Research Article

A Novel Complex Event Processing Engine for Intelligent Data Analysis in Integrated Information Systems

Dong Wang, 1 Mingquan Zhou, 1, 2 Sajid Ali, 3 Pengbo Zhou, 2 Yusong Liu, 4 and Xuesong Wang 2

1 School of Information Science and Technology, Northwest University, Xian 710127, China
2 College of Information Science and Technology, Beijing Normal University, Beijing 100875, China
3 Department of Computer Science, University of Education, Lahore 60000, Pakistan
4 School of Information, Renmin University of China, Beijing 100872, China

Correspondence should be addressed to Mingquan Zhou; mhzhou_nwu@yahoo.com

Received 19 November 2015; Revised 31 January 2016; Accepted 24 February 2016

1. Introduction

In recent years, the capacity and usage of information systems have been increasing. These are integrated together primarily due to business processes [1]. The data volume generated by a variety of heterogeneous sources has increased around the real world [2, 3]. As a result, it is important that systems need to efficiently collect and analyze the huge amount of data and events in real time to discover meaningful results.

Data analysis is a process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision-making [4]. It is increasingly vital for the success of business information systems. Therefore, there is a need of extremely efficient and flexible data analysis platforms to manage and process such data sets [5]. Traditional methods are not suitable for the massiveness and variety of data. As a result, a new efficient solution for data processing is imminently needed.

Some reliable approaches are available for data processing [6, 7]. The well-known MapReduce paradigm has been widely used and experienced by both academia and enterprise. It is a programming model and software framework developed by Google. It simplifies the processing of vast amounts of data in parallel on large clusters of commodity hardware in a reliable, fault-tolerant manner [6]. However, Hadoop and MapReduce are not suitable for processing event streams. They were designed for offline batch processing of static data, in which all input data needs to be stored on a distributed file system in advance. Storm is another efficient solution [7]. It is a distributed, reliable, and fault-tolerant system, which is used particularly for processing continuous real-time data.
streams. It adopts a default scheduler, which employs a simple round-robin algorithm based cluster, without considering the internode and interprocess traffic, and it may have a significant impact on performance [8]. Besides this, the default scheduler in Storm often uses all available nodes in a cluster, regardless of workload. For the light workload of input stream, the operational cost can be reduced by using a single node in a cluster [9].

Therefore, a solution for real-time processing of large-scale web data stream with less cost should satisfy four requirements. They are as follows: (1) The local agent is required to get real-time data from information systems and deliver events to an engine with high throughput. It must be easily integrated in information systems without major changes. (2) A high-performance engine is a key point for the solution. An efficient event processing algorithm is capable of processing large volume of incoming events. It could be adopted by engine to improve the performance in real-time processing. (3) Event transmission mechanism for delivering events should reflect the up-to-date state of the actions and operations in information systems. The mechanism is expected to achieve high throughput for transporting events from multisources to CEP engine. (4) Well designed system architecture is flexible and scalable in development and deployment. It is a combination of the mentioned agent, event transmission mechanism, and engine.

However, the volume of data created by integrated information systems is enormous, which is beyond the processing capabilities of the system. A new solution is needed to process the continuous data in real time as well as discover the exceptions, threats, and actionable information behind all data, indicating that the Complex Event Processing (CEP) is a good candidate. Applying CEP approach could provide an adequate solution: enable access to real-time information while providing advanced query capabilities and minimal perturbation [10, 11]. Therefore, CEP could be used in intelligent data analysis for enabling real-time processing and situation detection consequently leading to a new quality of system.

In general CEP applications, engine usually processes events according to the rules and detects different situations during events. Rules are defined by system administrator in build time. It is a component of CEP engine for saving the composite constraints of complex event. The general architecture of such CEP applications is shown in Figure 1. It is working in three steps. Firstly, the streams of incoming events are captured by CEP engine (run time). Secondly, event patterns and CEP rules defined by users (build time) are implemented in CEP engine to analyze the event streams and detect meaningful data (run time). Finally, immediate actions are triggered in CEP engine (run time).

According to the traditional CEP technology, we aim to build lightweight, modularized, and extensible architecture to improve the data analysis capability in integrated information system by CEP approach. In particular, the need of a novel CEP engine to deal with a large number of input events from integrated systems is obvious. Starting from these premises, a lightweight intelligent data analysis system (LIDAS) is proposed. It has a CEP engine LIDA-E, which is developed to provide event processing services. In order to represent the relationship and operation of events, we propose event models with operators to describe event rule through Event Processing Language (EPL). CEP engine is an important component, which adopts an efficient event processing algorithm. It translates rule statement into rule instance for processing input events. Inside the system, we introduce a data knowledge base used for business data knowledge storage and presentation as well as a rule base presented for rules storage. A distributed event transmission mechanism is presented for event delivery from information systems (agents) to CEP engine (filter).

As a solution, we propose LIDAS so that the administrator (has no knowledge on CEP) can concentrate on the definition of event rules without the need of handwriting any code. The contributions of this paper can be summarized as follows:

(i) Event models define the semantics of both atomic and complex events. They allow easy adaptation to information system needs and are used to describe event aggregation in event processing rules.

(ii) The high-performance CEP engine LIDA-E employs the event processing algorithm and knowledge base is introduced for intelligent data analysis in real time.

(iii) A distributed event transmission mechanism is proposed by agents (deployed in information systems) and filter (deployed in LIDA-E engine) explicitly conceived to realize easy and efficient event delivery.

