IJICTR
International Journal of Information & Communication Technology Research

Volume 10- Number 2 – Spring 2018 (33 -44)

# EDBPM: An Event-Driven Business Process Monitoring Mechanism

Mohammad Ali Fardbastani
School of Computer Engineering
Iran University of Science and Technology (IUST)
Tehran, Iran
fardbastani@iust.ac.ir

Farshad Allahdadi
School of Computer Engineering
Iran University of Science and Technology (IUST)
Tehran, Iran
allahdadi@comp.iust.ac.ir

Mohsen Sharifi*
School of Computer Engineering,
Iran University of Science and Technology (IUST),
Tehran, Iran
msharifi@iust.ac.ir

Received: 2 July, 2017 - Accepted: 8 January, 2018

*Abstract*—Many process-aware organizations need to monitor the execution of their Business Processes (BP). Changes in BP execution can be reported as events while real-time detection of event patterns from such events can help the monitoring of model-execution conformance or business activities. Complex Event Processing (CEP) techniques can detect event patterns that are specified as CEP rules. Given the high rate of events and numerous number of complex rules, existing CEP-based solutions are not scalable. We present a novel scalable Event Driven Process Monitoring Mechanism (EDBPM) using distributed CEP. Events are partitioned by process instance identifier and the events of each partition is dispatched to a compute node. As such, the processing load of BP monitoring is distributed adaptively to compute nodes in a load balanced manner. Using a prototyped implementation of EDBPM we show that EDBPM scales well horizontally, i.e. increases in throughput are nearly linear when the number of compute nodes increases. Compared to CPU and memory balancing in a general purpose distributed CEP-based solution, EDBPM keeps the CPU load doubly balanced and does balance the memory too, which is lacking in similar solutions.

***Keywords-Business Process Management, Business Process Monitoring, Complex Event Processing, Scalability, Load Balancing***

## I. INTRODUCTION

All deficiencies of a complex system cannot be resolved and not all violations of its behavioral constraints can be predicted during design and development of the system. On the other hand, the full behavior of a system is observable and all of its deficiencies and violations are detectable only at runtime. Keeping track of a system's behavior and interaction of the system with its environment and also

* Corresponding Author

interactions between its components during operation is called runtime monitoring [1]. One of the most important issues in most organizations is the monitoring of their business processes. Therefore, most of the proposed monitoring frameworks are developed for Business Process (BP) monitoring [1]. The main types of monitoring in a Business Process Management System (BPMS) are Business Activity Monitoring (BAM) [2], Service Level Agreements (SLA) violations monitoring [3] and execution-model conformance monitoring [4, 5].

The state changes of objects within the context of a BPMS are represented by events [6]. In a computing system, an event is produced, processed and stored as a data object [7]. Events may contain some attributes such as happening time, corresponding process instance, activity and user identifiers [8, 9]. Therefore, in a BPMS, an event set may be larger or equal in cardinality to its corresponding generated log set.

Using event aggregation, correlation and pattern detection, notable changes in Key Performance Indicators (KPI), execution status according to SLAs and execution deviations from models could be investigated. To the best of our knowledge, the best technology for such a monitoring in a real-time or almost real-time manner is Complex Event Processing (CEP) and the researchers of the BPM domain have not introduced any better solution [1, 8, 10, 11]. Therefore, CEP has been used widely in recent related works such as the works reported in [2, 12–17]. The result of CEP may trigger a reactor or even pro-actor system and can be visualized as notifications or graphs to show a detected issue, its root causes and its degree of importance [8, 18]. Furthermore, the result of CEP could be a feedback for the activities of running processes [19] and even a rule tuning system that modifies monitoring rules according to the current status of the system [11].

In general, CEP solutions abstract the behavior of a system by extracting high-level information from lower-level system events in a real-time or almost real-time manner [7]. This event abstraction is done through event aggregation, correlation, and pattern detection. Therefore, the best way for (almost) real-time event pattern detection through BP execution monitoring and extracting higher-level information (e.g. fraud detection from financial transaction logs) is the CEP deployment.

Nowadays, organizations have complex processes and high rates of process instance generation through concurrent process executions. So CEP systems are faced with high rates of input events and numerous complex rules, and performance, scalability, and low latency have become big challenges in BP monitoring [20]. In other words, for sound (Definition 9, defined in Section 5) and real-time operation of a CEP system, efficient load distribution and sufficient resource allocation are the biggest challenges.

The first step towards enhancement of BP monitoring using CEP is to optimize the generation of monitoring rules (queries on the generated events). Most existing research works such as those reported in [3–5, 21] have focused on optimized CEP rule generation. However, there is no work on the scalability of the processing of BP monitoring rules. Therefore, a better solution for scalable BP monitoring using CEP is needed in addition to general solutions proposed for Distributed CEP (DCEP) such as [2, 22]. In other words, a proper integration of a CEP system in a specific domain such as BPM leads to better performance.

In this paper, we present a novel Scalable Business Process Monitoring Mechanism (EDBPM) using CEP for isolated process instances that scales out the BP monitoring via distribution of the processing load on a set of compute nodes. The main attributes of EDBPM that make it novel include:

- Balancing the load of multiple resource types among multiple compute nodes
- Elasticity (the ability of runtime addition / removal of compute nodes)
- Adaptability with process instance generation rates (input event rates)
- CEP engine independence
- Heterogeneity support (compute nodes and/or CEP engines running on nodes)

We have organized the rest of the paper as follows. Section 2 presents related works. Section 3 highlights our motivation by outlining an important application of BP monitoring. Section 4 explains the performance challenge that is addressed by EDBPM. Section 5 present a formal specification of the problem to be resolved by EDBPM. Section 6 presents the EDBPM in detail. Section 7 reports the results of our experiments with a prototype implementation of EDBPM and Section 8 concludes the paper.

