Towards Trustworthy Complex Event Processing

Hua Chai and Wenbing Zhao
Department of Electrical and Computer Engineering
Cleveland State University, Cleveland, Ohio 44115
Email: wenbing@ieee.org

***Abstract*—Complex event processing has become an important technology for big data and intelligent computing because it facilitates the creation of actionable, situational knowledge from potentially large amount events in soft realtime. Complex event processing can be instrumental for many mission-critical applications, such as business intelligence, algorithmic stock trading, and intrusion detection. Hence, the servers that carry out complex event processing must be made trustworthy. In this paper, we present a threat analysis on complex event processing systems and describe a set of mechanisms that can be used to control various threats. By exploiting the application semantics for typical event processing operations, we are able to design lightweight mechanisms that incur minimum runtime overhead appropriate for soft realtime computing.**

Keywords*—*Complex Event Processing, Business Intelligence, Big Data, Dependable Computing, Trust, Byzantine Fault Tolerance

I. Introduction

Complex event processing (CEP) has become an important technology for big data and intelligent computing because it facilitates the creation of actionable, situational knowledge from potentially large amount events in soft realtime. CEP can be instrumental for many mission-critical applications, such as business intelligence, algorithmic stock trading, and intrusion detection. Hence, the servers that carry out complex event processing must be made trustworthy. By trustworthy, we mean the following two properties:

- The server must be continuously available. Streams of events generated by various event producers will continuously arrive at the CEP server, and they may trigger important decisions regarding business directions, buying and selling stocks, or the detection of an intrusion. If the CEP server becomes unavailable due to hardware failures, power outages, or cyber attacks, such decisions will not be made in time, which may result in significant financial losses.
- The server must ensure its integrity under cyber attacks. An adversary might attempt to subvert the integrity of the CEP server instead of crashing it, because it could lead to more severe damages. For example, if instead of selling a stock, a wrong decision on buying the stock is made, the user of the application could incur higher loss than if he/she does not sell at the time due to the unavailability of the CEP server.

Byzantine fault tolerance (BFT) is a promising technology to enhance the trustworthiness of complex event processing [1]. BFT can be used to protect a server against Byzantine faults, which denote various faults including hardware faults as well as software faults as the result of cyber attacks. The core mechanisms of Byzantine fault tolerance include:

- Replication. The server is replicated so that even if a small portion of the replicas become faulty, the remaining replicas can continue providing correct services to the clients.
- Byzantine agreement. To ensure continuous availability as well as service integrity of the server, the state of nonfaulty replicas must remain consistent. This requires the use of a Byzantine agreement algorithm to guarantee the following in the presence of Byzantine faults:
  - All nonfaulty replicas deliver and execute incoming requests in the same total order.
  - All nondeterministic operations, if any, are rendered deterministic so that given the same request executed in the same order, all nonfaulty replicas go through the same state transitions.

Well-known BFT algorithms such as PBFT [2] and Zyzzyva [3] are designed for general client-server applications with synchronous request-reply communication between the client and the server. Complex event processing, on the other hand, involves drastically different communication styles between the replicated server and other components (details will be given in Section II). Furthermore, CEP systems are time sensitive and an extended delay in processing may render the resulting decision obsolete. BFT algorithms usually have no provision for this scenario. Hence, they cannot be used directly as they are designed.

In this paper, we present a set of lightweight mechanisms to enhance the trustworthiness of CEP systems. We first present a threat analysis on CEP systems. We then describe a set of mechanisms to control most of the threats by exploiting the semantics of complex event processing. The core of our mechanisms is a lazy state synchronization scheme. State synchronization relies on the use of Byzantine agreement to ensure that all nonfaulty CEP server replicas generate the same alert for event consumers. A round of synchronization is run only when an event consumer fails to accept an alert message, which minimizes the runtime overhead during normal operation.

This paper makes the following contributions:

- We present a comprehensive threat analysis on CEP systems. In particular, we highlight the threats on the validity and consistency of replicated CEP servers.
- We introduce a lazy synchronization mechanism to ensure that nonfaulty CEP server replicas use the same

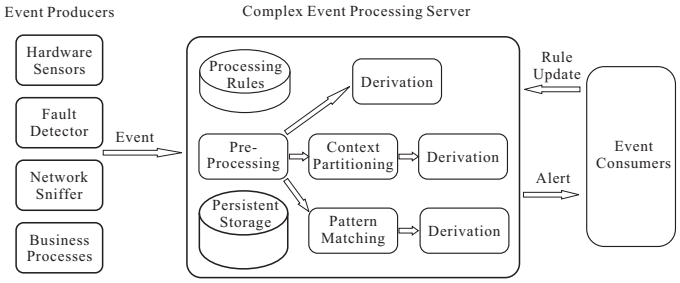

Fig. 1. Overview of a complex event processing system.

context partition for pattern matching and alert derivation in the presence of malicious event producers.