Figure 1: General architecture of CEP application.
(iv) We discuss the design and implementation of LIDAS architecture, which is an integration of LIDA-E engine and event transmission mechanism.

The rest of the paper is organized as follows. In Section 2, we discuss related works of CEP technology. Section 3 describes the relevant concepts of LIDAS, such as event model, operators, and SQL-like Event Processing Language. The proposed architecture is presented in Section 4. Section 5 shows the experimental results. Finally, in the last section, we summarize our research work.

2. Related Work

CEP is considered as extracting complex situations and reacting on them from massive events [12]. Meanwhile, CEP is not a new terminology, as Luckham first introduced [13]. It has been widely exploited during the past years mostly in big data analytics [5], RFID [14], Internet of Things (IoT) [15], failure prediction systems [16], real-time grid monitoring [11], and healthcare [17]. According to CEP, different algorithms have been proposed to increase Complex Event Processing capability. They adopt CEP engine (like Storm [7], Drools [18], and Esper [10, 19]) in their applications, while others design new event processing engines and system architectures [11, 12, 15, 16, 20] with evaluation to show the usefulness and scalability.

Storm is a free and open source distributed real-time computation system, which makes it easy to reliably process unbounded streams of data in real-time processing by batch operations [7]. It includes spouts and bolts, which could be executed with many tasks in parallel on multiple machines (worker nodes) in a cluster [8]. Storm is designed to process unbounded streams of data in a storm cluster (master node and worker nodes). However, as discussed by Xu et al. [8], Storm weakly focuses on the performance and job assignment with different workload. Moreover, the deployment of Storm requires more nodes.

Differing from Storm, Drools [18] and Esper [10, 19] are CEP engines which include a module providing native support for events evaluation and temporal logic analysis in a single work node. Yao et al. [18] presented a Drools based CEP framework which was used to process surgical events and provided sense and response capability for hospitals. In the framework, the author mentions that the knowledge with CEP is quite complicated, most of which comes from experts and is fuzzy and hard to verify the accuracy. To combine knowledge with CEP system, Bruns et al. [10] describe a novel event-driven architecture for a decision support system which leads to a new quality of M2M systems, which are intelligent and flexible. The system adopts Esper as the core event processing component and employs a rules base to explicitly represent the M2M knowledge of domain experts. Therefore, CEP could be used as an appropriate candidate with rule base for intelligent data analysis.

However, with the development of web technology, the open source engines become less efficient to deal with huge volume of data. In order to get higher performance, some special CEP engines have been developed for particular domains (LiSEP [12], DPCEP [15], and RTE-CEP [20]). Zappia et al. [12] described the design and implementation of a lightweight and extensible Complex Event Processing engine, called LiSEP, for sensing and responding applications. The author adopted the Staged Event-Driven Architecture (SEDA) principles to clearly separate core event processing logic from lower level resource management issues. However, event model and operators are not designed and used for LISEP, as well as event collection. The event processing method of LISEP is not clearly described in their research.

Combining CEP with distributed systems, GEMINI2 [11] is proposed as a custom framework dedicated for CEP based real-time monitoring of grid infrastructures. In GEMINI2, 100 sensors loaded 400 events in one second, and the performance of the CEP engine was not evaluated. The performance of working nodes and servers could not support massive events transmission.

In summary, most CEP system architectures used in existing distributed environment are not oriented towards real-time processing for integrated information systems. The prevailing CEP engines are not to explore the performance of event processing but rather expose the event processing methods, while the engines are suitable for processing event streams and are not well suited for analyzing massive data in integrated systems. The work described here is a result of the previous investigations in CEP and intelligent data analysis in distributed network.

3. CEP for Intelligent Data Analysis

3.1. Application Scenario. A Project Management Information System (PMIS) is one important type of Computer-Based Information Systems (CBIS) and is used to collect, process, analyze, and publish the data for a particular purpose [21]. Project management (PM) is a knowledge-centric and experience-driven activity supported by an appropriate PMIS [22]. PMIS is an integrated information system that helps project manager to carry out projects systematically and monitor progress of projects closely.

A project life cycle is divided into many states, such as proposal, review, implementation, and acceptance states. Starting from project proposal to implementation, Taghavi et al. [23] proposed a web based project management functional model that separates project work progress management into time, fund, quality, contract, bidding, demand, integrity, comprehensiveness, handover, and accomplishment. Each project state is supported by different subsystems, which are combined together as a PMIS. According to project states, we propose a project state model which shows the state changes in a project life cycle. In Figure 2, “−” is a start state and “+” is an end state; others are middle states (proposal, review, implementation, and acceptance); they represent the whole life cycle of a project. In different project states, systems verify \( E^{\text{Verify}} \) the reasonability and validity of massive project data. If the project data passes the verification, a state transition event \( E^{\text{declare}}, E^{\text{review}}, E^{\text{implement}}, E^{\text{accept}}, \) and \( E^{\text{complete}} \) changes the state from one to another. Otherwise, a failed event happens and changes the state to start.
According to the events attributes in PMIS, constraints are necessary for analyzing data availability and validity when events occurred by user activities. Some rules for data analysis are elaborated in the following. Rule 1: monitor the active state of a subsystem. Rule 2: examine the document examination time of a project in one day. Rule 3: examine whether the number of projects of a manager is above a defined value.