## II. Related Work

Some previous works such as [23] had focused on offline conformance checking or pre-mortem and non-real-time processing of events that persist in a database such as HBase [19]. These works are not suited to monitoring of systems whose generated events should be processed in real-time manner.

The first step in the enhancement of performance of BP monitoring systems using CEP is optimization of rule generation. For example in [3, 4] resource utilization is improved by reduction of complexity of rules. In such solutions, the workload of the system is assumed constant and known while the process instance generation and activities have dynamic nature and one cannot predict the workload of a real organization. Some other works, have tried to reduce monitoring rules for monitoring of a process. For example Backmann et al. [5] have used RPST [24] for workflow fragmentation and generated fewer number of rules from the workflow tree, and another work [25] has tried to improve this rule generation for the monitoring of choreography. Optimization of resource utilization is not enough when a BPMS generates high rates of input events and there are a high number of monitoring rules for many complex business processes.

On the other hand, general-purpose DCEP mechanisms could be used to deal with the challenges. Some generic distributed CEP mechanisms have been

proposed that by dispatching input events [26, 27] or partitioning CEP rules [28, 29], distribute the processing load among different compute nodes. These mechanisms include many simplifying assumptions and preprocessing to balance the load and handle dependencies of CEP nodes, resulting in high processing and communication overheads.

To the best of our knowledge, no approach is proposed for integrating a CEP engine cluster and a BPMS to monitor its processes. EDBPM provides a scalable CEP mechanism for monitoring of business processes. It partitions the generated events by a BMPS and distributes processing load of the events among a cluster of CEP nodes.

## III. Sample Application: Conformance Monitoring

There are two types of BP models, normative and descriptive. Normative models show the constraints that a BP should comply with it at runtime. They are used to influence reality. Descriptive models are used to capture or predict reality by using process mining [30]. On the other hand, all changes in a BPMS are logged as a set of sequences of events called traces [9]. These logs could be processed in two ways. The first is post-mortem or offline that only considers logs of terminated process instances. The second is pre-mortem or online processing that refers to the log processing of running or alive process instances [30].

The online processing of logs to detect deviations of processes from their normative models is called compliance monitoring. Compliance monitoring is the processing of the logs of a BPMS, in order to detect any violations in the constraints (defined as rules) of the processes. These constraints may correspond to various aspects, e.g. behavior profiles [4] or even non-functional requirements such as security [31].

Offline cross-checking between descriptive and normative model and logs and quantification of discrepancies between the log and the model is called conformance checking [32]. In conformance checking, discrepancies between the normative model and the logs refer to the deviation in the process execution while discrepancies between the descriptive model and the logs refer to model insufficiency and necessity of promotion of the model. If a model is not fit, it should be extended and if an extra behavior (that never occurs in reality) exists in the model, it should be pruned [9].

We call real-time crosschecking between the descriptive model and the normal model and logs and quantification of discrepancies as *conformance monitoring*. It is more general than compliance monitoring. In conformance monitoring, execution control of processes and the descriptive model can be promoted using real-time detection of deviations and model deficiencies. In addition, real-time monitoring in this context is more specific than online monitoring. In real-time monitoring, generated events are processed upon their generation and before they are stored persistently. Thus same as CEP [33], we do not make a query on persisted events; queries are applied to input streams.

## IV. Problem Statement

Increased complexity and usage of BPMSs has led to increases in the rate of process instance generation and the number of concurrent executing processes and as a result the rise in the rate of input events. Consequently, the monitoring of big and complex BPs needs a high number and more complex CEP rules. Thus, performance is an important challenge in the context of BP monitoring using CEP. Most existing related works have focused on optimizing CEP rule generation from process models [3, 5, 21]. Furthermore, the performance and scalability enhancements to CEP have been studied in general quite independent of applications (e.g. BP monitoring). In contrast, in this paper we propose a mechanism that enhances the performance and scalability of CEP specifically for BP monitoring application by integrating CEP and BP monitoring more efficiently.

There are two types of Distributed CEP (DCEP). The first is EPN [34] that consists of several event processing agents and each agent processes a particular part of a rule. Therefore, in an EPN, an agent cannot process a rule alone because CEP rules are matched collaboratively by many agents. The second type of DCEP include systems that are built from a cluster of CEP engines, wherein each engine can process any rule independent from the others. In this type of DCEP systems, the processing load should be distributed on compute nodes in such a way that each node is only concerned with a portion of input events [27, 35] and/or with matching a given subset of CEP rules [36–38]. However, since BP monitoring is stateful in the sense that the history of previous activities of a process instance must be available upon processing of a newly arrived activity of the instance (event), and there are correlations among processing of rules for a process instance, general purpose rule partitioning mechanisms have high communication overheads for sharing the state of rules among compute nodes. Furthermore, rule improvement techniques such as refined process structure tree (RPST) [39] are not suited to these general purpose CEP mechanisms.

Also according to the stateful nature of CEP and BP monitoring, load distribution through event dispatching may lead to false detection of patterns because of event dependencies. For example, if we have "`A occurs after B`" rule and the event dispatcher sends events of type `A` to a node and events of type `B` to another node, the pattern is never detected.

A couple of solutions can be suggested to eliminate errors of event pattern detection when using event dispatching in a DCEP. The first solution would be to multicast shared events in multiple rules. For example, in Figure 1, one can think of dispatching events of type `A` to the left node, dispatching events of type `C` to the right node, and dispatching events of type `B` to both nodes. In this solution, event multicasting may waste network resources. The second solution is to create a shared memory for partial pattern matches. This can cause lots of processing overhead while accessing the shared memory. A better solution would be to partition events into independent subsets and then dispatch each part to a separate CEP node. For example, in Figure 1, events could be partitioned into two parts: (1) events of

type A and events of type B whose x attributes are greater than 50, and (2) events of type C and events of type B whose x attributes are less than 50.