- We employ a voting mechanism at the event consumer to be resilient against malicious CEP server replicas as well as faulty event producers.
- We use an evidence-based mechanism to validate each synchronization request to prevent a malicious event consumer from inducing unnecessary rounds of state synchronization.

II. Complex Event Processing

A CEP system consists of three major components, the event producers, the event processing server, and the event consumers, as shown in Figure 1. Event producers are the entities that produce events. Examples of event producers include hardware sensors that measure the environment, fault detectors that monitor the health of various processes and generate an event whenever the failure of a process is detected, network sniffers that report important network events, and business processes that generate events (such as an order has been placed or a stock trading transaction has been completed) based on business logic. Event consumers are normally decision making processes that are interested in receiving higher level reports (referred to as alerts) from the CEP server. The use of a CEP server frees the event consumers from the burden of directly handling multiple streams of events and derive the high level situational reports.

The CEP server is usually powered by specialized event stream processing frameworks such as Esper[1]. Normally, incoming events are first pre-processed according to predefined criteria and non-qualified events are discarded. During the server deployment, and also during runtime, the event consumer could register and update event processing rules. The server processes the incoming events according to the process rules for each event consumer.

Event processing may be stateless, where each event is processed independently from other events. As shown in Figure 1, for stateless processing, the derivation step occurs right after the pre-processing step. During the derivation step, an event might be translated, enriched, or projected into another event for the event consumer to handle [4]. These stateless operations manipulate the attributes of the incoming events.

More complex event processing are stateful in that multiple events are involved. Stateful operations usually belong to one of the following two types:

- Aggregation based: In this type of operations, attributes of a sequence of events possibly belong to different event streams are computed to derive higher level information. For example, the average, maximum, and minimum values of the collection of events may be computed.
- Pattern matching based: In this type of operations, certain predefined event patterns are being matched against the incoming events. Such a pattern usually means an important situation, such as an intrusion occurred.

For stateful event processing, typically the operation is applied to a subset of the events in one or multiple streams. The subset is determined by the context as defined in the processing rule. The events may be partitioned for context formation based on a number of criteria:

- Time: The context is determined based on the timestamps of the events. Events that belong to one or more time intervals may belong to a temporal context. Different temporal contexts may overlap. In fact, sliding windows are one of the most common temporal contexts used in complex event processing.
- Location: The context is determined based on the location where events occurred.
- Identifier: Events may contain an identification attribute, such as customer id, that can be used to partition the events to different contexts.
- State: Which events are included in a context depends on the external state. The external state may in turn be temporal, spatial, or identifier based.

The communication pattern between different components of a CEP system is drastically different from that of client-server systems. In client-server systems, the client typically sends requests to the server for processing, and blocks waiting for the corresponding replies before it issues the next one. In CEP systems, on the other hand, different components communicate via one-way messages:

- An event producer continuously streams events to the CEP server and does not expect any reply.
- When an event consumer wants to update or create a new processing rule, it sends a one-way request to the CEP server.
- The CEP server sends an alert message to the designated event consumer whenever one is generated at the end of the derivation step.

III. Threat Analysis

In this section, we analyze potential threats that could compromise the trustworthiness of complex event processing. In particular, we discuss new threats that arise due to the replication of the CEP server. We do not consider general threats that could target any online services, such as distributed denial of service attacks. We divide the threats under consideration

[1] http://esper.codehause.org

into three categories based on the originator of the threat: (1) threats imposed by a faulty event producer; (2) threats imposed by a faulty CEP server; and (3) threats imposed by a faulty event consumer.

A. Threats Imposed by a Faulty Event Producer

A compromised event producer could attack the CEP system in the following ways:

- Omit important events.
- Emit wrong events.
- Send conflicting events to different CEP server replicas when the server is replicated.