3.2. Event Model. Luckham and Schulte [24] defined an event, as there is a large amount of events occurring inside or outside the system during multiple phases such as making a business transaction, receiving an email, creating a sales report, or uploading a file. Events are extracted from web services, user activities, system log, database, and so on. They are separated into atom and complex; atom events are identified through data collection, and complex events are detected by the analysis of atom event stream [25, 26].

Definition 1 (atom event). An atom event means that a special activity or data transition occurs in a point in time, which is represented by a six-tuple. The elements of the six-tuple are as follows:

Atom Event: \( ae = (AEid, Pid, A, S, T, t) \) .

\( AEid \) is the identification of an atom event. \( Pid \) presents the identification of a project. \( A \) is the set of project attributes and data. \( S \) is a source from where the event is generated. \( T \) is event type and \( t \) is timestamp which identifies when the atom event occurs. In the project management system example, each user activity or operation in system generates an atom event. Atom events usually represent the state of a business attribute or some changes.

Definition 2 (complex event). It is a combination of atom events or complex events consisting of the predefined rules, such as constraints, logical relationship, and time sequence combination. It is also denoted by a six-tuple:

Complex Event: \( ce = (CEid, A, C, T, t_b, t_e) \) .

\( CEid \) is the identification of a complex event. \( A \) is a combination of the atom events and complex events that trigger this event to happen, where \( A = \{ae_1, ae_2, \ldots, ae_n, ce_1, ce_2, \ldots, ce_m\}, n + m > 0 \). \( A \) is a set of attributes. \( A = \{ae_1, A, ae_2, A, \ldots, ae_n, A, ce_1, A, ce_2, A, \ldots, ce_m, A\} \). \( t_b \) is the starting time and \( t_e \) is ending time of complex event, where \( t_e >= t_b \). A complex event is a combination of atom and complex events by pattern matching rules, that is, a specific set of event operators. We will introduce the event operators and rules in the following subsection.

**Definition 3** (event type). The set of all similar events is called the event type \( T \). It shows the object which triggers the event. Hence,

\[
E^T = \{e_1^T, e_2^T, \ldots, e_i^T, \ldots, e_n^T\} = \bigcup_{i=1}^{n} e_i^T .
\]

Events in an event type have the same attributes of \( T \). For instance, the type may refer to events regarding information verification; however, the other event properties may be different (e.g., source or attribute). An example is the type of all events that denote the submitted information. \( E_{\text{submit}} \) shows that this event type indicates submitted information and is constituted by \( e_{\text{submit}}^1, e_{\text{submit}}^2, \ldots, e_{\text{submit}}^n \).

**Definition 4** (event space). The set of all possible events known from a certain information system is called the event space \( \hat{E} \). Hence,

\[
\hat{E} = \{E^{T_1}, E^{T_2}, \ldots, E^{T_x}, \ldots, E^{T_T}\} = \bigcup_{1}^{T} E^{T_x}
\]

\[
= \{ae_1, ae_2, \ldots, ae_n, ce_1, ce_2, \ldots, ce_m\}
\]

\[
= \left\{ \bigcup_{i=1}^{n} ae_i, \bigcup_{j=1}^{m} ce_j \right\}
\]

The event space is formed by all the sets of event types, as well as all the atomic events and complex events. For example, \( E_{\text{portal}} \) represents all the events that occurred in portal. It can be illustrated as \( E_{\text{portal}} = \{e | \forall e, e.S = "\text{portal}" \} \).

3.3. Operators. This section describes the operators for aggregating events. They are used to show the relationship among detected events. We extend them from event constructors [18, 27], which consider the aggregation demands on event composition. We present three types of operators: logical operator, mathematical operator, and temporal operator. Table 1 shows the most frequently used operators for event detection. Atom and complex events are both denoted by \( e \).

These operators are used to aggregate events and detect complex patterns that catch the meaningful information from real-time data streams. For example, an event pattern \( \text{WITHIN}(e_1 \lor e_2, 1 \text{ min}) \) means when either \( e_1 \) or \( e_2 \) occurs within time period less than 1 minute, the pattern \( \text{SEQ}(e_1 \land e_2, e_3, e_4, e_5) \land (\text{AVE}(\text{score}, e_3, e_4, e_5) > 80) \) is matched when both events \( e_1 \) and \( e_2 \) occur and then \( e_3, e_4, \) and \( e_5 \) occur continuously; meanwhile, the average score in \( e_3, e_4, \) and \( e_5 \) is greater than 80.

3.4. Event Processing Language and Rules. Event Processing Language (EPL) provides a set of patterns like filtering, correlation, applying constraints, and aggregating. These patterns define the availability of events querying and filtering. EPL statement is a description of event operators that are used to derive and aggregate information from event streams. The authors in [27] elaborated twenty EPLs to compare their
operators and consumption modes. According to the proposed event operators and consumption modes, we employ Esper EPL syntax [28] for specifying event processing rules supporting by event operators in this investigation. The EPL syntax is presented in the following:

```
select select_list from stream_def [as name] [, stream_def [as name]] [, ...].
[where search_conditions] [group by grouping_expression_list] [having grouping_search_conditions].
[output output_specification] [order by order_by_expression_list] [limit num_rows].
```