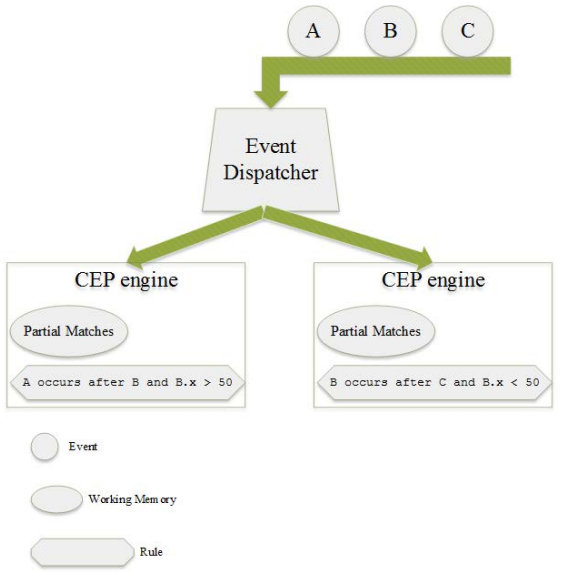


Figure1. Event partitioning mechanism

Unfortunately, most existing load balancing mechanisms do not adapt to load changes that arise due to changes in input event rate (process instances' rate) [36–38]. However due to the dynamic nature of organizations, it is often required that load balancing be *adaptable* to transient changes in the rate of input events. On the other hand, the workload of BP monitoring may increase or decrease for a period, requiring changes to the number of active CEP nodes. It is thus necessary to provide support for elastic number of compute nodes at runtime without stopping BP monitoring.

Another challenge of distributed BP monitoring is *heterogeneity*. A computer cluster may well consist of heterogeneous compute nodes running even different CEP engines. Therefore, if a proposed mechanism for scalable CEP supports heterogeneity, it should not depend on the types of computers and the types of CEP engines.

In this paper, we propose EDBPM for horizontal scaling of BP monitoring using CEP. It scales out the system using adaptable load balancing via runtime event partitioning that can work elastically on heterogeneous environments.

## V. Formal Problem Definition

Before presentation of EDBPM, we formally define the problem that is addressed by it. Therefore, we define the process model, the model execution, process instance, event and related concepts of an event-driven BPMS hereafter.

*Definition 1* (Process Instance). A tuple $P = (pid, m, AI)$ is a process instance, wherein $pid$ is a globally unique identifier for the process instance, $m$ is the process model that $P$ is instantiated from it and $AI$ is the set of activity instances of the process instance.

*Definition 2* (Event Type). An event type $\tau = (typeid, ATR)$ has a unique identifier, and an attribute type set. The attribute type set $ATR$ specifies the event type has what attributes.

*Definition 3* (Event Type Set). The event type set $T$ of a system is the set of all valid event types of the system.

*Definition 4* (Event). Event is an object $\varepsilon = (\tau, eventid, pid, timestamp, attrs)$. The event is an instance of type $\tau \in T$. It has a unique event identifier $eventid$, a $pid$ that identifies the event generated during the execution of a process instance, a $timestamp$ that indicates the occurrence time of the event, and a set of attributes $attrs$ whose numbers and types conform with the specification of $\tau$.

Given the stated definitions, we formally define the problem space of EDBPM in the context of a specified number of compute nodes for BP monitoring that uses a specified set of resource types for running CEP engines.

*Definition 5* (System Current Utilization function). If we have a node set $M = \{m_1, m_2, \dots, m_n\}$ and a CEP engine runs on each node, and there are $k$ resource types on each node, the System Current Utilization (SCU) function $\Gamma(t)$ denotes the utilization of each resource at each node at time $t$

$$\Gamma(t) = \begin{bmatrix} \gamma_{11}(t) & \cdots & \gamma_{1k}(t) \\ \vdots & \ddots & \vdots \\ \gamma_{n1}(t) & \cdots & \gamma_{nk}(t) \end{bmatrix}$$

wherein $\gamma_{ij}(t)$ shows the utilization of resource $j$ at node $i$ and $0 \leq \gamma_{ij} \leq 1$.

The output of SCU function is a matrix that shows the utilization of each resource type in each compute node at any moment in time. The average of each column of $\Gamma(t)$ yields the average utilization of each resource type in the system. System Utilization (SU) function returns a vector that contains this information.

*Definition 6* (System Utilization function). The SU function $\Psi(t)$ returns a vector $[\psi_1(t), \psi_2(t), \dots, \psi_k(t)]$ that denotes the average resource utilization over all nodes and $\psi_j$ denotes the average utilization of resource $j$ at time t.

$$\psi_j(t) = {}^1\!/_n \sum_{i=1}^{n} \gamma_{ij}(t)$$

We define the load imbalance of each resource type in the system using the standard deviation.

*Definition 7* (System Imbalance function). The System Imbalance (SI) function $I(t) = [\sigma_1(t), \sigma_2(t), \dots, \sigma_k(t)]$ denotes the imbalance factor (IF) of each resource using standard deviation of utilization of the resource in the system and $\sigma_j(t)$ shows the standard deviation of resource $j$ at time t.

$$\sigma_i(t) = \sqrt{{}^1\!/_n \sum_{j=1}^{n} (\gamma_{ji}(t) - \psi_i(t))^2}$$

*Definition 8* (Total System Imbalance). Because for all resource types of all nodes $0 \leq \gamma_{ij} \leq 1$, we can define Total System Imbalance (TSI) as the average standard deviation of all resource types at time t.

$$TSI(t) = {}^1\!/_k \sum_{i=1}^{k} \sigma_i(t)$$

*Definition 9* (Soundness of CEP). A CEP mechanism is sound if and only if all occurred complex events are detected by the mechanism and all detected complex events by the mechanism have actually been occurred. In other words, there is no false negative and no false positive detection when using the mechanism.

