarding time, a frequent component in data. In CEP, data that is not processed is simply discarded.

CEP was motivated and introduced in a very comprehensive publication [3]. The book provides a basic framework for many CEP usages, including dynamic complex event processing. The book introduces the so called *Event Processing Agent* (EPA). EPA is defined as an object that monitors events to detect certain patterns in them. Another abstraction is then introduced in the form of an *Event Processing Network* (EPN) which connects many EPAs into a network where each EPA sends events to another EPA. EPNs are added to the model for flexibility and allow for the easy reuse of EPAs and for the building of dynamic EPNs. In these dynamic EPNs, processing agents may enter or leave the network at the time of processing.

1.2 Peer to Peer Complex Event Processing

We exploit the fact that events from certain producers are related to each other in some way. We do this by breaking up the event space by identifying related producers using CEP itself. We then exploit this information and put processing only where it is most needed. For example, knowing that two retail stores were visited by the same customer in the last two days signifies some connection between the two retail stores. With information that the producers are related, we are able to add event processing between these two retail stores. This event processing between these two stores may consider much more fine grained events. By using this approach we may lose some information because fine grained analysis are not done everywhere. A similar probabilistic view of CEP was also discussed in [6] using an example of a hypothetical surveillance system that is monitored using CEP.

To show how we help scale CEP, we are introducing two graph oriented event processing abstractions: *peer model* and *centralized model*. In both models, we view a graph of CEP engines and producers as a basic building block for creating scalable event processing architecture.

To uncover possible correlations between producers, we use *partitioning algorithms*. These two algorithms help to identify which producers should have the engine among them (using the centralized model). We later also map this centralized approach to the peer model.

We have developed two partitioning algorithms. Their properties are discussed and they are studied on an experimental basis in a simulated CEP environment.

2 State of the Art

In this chapter we are introducing current models of Complex Event Processing that relate to our research. The word ”distributed” is being attached to CEP in many publications, but its meaning varies. In [3] the interpretation is: the processing of heterogeneous events, generated from various sources in an enterprise environment. In this environment, events are generated very often and they travel via various channels (a private computer network, the Internet) to be processed.

There are several approaches to scaling CEP. A straightforward way is by analyzing bottlenecks in current CEP systems. The authors of [9] did extensive profiling of the Borealis CEP system and identified that communication is one of the biggest bottlenecks. Another approach was introduced in [10] where traditional query planning was applied to CEP and optimizations. Sliding window push downs were successfully applied. Scalability via hierarchy was introduced in [2].

Stratification is a particularly interesting way of scaling CEP. It was introduced in [12]. Stratification significantly improves event processing capabilities by parallelizing certain matching operations. The input for a stratification algorithm is EPN with dependency information. Dependency information states which EPA depends on the input of another EPA. Using a simple algorithm, the authors were able to cleanly separate EPAs into sets called strata, which contain agents that can run in parallel.

In this paper, we understand distributed CEP as distributed collection, distributed processing and most importantly, distributed matching of events.

3 Peer to Peer Complex Event Processing

In this section we will introduce our CEP models. We will also discuss the *event space correlation problem* as well as our approach to solve it.

Our notation has two parts: graphical representation and query language.

First we will look at graphical representation. Figure 1 shows the basic building blocks of our model. The black node represents the centralized engine that accepts events. The producers of the events are represented by red nodes with light rings. The producers only produce events and do nothing else. Red nodes with bold rings

Fig. 1 CEP Graph Model

represent peers. A peer is a CEP engine that is placed directly on a producer and replaces the centralized engine in our model. Our notation further includes an event which may be augmented with time information pertaining to its creation as well as with the name of the producer.

In our model, we denote the set of producers, event engines, and peers using capital P, e.g. $P_1, P_2, ..., P_N$. Set of events is denoted E and discrete events may be denoted by syntax $name : time : producer$. Events are described using two functions: $time : E \rightarrow N$ for the time when the E originated and $producer : E \rightarrow P$ to get the producer of the event. The producers may be connected in an undirected graph and a producer may send some of his events to his neighbor. The function $events : P \times P \rightarrow 2^E$ is used to retrieve a set of events that have traveled between the two producers. The event engine is a node in our graph that, upon reception of an event (E_0 signifies incoming event), tries to use this event to match a pattern. Patterns MP are represented by a simple SQL like syntax, e.g. WHERE E_0=E_1 WINDOW(10). The WINDOW part of SQL syntax denotes the time frame in which the matching takes place.