The above EPL syntax includes some query clauses that are introduced by Ahmad et al. [29]. The select clauses are used to select all properties or to specify the list of event properties and expressions. The "from clause" indicates event stream name. The "where clause" is an optional clause used to join and correlate event streams. The "group by clause" separates the output events into groups. The "having clause" is used in combination with the "group by clause" to restrict the groups of returned rows to only those whose condition is true. Adopting EPL syntax, rules declared in Section 3.1 can be represented in a standard structure as follows:

Rule 1: MissingEventAgent

```
SELECT COUNT(e.Sid) WHITIN(Require Time)
IF (COUNT(e.Sid) = 0) THEN ACTION create MissingEventAgent Warning
```

Rule 2: UserRepeatDocExamine

```
SELECT COUNT(e.Pid) WHERE e.TYPE = 'DocExamine' WHITIN(1 day)
IF ANY(e.COUNT(e.Pid)) > LIMIT_DOCEXAMINE THEN ACTION create UserRepeatDocExamine Warning
```

Rule 3: Project ManagerExamine

```
SELECT COUNT(e.Manager) WHERE e.TYPE = 'SubmitProjectInfo' \&\& e.Manager = e.Manager \&\& e.ProjectState = 'Uncompleted' V Submited'
IF COUNT(e.Manager) > LIMIT_MANAGER THEN ACTION create ManagerRepeated Error
```

We assume that the above rules provide analysis engine with rules for what to do in event streams. For this aim, the engine is intelligent to execute rules by adopting a knowledge base. Knowledge base provides intelligence for the engine as it contains rich parameters for both engine and rules.

For example, LIMIT_DOCEXAMINE is defined by LIDAS administrator in Rule 2. Its value is according to the document of examination server runtime state. Rule 2 is constructed to balance the capacity of the server to each project and detect vicious document examination request. It specifies that the trustworthiness of a document examination request should be continuously monitored. The notification is generated as soon as the value rises above a given threshold.
value. Rule 3 is made for analyzing how many uncompleted or submitted projects does a manager have in system. If the number of a manager’s projects is beyond the limitation, analysis engine will create an error action.

4. The Architecture of LIDAS

4.1. Architecture Design of LIDAS. We have to extend the capability of integrated information systems towards the intelligent data analysis by adopting advantage of event processing technologies, keeping clearly apart each peculiarity of the CEP components to perform even deep changes without affecting related integrated systems. In designing the intelligent data analysis architecture, we started by specifying the CEP components that interact with the technological architecture and traditional integrated systems.

As shown in Figure 3, LIDAS is a lightweight and modularized framework designed to connect the integrated systems for intelligent data analysis. The bottom of the framework is agents that are deployed in the traditional integrated information systems to extract data in complicated business processes. The upper part is an intelligent data analysis with CEP which deals with event processing and data analyzing. It is implemented as an intelligent processing center connected to explicit event queues in accordance with predefined event rules. System administrators define event processing rules and knowledge through user admin board in system build time. The users get analysis results from user dashboard in real time.

In order to show the workflow of LIDAS framework, we introduced agent, filter, and analysis engine to illustrate event transmit and processing (see Figure 4). First, agents send atom events to the listener module in filter. And then preprocess module reads events from listener and stores them in queue; it sends a message of event arriving to analysis engine. And the engine receives messages from filter; it enables the processing thread to access new events from queue. The graphical representation of the workflow is illustrated in Figure 4 and introduced in the following subsections in detail.

4.2. Event Transmission Component. In order to improve the scalability in large-scale integrated systems and reduce the network delay caused by online event streams delivery, almost all current event processing frameworks are based on
CEP. However, event receiving and dispatching problems are commonly ignored. Figure 3 illustrates the simplified view over the LIDAS architecture in distributed systems. The combination of filter and agent works as a bridge that transmits events from information systems to LIDA-E. They are basic components which act as an event provider to CEP engine.

We use agent which is similar to sensor [11] and adapter [30] used in monitoring system to collect information from distributed environment. Agent is responsible for extracting real-time business data and generating atom event. It is event source and is deployed in an integrated system to generate massive events with business data. Using the proposed event model, agent reads data from an interface of integrated system and translates them into atom events. According to the event model (Definition 1), agent puts data as attributes in A as a list. It generates atom event immediately when data is detected in systems.

Collaborating with CEP technology, the filter detects composite events from different information systems [27]. Filter is an event receiving and preprocessing module that handles subscription related to control coming messages from analysis engine and processes high volumes of event objects which are received from agents. According to the availability of business data, filter pushes each valid atom event into a map that is held in memory while it notifies the analysis engine that a new event has arrived and is stored in map. We summarize the functions of filter as follows: (1) receive events from distributed agents; (2) preprocess events streams according to event availability; (3) push events into memory and notify analysis engine.

Some methods, for example, TCP, FTP, SMTP, HTTP, and TELNET, can be used in event transfer method between filter and agent. However, we use a New IO (NIO) client server Netty framework in our event transmission mechanism to find a way to achieve ease of development, performance, stability, and flexibility with high throughput, low latency, less resource consumption, and minimized unnecessary memory copy. We have developed a socket client and server by Netty; client is deployed in agent and server, which is adopted by filter. By accepting socket communication protocol, filter has the capability to listen on a system port and receive events from different agents. Event transmission mechanism is the first step that collects real-time data from integrated systems; the CEP based event processing method will be introduced in the next subsection.