All these attacks could lead to wrong alerts or the absence of alerts, which could ultimately lead to wrong decisions or indecisions made at the event consumers. The third attack is a specific attack when the CEP server is replicated and its objective is to cause the CEP server replicas to generate inconsistent alerts, which could confuse the event consumers.

Because all derivations at the CEP server hinge on the trustworthiness of the events, it is essential to adopt the following principals when designing a CEP system:

- Use robust pattern matching and/or derivation algorithms so that the presence of a single faulty event or the absence of an event would not adversely impact the matching or derivation result. For example, taking the medium value of the attributes of a collection of events is more robust than taking the average value.
- Avoid rely on a single event producer for input. By using multiple event producers for the same type of input, the trustworthiness of the events may be assessed by correlating the events.

B. Threats Imposed by a Faulty CEP Server

A compromised CEP server could attack the system by sending wrong alerts to event consumers, or refusing to send alerts as it should to event consumers. Again, both attacks could lead to wrong decisions or indecisions made at the event consumers.

When the CEP server is replicated, a compromised CEP replica could also attack other replicas aiming to cause non-faulty replicas to generate wrong alerts or not to generate an alert when it should.

To control the first type of threats, the CEP server must be replicated and the alerts sent by different CEP replicas must be voted on at each event consumer. It is more complex to control the second type of threats because Byzantine agreement must be used to ensure the consistency of nonfaulty replicas.

C. Threats Imposed by a Faulty Event Consumer

The event consumer is typically run as a client-side application that interacts with users directly. Hence, a compromised event consumer may confuse the user in arbitrary ways. To control this type of threats, a host-based intrusion detection system would have to be used, which is out of the scope of this paper.

A compromised event consumer may also attempt to attack the CEP server in the following ways:

- Change the processing rules for other event consumers stored at the CEP server.
- When the CEP server is replicated, the event consumer could attack the replication mechanism by inducing unnecessary rounds of state synchronization.

The first type of threats can be controlled by using robust authentication mechanism so that an event consumer cannot access the processing rules that belong to another event consumer. The second type of threats can be controlled by requiring the inclusion of non-forgeable proof for the need of synchronization.

IV. Trustworthy Complex Event Processing

In this section, we present detailed counter measures for trustworthy complex event processing. First, secure messaging must be used for communication between different components of the system. This ensures that the sender is properly authenticated (*i.e.,* one sender could not impersonate as another sender) and the integrity of the message is protected (*i.e.,* a tempered message can be detected). The event messages sent by the event producers are protected by message authenticated code. All other messages exchanged between the replicas and between each replica and the event consumer are protected by digital signatures for accountability.

The validity of complex event processing in the presence of faulty event producers is ensured by using redundant event producers and robust matching and derivation algorithms. The number of faulty event producers that can be tolerated by a CEP system depends on the algorithms used.

The consistency of complex event processing at different replicas can be achieved trivially by using a Byzantine fault tolerance algorithm such as PBFT. However, doing so would mean the total ordering of all event messages, which would incur too much runtime overhead to satisfy the soft realtime requirement for CEP systems. Hence, we choose to use a lazy state synchronization scheme. In the absence of faulty event producers, no state synchronization is necessary, hence, minimizing the runtime overhead.

We require the use of $3f+1$ CEP replicas to tolerate up to f faulty replicas. Each replica is assigned a unique id k, where k varies from 0 to $3f$. For each context partition P with events e_1, e_2, ... e_n, a CEP replica maintains a history hash $h_P = hash(e_1||e_2||...||e_n)$, where $hash()$ is a secure hash function such as SHA-1 or MD5. For stateless event processing, the context partition is reduced to a single event. An important property for the history hash is that it is independent from the relative ordering of the events, *e.g.,* $h(e_1, e_2) = h(e_2, e_1)$. If an alert is produced for a context partition, the corresponding signed history hash is piggybacked with the alert as proof of the context formation.

An event consumer may encounter one of three scenarios:

- It receives $2f + 1$ or more matching alert messages from different replicas. Such alerts are accepted.

- It receives at least $2f + 1$ alert messages but fails to receive $2f + 1$ matching alert messages before a predefined timeout occurs. In this case, the event consumer initiates a round of state synchronization by broadcasting a synchronization request to the CEP replicas and piggybacking the set of alert messaged collected as the proof for the need of synchronization.
- It fails to receive $2f + 1$ alert messages no matter how long it waits. This may happen in the presence of faulty event producers or faulty CEP replicas. No action is taken in this case.