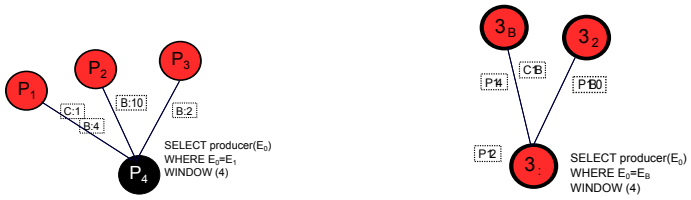
Function $patterns : P \rightarrow MP$ returns subset of patterns running on the peer.

Function $matches : MP \times N \rightarrow 2^E$ returns matched set of events matched by a pattern since specified time.

There are two possible ways to model an event processing situation using these building blocks: peer model and centralized model. Centralized model uses centralized engines and producers. In this case, events flow from the producers to the engines. Figure 2a shows a simple CEP network where producers P_1, P_2, P_3 send their events to P_4. The P_4 in this example contains only one matching rule. This rule fires only once when event $B : 4$ arrives to P_4. The $B : 10$ doesn't match the rule because the sliding WINDOW is only 4 time units in this case.

The second possible way to model an event processing situation is *peer model*. The basic building block is peer: both a producer and an event processing engine. In this peer model, a graph is formed only by peers and it is possible to achieve the same matching capabilities as with the centralized model. In Figure 2b, you can see an example of peer model notation. P_3 works both as the producer and the engine that works instead of P_4. Note that P_3 also produces events back to itself.

Using the centralized model is better when explaining the concepts. On the other hand, the peer model resembles our targeted implementation more as well as introduces additional advantages.

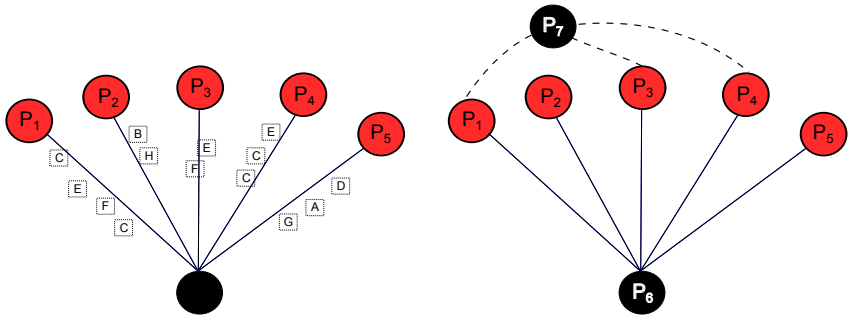


(a) Simple centralized CEP (b) Simple Peer CEP

Fig. 2 CEP Models Comparison

3.1 Event Space Correlation Problem

Suppose we are building a CEP solution for 5 producers which are producing events. Figure 3a shows such a situation. Suppose we want the engine to answer to the following query: SELECT $producer(E_0)$, $producer(E_1)$ WHERE $E_0 = E_1$ WINDOW(10).



(a) Centralized CEP (b) Dynamic CEP deployment

Fig. 3 CEP Models Comparison

Consider the technological implementation of P_6. To be able to match the equality, the CEP engine needs to accumulate all of the events in the time frame and continually analyze whether a new event is correlating with any of the other events thus far accumulated. Because all of the events are related to each other, no stratification is possible here. This problem is common among CEP engines and was also discussed in [2], where the authors used rule allocation algorithms in their distributed CEP model. They considered a structure of rules to circumvent the problem. A different approach to solving a special case of this problem is to design a special data structure to hold shared events [8].

We have exploited the observation that producers are related. In the given example on Figure 3a, we can see that producers P_1, P_2, P_4 are correlating together more

than the other producers. The fact that these 3 are related in some way can be exploited by placing a new P_7 engine between them. This engine will have to consider a lower number of events and thus its performance will also be better.

Methods for detecting some related producers are discussed further in this paper. We call these methods *partitioning algorithms*.

In this section we discussed the dynamic placement of engines between producers and we made use of a classic centralized CEP model. This facilitates easy understanding, but it is important to note how this can be reinterpreted into our peer model. In the peer model, we do not deploy additional engines, but we only connect the appropriate engines and assign roles. Figure 4 shows an example of peer model. The peer P_5 in this situation assumes the role of P_6 from the previous example. When our algorithms decide that P_1, P_2, P_3 are related, the three peers connect (dashed line) and P_3 takes responsibility of correlating events between them. Let us portray the example in real world scenarios.