4.3. LIDA-E Engine. The analysis engine, called LIDA-E, is the core component of LIDAS architecture; it is responsible for processing the events queue and judging whether there is a complex event inferred [25]. It provides effective queuing, scheduling, time and count-window support, and fast in-memory processing of high-speed, continuous, unbounded data streams. In order to divide functions of engine, we design three modules in LIDA-E; they are main thread, event access, and event process. Main thread module is used to monitor filter and control event access module. Event access module is dedicated for accessing events from the queue which is written by filter. Event process module executes rules and processes all the events. It is a linear task controller that deploys rules one by one and manages events to be processed through them.

CPE component comprises a set of rules and knowledge to detect a predefined group of anomalies on the basis of the receiving event attribute which are the same in [31–33]. The rule base specifies how to infer complex events from the event stream. However, knowledge base provides the link between known information (the antecedent) and the information to be deduced (the consequent) or actions to be executed. It is a useful supplement to rule base for processing events. Both rules and knowledge are defined in build time and loaded when LIDAS starts to work.

LIDA-E workflow is shown in Figure 5; it is a runtime workflow in which the analysis engine, waiting for events arriving, has already executed rules with knowledge in event process module. The workflow follows the substeps as follows: (1) filter pushes events into queue as soon as it receives events from different agents; (2) filter sends a message to analysis engine to notify it that a new event has come; (3) main thread module receives the message and notifies event access module that new events have been pushed in queue; (4) event access module reads events from queue and prepares the submission for event process module; (5) event access module sends event to event process module immediately; (6) event process module accesses queues for event processing; in event process module, each event is orderly processed by rules; for each rule, at least one queue is provided for runtime event storage;...
Input: Input event. Output: Complex event (ce) and action.

(1) Load_Rules();
(2) Load_Knowledge();
(3) WHILE (Message == ‘EventArrive’ && EventQueue.size() > 0)
(4) AtomEventItem ae = EventQueue.shift();
(5) if (ae == null)
(6) Thread.sleep(10);
(7) continue;
(8) for (i = 1; i <= CountRules; i++)
(9) if (Rule[i].enable == True)
(10) Rule[i].getParament;
(11) ComplexEventItem ce = Rule[i].execute(ae);
(12) if (ce != null)
(13) Action(ce);
(14) AnalysisResult.add(ce);
(15) EventStorage.add(ae, ce);

Algorithm 1: Event processing algorithm.

After processing, analysis results are submitted to LIDAS users.

4.4. The Proposed Event Processing Algorithm. In this section, we present the new Complex Event Processing algorithm, which is a simplified version of the algorithm. It has been implemented in LIDA-E. Each rule is translated into rule instance and applied in event processing module. Additional constraints are applied by parameters, which are stored in knowledge base, such as LIMIT_DOCEXAMINE in Rule 2. It identifies the max value of document examination time: the instance used to process the input events instead of automation instance that was introduced in AIP Algorithm [34]. The detailed steps for engine algorithm are shown in Algorithm 1.

The key role in our approach is played by rule instances, which are used to detect complex events in LIDA-E. Event queue is a queue which stores the undisposed events in LIDA-E. At the beginning, LIDA-E loads predefined rules and knowledge (lines 1 and 2), which run in the initial state for the arrival of appropriate events. First, engine gets a message of “EventArrive” from filter and a new event arrives (line 3). If the EventQueue is not null (line 3), a variable ae (atom event) is defined and assigned by an event in EventQueue (line 4). The function shift of EventQueue not only reads event from EventQueue but also deletes the event which is read (line 4). If ae is null (line 5), the processing thread will sleep for 10 milliseconds and then go on working (lines 6 and 7). If ae is not null, it will be processed by every rule (line 8). In each rule instance, once a complex event is detected, a variable ce (complex event) will be assigned by the detected event (line 11). After checking the validity of ce, an action of ce is generated to system user (line 13), and the detected event is sent to processing result (line 14). Finally, all events would be stored in a database as history records (line 15).

The graphical representation of algorithm is shown in Figure 6. Input events are processed from rule 1 to N, in which a complex event is sent to result once it is detected. Otherwise, the events are signed by pass. Each rule instance maintains its own event processing status in a private queue. Finally, all the complex events results are collected from each rule instance.

5. Experimental Evaluations

In order to perform functional validation and performance evaluation of the proposed LIDAS architecture with LIDA-E, we have developed a simple proof-of-concept prototype system, supporting both the event transmission and data analysis capabilities, with the use of currently available open source components. Evaluating the performance of the proposed LIDAS framework is not easy as it is strongly influenced by the workload. The number of rules and events to be processed is considered in each test case [34]. It is also limited by the event transmission capability in network environment. In experiments, we evaluated both the event transmission mechanism and the analysis engine. They show the performance of LIDAS architecture. All the test is demonstrated on notebook computer having Intel Core i5-3210 M CPU 2.50 GHz and 4 GB of RAM, running 32-bit Windows 7 Professional.
Our evaluation had two main goals: (1) studying the performance of event transmission mechanism in our system architecture and (2) comparing engine LIDA-E with another common processing engine that could handle some rules described in Section 3.

5.1. Event Transmission. Event transmission refers to the procedure that an event is sent from an agent until it is received by filter. We started by using HTTP request to send data agent to filter. First, we used the PHP to develop an Apache server as filter. In experiment, the Apache server (versions 2.2 and 2.4) could receive no more than 2000 HTTP requests in one second. Moreover, we built the filter by Servlet, which is developed by Java that extends the capabilities of a server. The performance of Servlet service was similar to PHP service that receives about 1500 events per second on Tomcat 6.0 server. Additionally, we have used a special server technology Node.js [35] to improve the performance of filter. However, its performance was worse than the filter that is built by PHP and Servlet.