*Definition 10* (Dynamic Event Partitioning problem). Suppose CEP engines on all nodes can detect all event patterns that are defined for conformance checking. The Dynamic Event Partitioning (DEP) problem can be resolved by a mechanism that could dynamically partition the input events in a way that if events of each part are dispatched to a separate node, all defined complex events are detected soundly.

*Definition 11* (Multiple Resource Load Balanced DEP problem). If the system works as long as $L$, the DEP problem should be solved in a way to minimize ${}^1\!/_L \int_{t=0}^{L} TSI(t)$. This is an optimization problem to minimize the average of the total system imbalance during the system lifetime.

The stated problem is a complex non-linear programming like problem. In addition, the optimization condition depends on dynamic and non-deterministic conditions at execution time. Therefore, we should propose a heuristic solution to the problem using Definition 11.

## VI. EDBPM

EDBPM distributes the processing load of BP monitoring via event partitioning based on process instance (as event source) and dispatching of each partition to a CEP node. We assume that the monitoring of each process instance is independent of others. Because most of monitoring tasks in a BPMS are the monitoring of workflows, service level agreements and some other organizational constraints throughout the execution of each process instance independent from execution of the other process instances, similar to the mechanism in [4], EDBPM does not consider any relationship between process instances. Therefore, we cover most of the requirements of BP monitoring. According to the assumption, when all generated events of a process instance are dispatched to a CEP node, there is no need for event multicasting or CEP state sharing among CEP nodes. In other words, EDBPM assigns the monitoring of each process instance to a specified CEP node. Also, we assume all input events are aligned (alignment issues such as event miss and redundancy have been discussed in previous works such as [40]) and homogeneous (event heterogeneity has been discussed in [15]).

In EDBPM, events are produced, processed and consumed in 4 layers (Figure 2). Event producers are in the first layer. BPMSs that are monitored by EDBPM are the event producers of the system. Any changes in their activities are reported as events and these events are the input of the coordinators in the second layer. The most costly task of coordinators is in the dispatching of input events. Therefore, we can say they filter the input events for each CEP node. However, the CEP nodes, in addition to filtering of the input events according to the rules, detect complex logical and temporal patterns. Therefore, we can say, the load of coordinators is very low in comparison with CEP nodes. Thus, we focus on load balancing of CEP nodes; each coordinator subscribes to a subset of event producers and the cardinality of these subsets is almost equal.

Coordinators dispatch input events and monitor resource utilizations of CEP nodes. Upon generation of a new process instance, one of the coordinators that receives the first event of the instance selects a CEP node with the least resource utilization among all CEP nodes to process producing events of the new process instance and add a new entry in the Dispatching Table (Definition 12) for this decision. In other words, each CEP in EDBPM is responsible for monitoring of a subset of running process instances and all produced events of each process instance are only dispatched to a specific CEP node. So collectively, coordinators balance the loads on CEP nodes using resource utilization monitoring, event partitioning, and event dispatching, i.e. provide *scalability* via load balancing. Furthermore, EDBPM updates its dispatching decisions upon new process instance generation and balances the system load *adaptable* to changes in the system workload (i. e., the rates of process instance generation).

After coordinators dispatch input events to the next layer, CEP nodes in the third layer process the basic input events to derive complex events. Each CEP node runs an independent CEP engine. The running engines can be homogenous or heterogeneous but they should have equal monitoring rule sets. Upon derivation of a complex event by CEP nodes, they send the event to the consumers in the last layer. Consumers may show the results in a dashboard, react upon detection of new situation or even re-process the output of EDBPM for deriving further information such as inter-process correlations.

EDBPM supports *elasticity*, implying that the number of coordinators and CEP nodes can be changed at runtime. Because the state of coordinators is stored in a shared table (Dispatching Table), addition and removal of coordinators only requires synchronization with the table and informing the event producers.

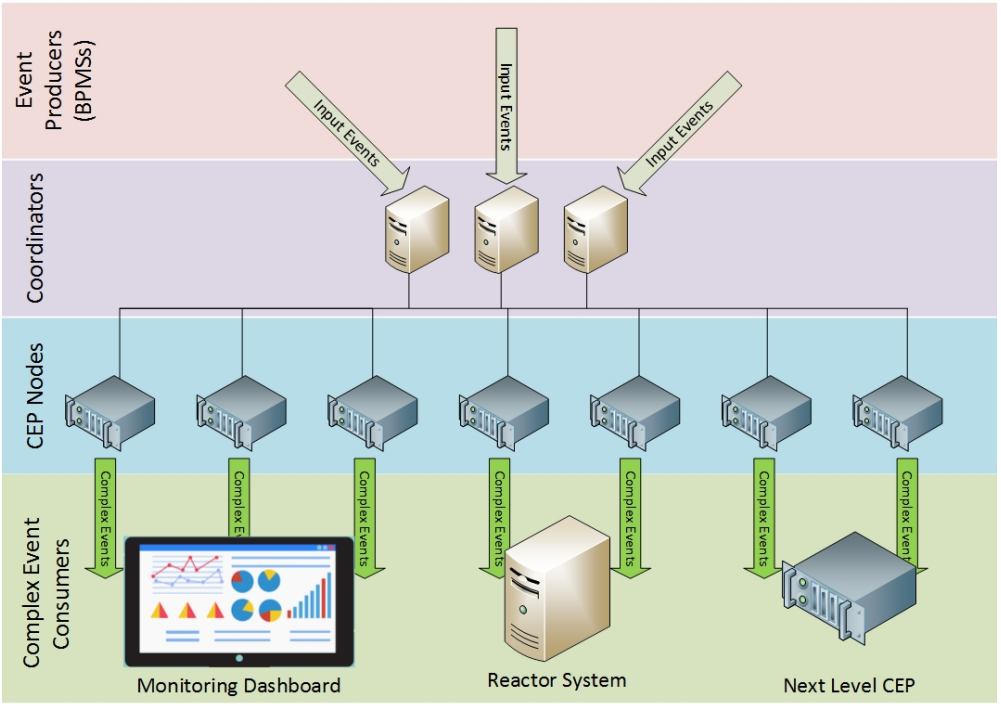


Figure 2. EDBPM Architecture

When a new CEP node is added, it sends its resource utilization status to coordinators and coordinators start to assign a new task to the new CEP node according to the load-balancing policy. For removing a CEP node, coordinators stop assigning new monitoring tasks to the node to be removed and after the termination of all process instances previously assigned to the node, the node is removed.