One example might be the detection of network attacks [13]. Computers and network elements are the producers of network logging information. It is possible, by monitoring coarse grained events in a network, to find out that some limited set of computers is vulnerable to attack. Using this knowledge, we can deploy an additional engine among the limited number of producers and process more fine grained events.

Another example would be CEP engine used for monitoring public transportation [14]. Sensors in vehicles are producers. By detecting the coarse grained event of "user comfort by temperature", we can detect that some vehicles are good candidates to be closely analyzed and we can deploy an additional engine in the vehicle computer itself and analyze more fine grained events (speed changes, vehicle technical data, time table precision of the driver).

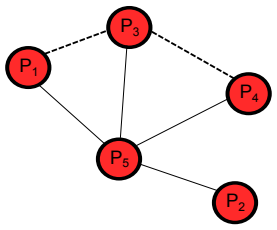


Fig. 4 Peer Model

The peer architecture of Complex Event Processing has some obvious downsides. In the real world, it may be challenging for a producer to be able to see every other producer, e.g. every supermarket server may not have access to every other supermarket server. Some event patterns might not be detected when the partitioning algorithm doesn't connect the correct producers.

However, there are many advantages of this architecture. The previously mentioned scaling capabilities. It is easy to dynamically connect several nodes and

thusly begin processing among them. Also, because producers feature an event processing engine, it is possible to easily filter events. In addition to that each added producer will improve the power of the whole system because of the addition of another processing engine.

3.2 Partitioning Algorithms

We are introducing and comparing two partitioning algorithms: the CEP based algorithm and the Monte Carlo Partitioning Algorithm. The algorithms dynamically add/remove CEP engines among peers to conduct pattern matching.

The CEP based algorithm uses complex event processing of coarse grained events (less frequent, not consuming too many resources) to later deploy more fine grained events. The present matching patterns are configurable by the user and should be prepared for a specific purpose. The algorithm uses *centralized model.*

We have used **wait** in the algorithms, which signifies waiting either for a specific guard or to wait for a specified time. While this wait is in progress, matching continues in the CEP environment. This waiting happens only to the partitioning algorithm.

The first algorithm has matching patterns as an input (MP_1, MP_2, MP_3), together with a graph representation of producers (P) and edges between them (channels). The algorithm divides the producers into two subsets (P_{left} and P_{right}) and continually checks whether the division into these two subsets is valid. To do the check it uses *differ significantly* - this check can be substituted to any configurable set difference algorithm. We use a simple set difference with threshold of 25% difference.

INPUT: COARSE GRAINED MP_1. FINE GRAINED MP_2, MP_3. CEP GRAPH $G(P, channels)$
2. $channels := P_c \times P$; $patterns(P_c) = \{MP_1\}$; T:=0;
3. **WAIT** UNTIL $|\{p|\exists e \in matches(MP_1, T).p \in producer(e)\}| >= |P|/2$
   $X := \{p|\exists e \in matches(MP_1, T).p \in producer(e)\}$
   T:= CURRENT TIME
   $channels := channels \cup \{(x, P_{left})|x \in P\} \cup \{(x, P_{right})|x \in P/X\}$
   $patterns(P_{left}) := \{MP_2\}$
   $patterns(P_{right}) := \{MP_3\}$
4. SAME WAIT AS IN (3)
   IF $\{p|\exists e \in matches(MP_1, T).p \in producer(e)\}$ DIFFER SIGNIFICANTLY FROM X **GOTO** (2)
   ELSE **GOTO** (4)

The Monte Carlo algorithm uses a centralized engine at the beginning. It derives the interesting producers and correlates with the current set of matching rules. It then divides the event space. A big advantage of this algorithm is that it doesn't need any input from the the CEP user. It works with matching patterns which have already been provided for the centralized version of CEP. The algorithm divides producers to two sets *SH* and *SL* based on the vector *STAT*.