Unfortunately, the main technologies for accepting HTTP requests did not reach the requirement of massive events transmission in LIDAS architecture. This issue was well recognized for the complex structure of HTTP request, as demonstrated by the investigation in HTTP request message headers [36]. To address this issue, we decided to use socket to transmit event data, exploring socket client and server based on Netty 4.0.40 Final. We implemented the proposed event transmission mechanism in a prototype based on a product compliant with the JMS standard, specifically Netty, which provides a popular and powerful asynchronous event-driven network application framework [37, 38].

Netty client and server were developed in agent and filter, which were performed on two notebook computers to check the performance of event transmission capability. During our tests, the number of events sent by agent was increased from 10,000 to 100,000, and we recorded both time costs in agent and filter side. Figure 7 shows the time cost of event transmission in agent and filter.

In terms of the number of events to be transmitted, there is no apparent difference between agent and filter in event transmission time cost. And event receiving time cost of filter is a little less than event sending time cost of agent. Our experimental results show that the agent and filter have good performance to transmit huge volume of events in a short time.

5.2. Event Processing Capability. We evaluated the event processing capability of our engine compared with the Esper [39] engine. Esper has many features, including high scalability, memory efficiency, in-memory computing, SQL standard, minimal latency, and historical event analysis. It is a streaming-capable engine for processing real-time arriving variety data. It was embedded in Java, which makes the comparison of our engine easier.

The prototype LIDA-E was implemented in Java language, which is the most popular programming language in 2015 [40]. The applicability and usefulness of our approach have been evaluated by sample implementation of LIDA-E that processes events sent by filter (see Figure 5).

5.2.1. Event Filtering. The base operation of CEP engine is selecting input events as soon as the listener module receives new events. For this reason, the evaluation is started by the comparison of filtering capability. 100 different rules were deployed in both engines. The rules selected input events according to their attributes and generated new events containing the same attributes. Therefore, each event was filtered by 100 rules, and this meant the event input rate should be equal to the output rate. The filtering results of the two CEP engines are shown in Figure 8.

This evaluation highlights how LIDA-E processes input events faster than Esper with the same input events and deployed rules. As the workloads of the two engines are identical and are processed by a similar set of rules, the throughput
should theoretically be the same. We observe that both engines process huge numbers of events in a short time, particularly when the input rate increased from 5,000 to 50,000 events per second. The throughput of LIDA-E is higher than of Esper. Esper handles up to about 9,000 composite events per second as the maximum throughput. However, LIDA-E starts to drop the input events at rate of 15,000 events per second. The maximum throughput of LIDA-E is 18,000 composite events/s.

5.2.2. Data Analysis. Processing input events by rule instances is the main task of a CEP engine. So we continue to perform the experiments by comparing event processing capability of LDIA-E and Esper in complex operations. To demonstrate the performance of event processing without temporal operation, we defined Rule 2-1 by deleting the WHITIN operator of Rule 2. Five rules (Rule 2-1) were deployed in both engines in this test. Figure 9 shows the results in terms of processing time and number of input events in both engines.

These measure highlights show that the LIDA-E processes events faster than Esper when both engines deployed the same rules. Compared with Esper, LIDA-E has 48.65% averagely reduced time cost in different tests. It has apparent advantage over Esper because of its event processing algorithm. When the number of atom events was increased from 10,000 to 100,000, the event processing time was increased from 0.65 to 3.02 (seconds) in LDIA-E. However, if we used Esper, the processing time was increased from 0.86 to 7.18 (seconds).

As a second one, we have compared the performance of LIDA-E on different number of rules. This step used the definitions of Rule 2-1. We deployed 10 rules for performance evaluation. Figure 10 illustrates that more rules cost more time in event processing; the event processing time is linearly increased when rules increased. When the number of atom events increased from 10,000 to 100,000, the time cost is increased from 0.65 to 3.02 seconds under 5 rules. However, if we increased the number of rules to 10, the time cost also is increased from 1.17 to 5.00 seconds. Therefore, the number of rules has a significant impact on the performance of event processing.

5.2.3. Sliding Window. The capability of the two engines in processing sliding window is compared here. The third case of LIDA-E is based on a simple testing rule computing the value of the selected events with two strict constraints. Rule 4-1 contains the time window and Rule 4-2 includes the batch window.

To evaluate the capability of event processing, LIDA-E and Esper were, respectively, deployed in Rule 4-1 and Rule 4-2. 20 rules with the same constraints were deployed in both engines. In order to simulate input events, the event filter sent atom events with uniformly distributed attribute between 1 and 100, while all the input events had the same event type and source. In this test, we have used three different workloads tasks to evaluate the engines, transmitting, respectively, 20%, 50%, and 80% available events of input events (the remaining events were not selected by the engine):

Rule 4-1: UserRepeatDocExamine

```
SELECT EVERY(e.Value) WHERE e.TYPE = 'CPU_Usage' ∧ e.Value > Selectivity
WHITIN(0.1 sec)
THEN ACTION create CPU_Usage_High Warning
```

Rule 4-2: UserRepeatDocExamine

```
SELECT EVERY(e.Value) WHERE e.TYPE = 'CPU_Usage' ∧ e.Value > Selectivity BATCH(10)
THEN ACTION create CPU_Usage_High Warning
```
First of all, Rule 4-1 is deployed in the two engines to test the capability of executing time window. Figure 11 shows how processing time varies in relation to the selected rate of available events. We observed that both engines performed very well, even if the event provider sent a huge number of input events to them in a short time. The tests also show processing time of both engines when the selected rate of input events is increased from 20% to 80%. Even more important from Figure 11 is that LIDA-E outperforms Esper in the three workloads. Particularly, the processing time of LIDA-E is about half of Esper’s to deal with the same number of input events, which means LIDA-E processes events faster than Esper with the same deployed rules and workloads.