Because all CEP nodes operate independently, to provide support for *heterogeneity* in EDBPM, it is sufficient to implement equivalent rule sets for each CEP engine.

### A. *EDBPM Load-Balancing Heuristic Algorithm*

Coordinators of EDBPM assign the monitoring of each process instance to a CEP node. Therefore, relevant events of a process instance are dispatched to that CEP node. Such assignments and dispatching policies are registered in Dispatching Table of coordinators (Definition 12).

*Definition 12* (Dispatching Table). The Dispatching Table has two columns. Items of the first column are unique and refer to the identifiers of currently running process instances. The second column refers to CEP nodes of the system. Each row determines the producing events of each running process instance that should be dispatched to a CEP node.

Any receiving event in coordinators is handled and dispatched by calling the `dispatch` function (Figure 3). The function lookups the Dispatching Table for the process instance id (*pid*) of the event. If the corresponding *pid* is in the table, coordinator dispatches the event to the CEP node that is registered in the looked up row. If not, coordinator adds a new row to the Dispatching Table for the *pid* to inform other coordinators that finding a proper CEP node for monitoring of the process instance is in progress. To find a proper node, a resource type with the greatest value in the current SI vector (the most utilized resource type among all CEP nodes) is selected. If the $k^{th}$ resource type is selected, coordinator looks for the node with the lowest value in the $k^{th}$ column of the current SCU matrix (the node whose $k^{th}$ resource type is the least utilized among all nodes). The added row in the Dispatching Table is updated and the selected CEP node identifier is added to the row.

### B. *EDBPM Validation*

The proposed dynamic mechanism partitions events dynamically. Therefore, according to Lemma 1, it is a valid solution for DEP problem. Also, the mechanism, based on Lemma 2, leads to minimum network utilization.

**Lemma 1**: Our proposed mechanism is a valid solution to the DEP problem.

Proof: We prove it in three steps. Firstly, the mechanism is dynamic. Dispatching policy changes according to process instance generation throughout conformance checking which is not fixed at the system startup. Secondly, there is no false positive detection. This can be proved by contradiction. Suppose a CEP node receives an event that is matched with a pattern incorrectly. If this event relates to process instances whose conformance checking is done by the node, the match is correct but if the event relates to other process instances, the dispatcher is not conformant with our mechanism. Similarly, suppose a CEP node does not receive an event that causes a mismatch. This means that an event that is related to an assigned process instance in the node, did not receive and it is incompatible with our mechanism. Thirdly, there is no false negative. This can be proved by contradiction too.

If a received event does not match with a pattern, it means that the event relates to the node and there is no error and if the node does not receive an event and consequently a pattern does not match, the dispatcher works incorrectly.

```
dispatch(event)
{
  if(DispatchingTable.entryExists(event.pid))
  {
    targetNodeId = DispatchingTable.getTargetNodeId(event.pid);
    dispatch(event, targetNodeId);
  }
  else
  {
    DispatchingTable.addEntry(event.pid, SEARCHING_CODE);
    currentSI = SI(current);
    mostUtilizedResource = getIndexOfMaxValue(currentSI);
    currentSCU = SCU(current);
    mostUtilizedResourceDetails = SCU.getColumn(mostUtilizedResource);
    selectedNodeId = getIndexOfMinValue(mostUtilizedResourceDetails);
    DispatchingTable.updateRow(event.pid, selectedNodeId);
    dispatch(event, selectedNodeId);
  }
}
```

Figure 3. Event Dispatching by Coordinators

**Lemma 2**: Our proposed mechanism minimizes network utilization.

Proof: Because of event partitioning, events are not duplicated. In addition, because all rules are on all nodes and all partial matches of a process instance's anti-patterns are also on a single node, there is no need for state sharing over the network.

## VII. EVALUATION

To evaluate performance and scalability of EDBPM, coordinators and the CEP module have been implemented and tested using different scenarios. In these scenarios, performance of EDBPM is tested using different number of compute nodes and variable input rates of process instances. For evaluating the scalability of EDBPM, the maximum throughput (Definition 14) when the number of compute nodes changes were evaluated. The load balancing of EDBPM for a variable rate of input process instances was evaluated and compared with a distributed CEP that is proposed in [36].

Our application domain was conformance monitoring and we developed a simulator that can randomly generate distinct process instances from some process models. Each activity instance has a random duration time and each activity has a maximum duration time. Ninety percent of the generated process instances were conformant to the process model and others deviated from their model. Because the count and the type of deviations were known, result of experiments are validated to determine the error rate of detected deviations. The error ratio evaluates the monitoring system's functionality that determines the system throughput.

Experiments were performed using a set of virtual machines. The specification of the coordinator node was as follows:

- Operating system: Ubuntu server 14.04
- Processor: 64bit 2 cores
- Memory: 2GB
- JVM: 1.8 Oracle

As mentioned before, the load of coordinators is very low in comparison with CEP nodes. Therefore we did not need more than one coordinator in all of the experiments.