INPUT: EXISTING MATCHING PATTERNS MP. T:=0. CEP GRAPH $G(P, channels)$
1. RANDOMLY SELECT $MP_{small} \subset MP$;
2. $channels := P_c \times P$; $patterns(P_c) = MP_{small}$; T:=0

3. **WAIT** DEFINED TIME. THEN $\forall x \in matches(MP_{small}, T)$
   $STAT[producer(x)]$**+=1**
   LET SH, SL BE SET OF PRODUCERS. $|SH| - |SL| < 1 \wedge \forall x \in SH, \forall y \in SL.STAT[x] >= STAT[y]$
   $channels := channels \cup \{(x, P_{left}) | x \in SH\} \cup \{(x, P_{right}) | x \in SL\}$
   $patterns(P_{left}) := MP$; $patterns(P_{right}) := MP$
4. T:= CURRENT TIME; CONSTRUCT STAT2 SAME WAY AS IN (2)
   IF STAT2 FROM STAT DIFFERS SIGNIFICANTLY
   GOTO (2)
   ELSE **GOTO** (4)

3.3 Evaluation

We have implemented and measured these two algorithms in a simulated event processing environment. This environment consisted of 100 event producers, each producing up to 10 000 event types at random times from a non-uniform distribution. The experiment was divided into two time periods.

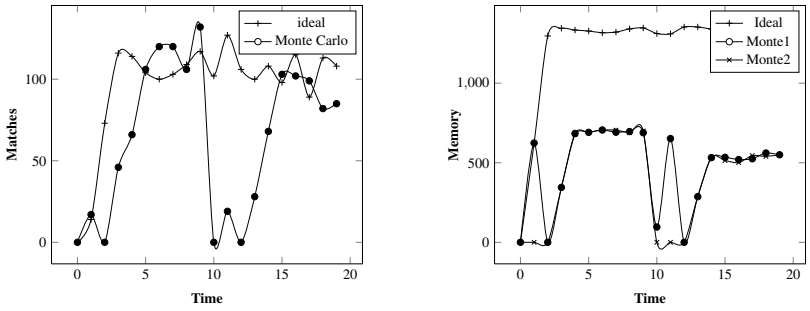
In the first period, a random 20% of the producers were related. They produced only 1 000 event types.

In the second time period, a different 20% of producers were chosen to be related.

As we can see from Figure 5a, the Monte Carlo is even better than ideal pattern matching at a point. This is due to the fact that the two engines operating in the parallel display had an overall better performance than the one overloaded ideal engine - from time to time.

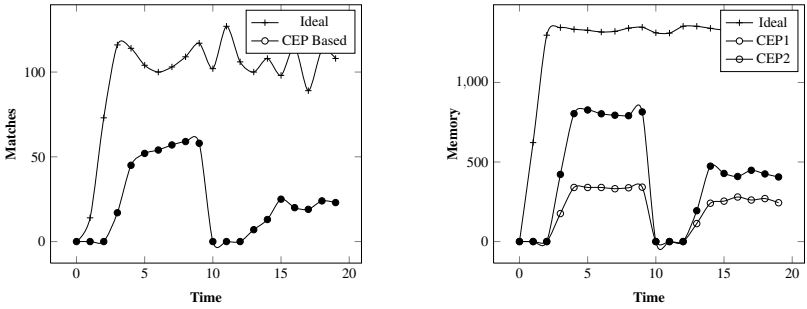
Figure 6 shows that ideal CEP engine has significantly higher matching capabilities over the CEP based solution. While disturbing, after deeper thought this is understandable. The CEP Based solution relies on the user to define the rule for dividing the producers. In a simulated environment this is hard to do, however we believe that this performance will be improved in more realistic scenarios.

From Figure 7, we can see that both algorithms drop in matching performance when different events become related. However, the Monte Carlo is better in overall matching capabilities. On the other hand, the CEP based algorithm provides better



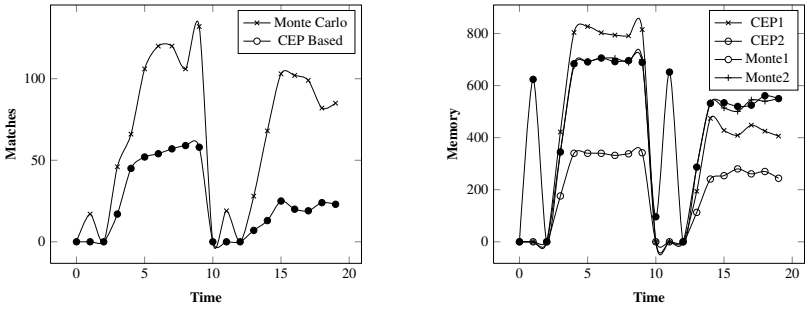
(a) Pattern Matches (b) Memory Consumption

Fig. 5 Ideal Engine vs Monte Carlo



(a) Pattern Matches (b) Memory Consumption

Fig. 6 Ideal Engine vs CEP based Partitioning Algorithm



(a) Pattern Matches (b) Memory Consumption

Fig. 7 Monte Carlo vs CEP based Partitioning Algorithm

overall memory characteristics because it selects smaller event spaces to consider, and it also uses coarse grained events to decide where to put fine grained engines. This operation doesn't consume much memory. The Monte Carlo, on the other hand, has to work as a centralized and memory overloaded engine.