On receiving a synchronization request, a CEP replica validates the set of signed messages by verifying their signatures. The CEP replica launches a round of synchronization for the context partition by broadcasting to other replicas a state-sync message via a Byzantine agreement service. The state-sync message contains the id for the context partition and the set of events in the local context partition (if the event message is long, only the event id and the digest of each event are included). The replica also collects the state-sync messages sent by other replicas. Upon receiving the first $2f + 1$ totally ordered state-sync messages sent by different replicas, a CEP replica proceeds to building a consistent context partition CP according to the following rules:

- If the same event is included in all $2f + 1$ state-sync messages, it is included in CP.
- If the same event is included in at least $f + 1$ state-sync messages, but no conflicting event is found in the messages, it is included in CP.
- If two or more records are detected with identical event id, but different event digests, they must have been sent by a faulty event consumer. Such events are excluded from CP.

Once CP is built, the CEP replica checks to see if its own context partition is identical to CP. If not, it replaces its own context partition with CP, and repeats the matching and derivation step. It is guaranteed that the alert messages generated using CP by different nonfaulty CEP replicas are consistent, which ensures the acceptance of the alert message at the event consumer.

V. Related Work

Replication of CEP servers has been used to improve their availability against crash faults [5]. Speculative execution and commit order have been employed to ensure concurrent execution of actively replicated CEP servers that are designed with software transactional memory. Previously, we extended [5] to tolerate Byzantine faulty CEP server replicas by using a BFT algorithm [2] to totally order all event messages and the derivation transactions [6]. There are several major differences between [6] and the current work:

- In [6], all event messages are totally ordered using a BFT algorithm. However, in our current work, a round of Byzantine agreement is needed only when an event consumer fails to accept an alert message, which would incur much less runtime overhead.
- In our current work, a comprehensive threat analysis on the CEP systems is presented, while in [6], no such analysis is carried out.
- The mechanisms described in [6] only work for CEP systems designed with software transactional memory, while our current work can be applied to any CEP systems.

This work is a continuation of our work on application-aware Byzantine fault tolerance [7], [8], [9], which is to design optimal mechanisms by exploiting the semantics of each type of applications. In our current work, we exploited the fact that the context formation is the basis for stateful event processing and designed a lazy synchronization mechanism to enhance the trustworthiness of CEP systems.

VI. Conclusion

In this paper, we discussed how to enhance the trustworthiness of CEP systems. We first presented a comprehensive threat analysis on CEP systems. In particular, we highlight the threats on the validity and consistency of replicated CEP servers. We then outlined the principles on controlling various threats on CEP systems, and introduced a set of mechanisms to ensure trustworthy complex event processing in the presence of malicious faults. The core of our contribution is a lazy synchronization mechanism that allow nonfaulty CEP servers to determine a consistent context partition for pattern matching and alert derivation. The mechanism is lazy because a round of synchronization is initiated only when an event consumer fails to receive sufficient number of matching alerts, and it is not needed in the absence of faulty event producers, which minimizes the runtime overhead during normal operation.

References

[1] W. Zhao, *Building Dependable Distributed Systems*. Wiley-Scrivener, 2014.

[2] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 398–461, 2002.

[3] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," in *Proceedings of 21st ACM Symposium on Operating Systems Principles*, 2007.

[4] O. Etzion and P. Niblett, *Event Processing in Action*. Manning Publications, 2010.

[5] A. Brito, C. Fetzer, and P. Felber, "Multithreading-enabled active replication for event stream processing operators," in *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*. IEEE, 2009, pp. 22–31.

[6] H. Zhang and W. Zhao, "Concurrent byzantine fault tolerance for software-transactional-memory based applications," *International Journal of Future Computer and Communication*, vol. 1, no. 1, pp. 47–50, 2012.

[7] H. Chai, H. Zhang, W. Zhao, P. M. Melliar-Smith, and L. E. Moser, "Toward trustworthy coordination for web service business activities," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 276–288, 2013.

[8] H. Chai and W. Zhao, "Byzantine fault tolerance for session-oriented multi-tiered applications," *Int. J. of Web Science*, vol. 2, no. 1/2, pp. 113–125, 2013.

[9] H. Zhang, H. Chai, W. Zhao, P. M. Melliar-Smith, and L. E. Moser, "Trustworthy coordination for web service atomic transactions," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 8, pp. 1551–1565, 2012.