The specification of all CEP nodes was as follows:

- Operating system: Ubuntu server 14.04
- Processor: 64bit single core
- Memory: 2GB
- CEP Engine: Drools Fusion 6.2
- JVM: 1.8 Oracle

### A. *EDBPM Maximum Throughput*

In order to measure the maximum throughput of EDBPM, different rates of process instances per second were iteratively injected into the system with a specified number of compute nodes and the maximum throughput of the distributed mechanism with that specified number of compute nodes is evaluated as defined in Definition 14. High load of the system may cause event loss (such as CEP engine load shedding, buffer overflow, network loss) and therefore detection error.

*Definition 13* (Complex Event Detection Error Ratio) CEDER is the sum of false positive detections (wrongly detected) and false negative detections (undetected) over the total number of actually occurred complex events.

$$CEDER = \frac{false\ \text{positive}\ CEs + false\ negative\ CEs}{occurred\ CEs}$$

We averaged the sample IFs of CPU and memory and recorded the average values in Table I. The imbalance factor shows the distribution of utilization values of all nodes around the average value and therefore the balance of the loads. For example, when evaluated EDBPM with 6 nodes, load of CPUs was distributed from average load minus 0.073244 till average load plus 0.073244.

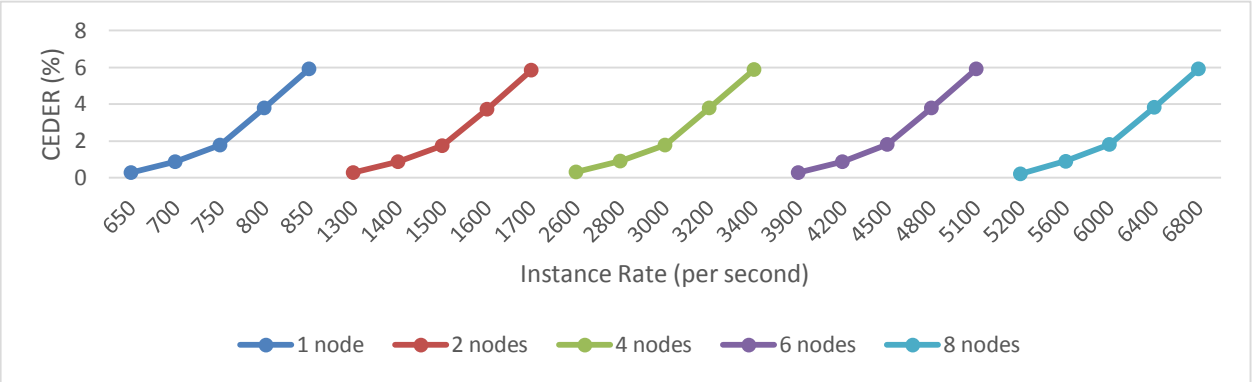


Figure 4. EDBPM Throughput

*Definition 14* (Maximum Throughput) Maximum throughput of a system is considered as the maximum process instance generation rate such that CEDER of the system is less than an acceptable threshold.

We repeated our experiments by increasing the number of compute nodes, starting with one compute node up to 8 compute nodes. Figure 4 shows the results of experiments. The results show that the EDBPM maximum throughput for five different values of CEDER increases in a near-linear fashion by increases in the number of CEP compute nodes. This is because CEP compute nodes work independently on different sets of process instances and with increases in the number of CEP nodes, the CEP system can process events of more process instances concurrently.

### B. *Evaluating the Load Balancing*

In order to evaluate the dynamic load balancing (load balancing in varying input rates) feature of EDBPM, we carried out experiments with varying rates of input using 6 and 8 CEP nodes. Therefore, for each second of the experiments, input rate were selected from 1000 to 4000 process instances per second randomly.

The duration of the experiment was 300 seconds and we sampled CPU and memory utilization with a rate of one sample per second. We calculated the imbalance factor (IF) of CPU (Figure 5-a) and memory (Figure 5-b) at each second of the experiment according to Definition 7.

Because of the high rate of events and their small size, CPU utilization was higher than memory utilization, while jitters of CPU utilization of nodes were higher. Therefore, CPU utilization was more imbalanced on average. We can see in Figure 5 that at the beginning of the experiment we had a warm-up period. After warming up, the IF of memory oscillated more than the IF of CPU. Since the more imbalanced resource type has higher priority in EDBPM load balancing, the mechanism tried to balance the load of CPU and the observed oscillation of CPU utilization was less than that for memory.

Table I shows the load imbalance of CPU has a little reduction when the number of nodes is increased. Since the load balancing of CPU has more priority, the change in the memory imbalance is near zero.

TABLE I. AVERAGE IF OF CPU AND MEMORY

| Number of Nodes | CPU IF | Memory IF |
|---|---|---|
| **6** | 0.073244 | 0.003213 |
| **8** | 0.070753 | 0.003228 |

### C. *Comparing Load Balance*

In our second set of experiments, we compared our load balancing mechanism with the one proposed by Isoyama et al. [1]. In these set of experiments, we used 8 compute nodes and a coordinator with the same specification as in the first set of experiments. We used 8 processes with different instance generation rates and the same number of conformance monitoring rules. The total process instance rate was 40000 per second. Load balancing mechanism that is proposed by Isoyama et al. is based on the number and similarity of the rules. In other words, the mechanism tries to balance the number of rules on each CEP node while the rules with similar event types are processed in the same CEP node. However, the required resources for the processing of each rule may differ from the others because of differences in the complexity of rules and the rates of events. Figure 6-a shows that the CPU load of Isoyama et al. mechanism is more imbalanced than EDBPM. Thus, some nodes are overrun and have higher CEDER (Table II). The high CPU utilization of these nodes increased the processing latency resulting in the accumulation of events in memory and a growing memory IF (Figure 6-b).