Second, Rule 4-2 is used to evaluate the performance of the two engines in batch window. Figure 12 shows the processing time of the two engines, adopting batch operator to deal with input events (batch size is 10). It can be observed that the time cost of LIDA-E was less than of Esper in different selected rate. Therefore, the performance of LIDA-E is more stabilized than Esper’s with different selected rate, especially when more input events are selected in CEP engine.

In general, our experimental results confirmed the advance in real-time event processing. The implementation of LIDAS with LIDA-E has power of dealing with large volume of events during different experiments. Therefore, it efficiently and reliably provides a real-time delivery and analysis service to high-frequency business data streams.

6. Conclusion

In this paper, a novel and effective architecture with the combination of a lightweight, intelligent data analytics system (LIDAS) and CEP engine (LIDA-E) is presented. In LIDAS, event models (atomic and complex) and operators
(logical, mathematical, and temporal) are formulated for event detection and analysis. LIDAS is based on a modular architecture, which clearly separates the core logic and is devoted to event transmission from agent to filter, and event processing handled by LIDA-E engine. Its knowledge base with rules is responsible for intelligent data analysis with the proposed event processing algorithm. An event transmission component, explicitly designed to target the needs of applications that have to cope with event generation and their transmission, utilizes an efficient engine LIDA-E capable of analyzing large volume of input data in an efficient way.

A comparison of LIDA-E and Esper, which is the most widely used commercial solution for CEP and is known for its expressiveness and efficiency, shows that the performance of LIDA-E is better than of Esper in different tests. In filtering events, the throughput of LIDA-E is almost twice that of Esper when input rate is above 20k events/s. Comparing with Esper, LIDA-E decreases the processing time in different selected rates on dealing with sliding window. The better performance of LIDA-E can be clearly summarized through the results of comparative tests.

**Competing Interests**

The authors declare that they have no competing interests.

**Acknowledgments**

This work was supported by the Research on Key Technology of Virtual Restoration Mosaic in Terracotta Army.
References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-oriented computing: state of the art and research challenges," Computer, vol. 40, no. 11, pp. 38–45, 2007.

[2] S. Tsuchiya, Y. Sakamoto, Y. Tsuchimoto, and V. Lee, "Big data processing in cloud environments," Fujitsu Scientific and Technical Journal, vol. 48, no. 2, pp. 159–168, 2012.

[3] Dain Hansen, Oracle Fast Data: Real-Time Strategies for Big Data and Business Analytics [I], Big Data Gets Real-time, Oracle Fast Data, Real-Time Strategies for Big Data and Business Analytics [I], 2013.

[4] Data analysis, https://en.wikipedia.org/wiki/Data analysis.

[5] C. Esposito, M. Ficco, F. Palmieri, and A. Castiglione, "A knowledge-based platform for Big Data analytics based on publish/subscribe services and stream processing," Knowledge-Based Systems, vol. 79, pp. 3–17, 2015.

[6] http://www.cs.colorado.edu/~kena/classes/5448/sl1/presentations/hadoop.pdf.

[7] Storm, http://storm-project.net.

[8] J. Xu, Z. Chen, J. Tang, and S. Su, “T-storm: traffic-aware online scheduling in storm,” in Proceedings of the IEEE 34th International Conference on Distributed Computing Systems (ICDCS’14), pp. 535–544, Madrid, Spain, July 2014.

[9] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in Proceedings of the USENIX Annual Technical Conference (USENIX ’09), p. 28, USENIX Association, San Diego, Calif, USA, June 2009.

[10] R. Bruns, J. Dunkel, H. Masbruch, and S. Stipkovic, "Intelligent M2M: complex event processing for machine-to-machine communication," Expert Systems with Applications, vol. 42, no. 3, pp. 1235–1246, 2015.

[11] B. Balis, B. Kowalewski, and M. Bubak, "Real-time Grid monitoring based on complex event processing," Future Generation Computer Systems, vol. 27, no. 8, pp. 1103–1112, 2011.

[12] I. Zappia, F. Paganelli, and D. Parlanti, "A lightweight and extensible Complex Event Processing system for sensor and respond applications," Expert Systems with Applications, vol. 39, no. 12, pp. 10408–10419, 2012.

[13] D. C. Luckham and B. Frasca, "Complex Event Processing in Distributed Systems," Tech. Rep., 1998.

[14] Y.-Z. Liu and F.-W. Han, "RFID complex event processing for RTLS," in Proceedings of the 4th International Conference on Multimedia and Security (MINES ‘12), pp. 392–395, Nanjing, China, November 2012.

[15] Y. H. Wang, K. Cao, and X. M. Zhang, "Complex event processing over distributed probabilistic event streams," Computers and Mathematics with Applications, vol. 66, no. 10, pp. 1808–1821, 2013.