4 Conclusion and Future Work

In this paper, we have introduced a simplified view of Complex Event Processing called *peer model*. Peer model aims to simplify the way of looking at CEP to easily introduce conceptual scaling optimizations. With this model, it is easier to think about distributed event collection and processing. We have also described practical mappings of this model in real world situations. Further, we have identified *the event space correlation problem* that stops current CEP models from scaling conceptually. We have introduced two partitioning algorithms and evaluated their performances in a simulated environment. The proposed partitioning has a semantic nature in a sense, that algorithms consider inner relations among events to uncover possible

relations between producers. Experimental results show that the Monte Carlo partitioning algorithm performs better with regards to matching capabilities, but suffers from memory consumption. The CEP based algorithm provides an overall better memory performance and is easier to deploy into high volume environments, but has significantly lower matching capabilities.

In the future, we plan to extend the peer model from a query semantics perspective. We will introduce distributed algorithms that will allow the deployment of CEP rules into the peer model and also allow for the collection of matching results from it. Additional studies of partitioning algorithms are also needed. We hope that the CEP based algorithm may be improved to give better matching performance. All of these efforts are aimed to create an Open Source platform Peer CEP that is written in the Java Programming language. The simulator which was used to conduct experiments for this paper will also be part of the suite - for testing and research needs.

References

1. Jacobs, A.: The pathologies of big data. Communications of the ACM - A Blind Person's Interaction with Technology CACM Homepage Archive 52(8), 36–44 (2009)
2. Van Renesse, R., Birman, K.P., Vogels, W.: Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. ACM Trans. Comput. Syst. 21(2) (2003)
3. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems, 376 p. Addison-Wesley Professional, New York (2002)
4. Tovarňák, D.: Towards Multi-Tenant and Interoperable Monitoring of Virtual Machines in Cloud. In: 14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania (2012)
5. Nguyen, F., Pitner, T.: Information System Monitoring and Notifications Using Complex Event Processing. In: Proceedings of the Fifth Balkan Conference in Informatics, Novi Sad, pp. 211–216. ACM, Serbia (2012) ISBN 978-1-4503-1240-0
6. Artikis, A., Etzion, O., Feldman, Z., Fournier, F.: Event processing under uncertainty. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems (DEBS 2012), pp. 32–43. ACM, New York (2012)
7. Isoyama, K., Kobayashi, Y., Sato, T., Kida, K., Yoshida, M., Tagato, H.: A scalable complex event processing system and evaluations of its performance. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS 2012. ACM, New York (2012)
8. Lee, S., Lee, Y., Kim, B., Candan, K.S., Rhee, Y., Song, J.: High-performance composite event monitoring system supporting large numbers of queries and sources. In: Proceedings of the 5th ACM International Conference on Distributed Event-based System, DEBS 2011, pp. 137–148. ACM, New York (2011)
9. Akram, S., Marazakis, M., Bilas, A.: Understanding and improving the cost of scaling distributed event processing. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems, DEBS 2012, pp. 290–301. ACM, New York (2012)
10. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD 2006, pp. 407–418. ACM, New York (2006)

11. Randika, H.C., Martin, H.E., Sampath, D.M.R.R., Metihakwala, D.S., Sarveswaren, K., Wijekoon, M.: Scalable fault tolerant architecture for complex event processing systems. In: International Conference on Advances in ICT for Emerging Regions (ICTer), Colombo, Sri Lanka (2010)
12. Biger, A., Etzion, O., Rabinovich, Y.: Stratified implementation of event processing network. In: 2nd International Conference on Distributed Event-Based Systems. Fast Abstract, Rome, Italy (2008)
13. Minařík, P., Vykopal, J., Krmíček, V.: Improving Host Profiling with Bidirectional Flows. In: International Conference on Computational Science and Engineering, CSE 2009, August 29-31, vol. 3, pp. 231–237 (2009)
14. Kaarela, P., Varjola, M., Noldus, L.P.J.J., Artikis, A.: PRONTO: support for real-time decision making. In: Proceedings of the 5th ACM International Conference on Distributed Event-Based System, DEBS 2011, pp. 11–14. ACM, New York (2011)