TABLE II. COMPARATIVE CEDER OF EDBPM AND ISOYAMA ET AL. MECHANISM

| Load Balancing Mechanism | CEDER |
|---|---|
| **EDBPM** | 0.0121859 |
| **Isoyama et al. mechanism** | 0.0275938 |

generation for BP monitoring but ignored the processing of these rules. On the other hand, some others have tried to enhance performance and scalability of CEP without considering its applications. In contrast, we have considered an integration of CEP and BP monitoring by proposing a scalable mechanism

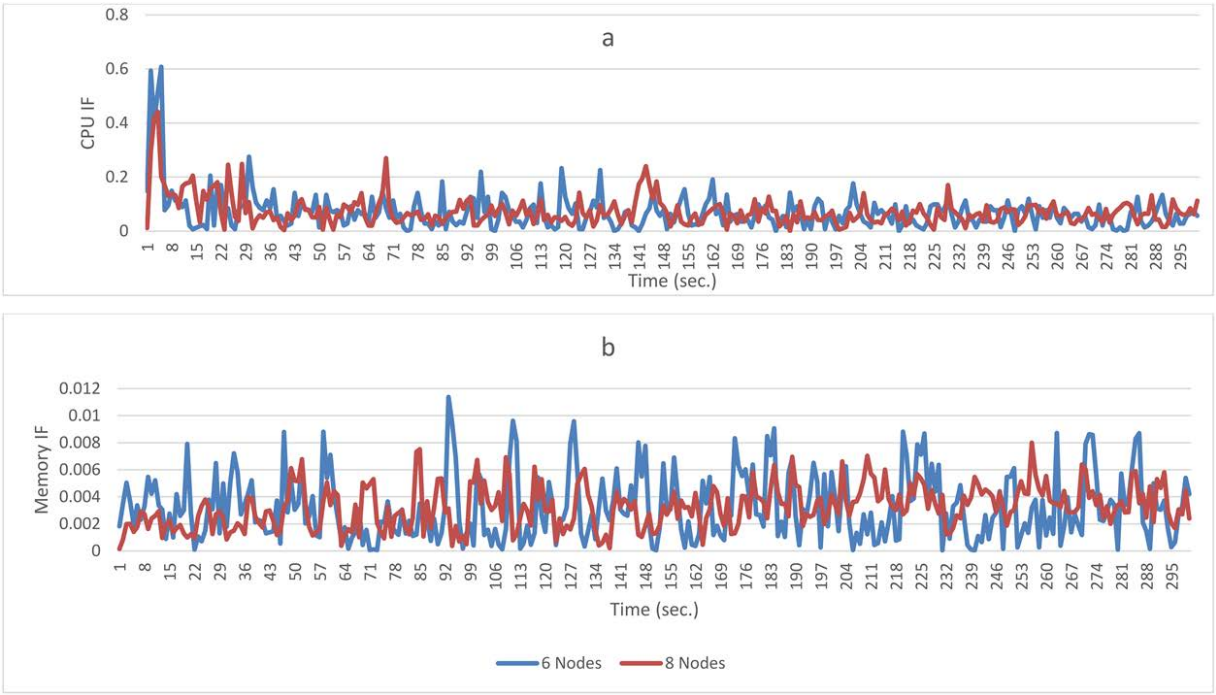


Figure 6. Load Balancing of EDBPM. (a) CPU IF (b) Memory IF

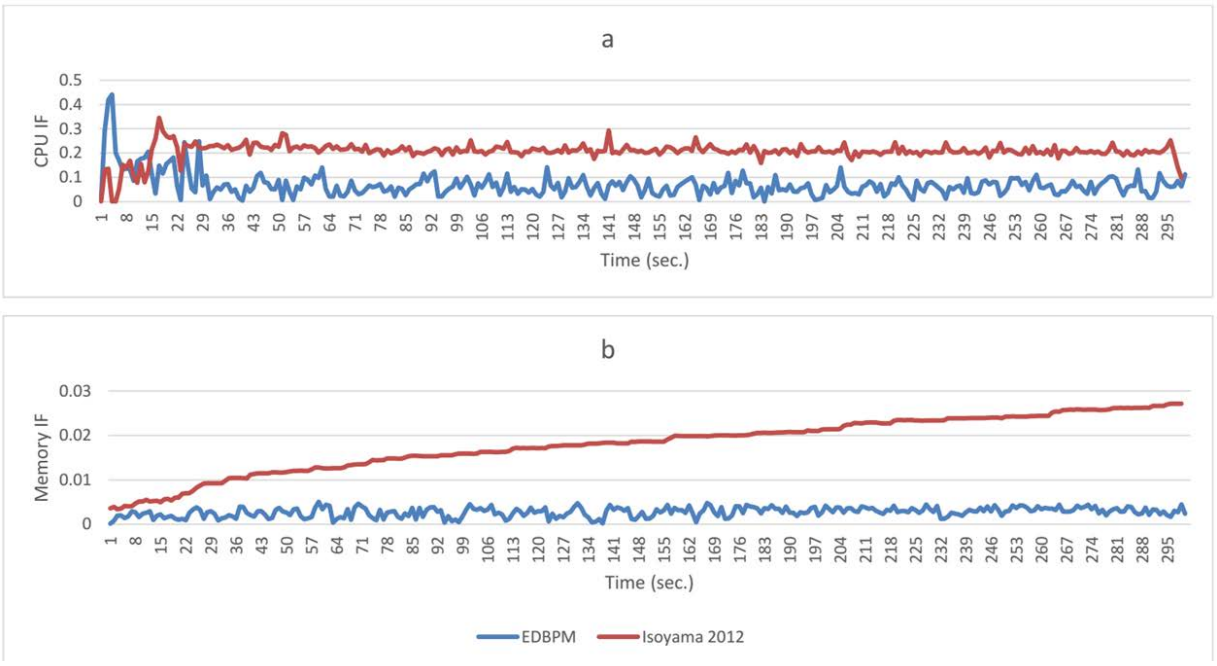


Figure 5. Comparative Load Balancing of EDBPM and Isoyama et al. Mechanism. (a) CPU IF (b) Memory IF