[16] R. Baldoni, L. Montanari, and M. Rizzuto, "On-line failure prediction in safety-critical systems," Future Generation Computer Systems, vol. 45, pp. 123–132, 2015.

[17] D. Wang, E. A. Rundensteiner, H. Wang, and R. T. Ellison III, "Active complex event processing: applications in real-time health care," Proceedings of the VLDB Endowment, vol. 3, no. 1-2, pp. 1545–1548, 2010.

[18] W. Yao, C.-H. Chu, and Z. Li, "Leveraging complex event processing for smart hospitals using RFID," Journal of Network and Computer Applications, vol. 34, no. 3, pp. 799–810, 2011.

[19] J. Boubeta-Puig, G. Ortiz, and I. Medina-Bulo, "A model-driven approach for facilitating user-friendly design of complex event patterns," Expert Systems with Applications, vol. 41, no. 2, pp. 445–456, 2014.

[20] C. Zang and Y. Fan, "Complex event processing in enterprise information systems based on RFID," Enterprise Information Systems, vol. 1, no. 1, pp. 3–23, 2007.

[21] M. Ahmadzadeh Ghasemabadi and P. D. Shamsabadi, "Application of five processes of project management based on PMBOK-2008 standard to run EPM-2010 project management system: a case study of Aryan Hamrah Samaneh Co," in Proceedings of the 2nd IEEE International Conference on Emergency Management and Management Sciences, pp. 792–795, Beijing, China, August 2011.

[22] S. Berziža and J. Grabis, "Knowledge reuse in configuration of project management information systems: a change management case study," in Proceedings of the 15th International Conference on Intelligent Engineering Systems (INES ‘11), pp. 51–56, IEEE, Poprad, Slovakia, June 2011.

[23] M. Taghavi, A. Patel, and H. Taghavi, "Web base project management system for development of ICT project outsourced by Iranian government," in Proceedings of the 2nd IEEE International Conference on Open Systems (ICOS ‘11), pp. 273–278, Langkawi, Malaysia, September 2011.

[24] D. Luckham and R. Schulte, "Event processing glossary—version 1.1," Tech. Rep., Event Processing Technical Society, 2008.

[25] Q. Li, Y. Jin, T. He, and H. Xu, "Smart home services based on event matching," in Proceedings of the 10th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD ‘13), pp. 762–766, Shenyang, China, July 2013.

[26] S. Jayasekara, S. Kannangara, T. Dahanayakage, I. Ranawaka, S. Perera, and V. Nanayakkara, "Wihidum: distributed complex event processing," Journal of Parallel and Distributed Computing, vol. 79-80, pp. 42–51, 2015.

[27] A. Hinzé and A. Voißard, "EVA: an event algebra supporting complex event specification," Information Systems, vol. 48, pp. 1–25, 2015.

[28] Esper Reference, http://www.espertech.com/esper/release-5.2.0/esper-reference/pdf/esper_reference.pdf.

[29] W. Ahmad, A. Lobov, and J. L. M. Lastra, "Formal modelling of complex event processing: a generic algorithm and its application to a manufacturing line," in Proceedings of the IEEE 10th International Conference on Industrial Informatics (INDIN ‘12), pp. 380–385, Beijing, China, July 2012.

[30] F. Meyer, R. Kroeger, R. Heidger, and M. Milekovic, "An approach for knowledge-based IT management of air traffic control systems," in Proceedings of the 9th International Conference on Network and Service Management (CNSM ’13), pp. 345–349, IEEE, Zurich, Switzerland, October 2013.

[31] I. Ari, E. Olmezogullari, and O. F. Celebi, "Data stream analytics and mining in the cloud," in Proceedings of the 4th IEEE International Conference on Cloud Computing Technology and Science (CloudCom ’12), pp. 857–862, Taipei, Taiwan, December 2012.
[32] F. Terroso-Saenz, M. Valdes-Vela, E. den Breejen, P. Hanckmann, R. Dekker, and A. Skarmeta-Gomez, “CEP-traj: an event-based solution to process trajectory data,” Information Systems, vol. 52, pp. 34–54, 2015.

[33] B. Idiri and A. Napoli, “The automatic identification system of maritime accident risk using rule-based reasoning,” in Proceedings of the 7th International Conference on System of Systems Engineering (SoSE ’12), pp. 125–130, IEEE, Genoa, Italy, July 2012.

[34] G. Cugola and A. Margara, “Low latency complex event processing on parallel hardware,” Journal of Parallel and Distributed Computing, vol. 72, no. 2, pp. 205–218, 2012.

[35] Node.js, https://nodejs.org/.

[36] M. C. Calzarossa and L. Massari, “Analysis of header usage patterns of HTTP request messages,” in Proceedings of the IEEE International Conference on High Performance Computing and Communications, IEEE 6th International Symposium on Cyberspace Safety and Security, IEEE 11th International Conference on Embedded Software and Systems (HPCC ’14, CSS ’14, ICESS ’14), pp. 847–853, Paris, France, August 2014.

[37] S. Zhang and S. Zhu, “Server structure based on netty framework for internet-based laboratory,” in Proceedings of the 10th IEEE International Conference on Control and Automation (ICCA ’13), pp. 538–541, IEEE, Hangzhou, China, June 2013.

[38] Netty Project, http://netty.io/index.html.

[39] EsperTech, http://www.espertech.com/esper/index.php.

[40] TIOBE Index for October 2015, http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