## VIII. CONCLUSION

In this paper, we proposed a Scalable Business Process Monitoring Mechanism (EDBPM) for scalable and distributed monitoring of Business Processes (BPs) using Complex Event Processing (CEP). In previous related works, performance of BP monitoring using CEP was enhanced in two ways. On the one hand, some researchers have focused on optimizing the CEP rule called EDBPM that uses partitioning of input events based on executing process instances on heterogeneous platforms to provide adaptable load balancing. We implemented EDBPM and our experiments showed that with increasing number of CEP nodes, the throughput of BP monitoring increased almost linearly. Also, our experiments showed adaptability and effectiveness of EDBPM load balancing in variable process instance

generation rates. Finally, we implemented a general purpose distributed CEP and our experiments showed better performance of EDBPM in comparison to a general purposed mechanism proposed by Isoyama et al [36].

EDBPM can be improved if it includes monitoring requirements for the correlation of process instances and inter-organizational relations. For this mean, the system can get feedback of monitoring of each process instance to itself to processing correlation of them or the system uses a multi-level CEP that each process instance is monitored in the first layer and monitoring of the correlations is assigned to the next level.

## REFERENCES


[1] R. Rabiser, S. Guinea, M. Vierhauser, L. Baresi, and P. Grünbacher, "A Comparison Framework for Runtime Monitoring Approaches," *Journal of Systems and Software*, vol. 125, pp. 309–321, 2017.

[2] C. Janiesch, M. Matzner, and O. Müller, "Beyond Process Monitoring: A Proof-of-Concept of Event -driven Business Activity Management," *Business Process Management Journal*, vol. 18, no. 4, pp. 625–643, 2012.

[3] M. Weidlich, H. Ziekow, A. Gal, J. Mendling, and M. Weske, "Optimizing Event Pattern Matching Using Business Process Models," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 11, pp. 2759–2773, 2014.

[4] M. Weidlich, H. Ziekow, and J. Mendling, "Optimising Complex Event Queries over Business Processes Using Behavioural Profiles," in *Business Process Management Workshops. BPM 2010. Lecture Notes in Business Information Processing*, Berlin, Heidelberg: Springer, 2011, pp. 743–754.

[5] M. Backmann, A. Baumgrass, N. Herzberg, A. Meyer, and M. Weske, "Model-Driven Event Query Generation for Business Process Monitoring," in *Service-Oriented Computing – ICSOC 2013 Workshops. ICSOC 2013. Lecture Notes in Computer Science*, vol. 8377, Cham: Springer, 2014, pp. 406–418.

[6] M. zur Mühlen and R. Shapiro, "Business Process Analytics," in *Handbook on Business Process Management 2*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 137–157.

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

[8] L. T. Ly, F. M. Maggi, M. Montali, S. Rinderle-Ma, and W. M. P. van der Aalst, "Compliance Monitoring in Business Processes: Functionalities, Application, and Tool-Support," *Information Systems*, vol. 54, pp. 209–234, 2015.

[9] L. Garcia-Banuelos, N. van Beest, M. Dumas, M. La Rosa, and W. Mertens, "Complete and Interpretable Conformance Checking of Business Processes," *IEEE Transactions on Software Engineering*, vol. PP, no. 99, pp. 1–1, 2017.

[10] M. Daum, M. Götz, and J. Domaschka, "Integrating CEP and BPM," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems - DEBS '12*, New York, 2012, pp. 157–166.

[11] R. Mousheimish, Y. Taher, and K. Zeitouni, "The Butterfly: An Intelligent Framework for Violation Prediction within Business Processes," in *Proceedings of the 20th International Database Engineering & Applications Symposium on - IDEAS '16*, New York, 2016, pp. 302–307.

[12] J. M. Garro, P. Bazán, and J. Díaz, "Using BAM and CEP for Process Monitoring in Cloud BPM," *Journal of Computer Science & Technology*, vol. 16, no. 1, pp. 38–46, 2016.

[13] A. Baumgrass, C. Di Ciccio, R. M. Dijkman, M. Hewelt, J. Mendling, A. Meyer, S. Pourmirza, M. H. Weske, and T. Y. Wong, "GET Controller and UNICORN: Event-Driven Process Execution and Monitoring in Logistics," in *Proceedings of the Demo Session of the 13th International Conference on Business Process Management (BPM 2015)*, Innsbruck, 2015.

[14] F. M. Maggi, C. Di Francescomarino, M. Dumas, and C. Ghidini, "Predictive Monitoring of Business Processes," in *Advanced Information Systems Engineering. CAiSE 2014. Lecture Notes in Computer Science*, vol. 8484, Cham: Springer, 2014, pp. 457–472.

[15] S. Bülow, M. Backmann, N. Herzberg, T. Hille, A. Meyer, B. Ulm, T. Y. Wong, and M. Weske, "Monitoring of Business Processes with Complex Event Processing," in *Business Process Management Workshops. BPM 2013. Lecture Notes in Business Information Processing*, vol. 171, Beijing: Springer International Publishing, 2014, pp. 277–290.

[16] C. Cabanillas, C. Di Ciccio, J. Mendling, and A. Baumgrass, "Predictive Task Monitoring for Business Processes," in *Lecture Notes in Computer Science*, vol. 8659, Cham: Springer, 2014, pp. 424–432.

[17] E. Mulo, U. Zdun, and S. Dustdar, "Domain-Specific Language for Event-Based Compliance Monitoring in Process-Driven SOAs," *Service Oriented Computing and Applications*, vol. 7, no. 1, pp. 59–73, 2013.

[18] D. Knuplesch, M. Reichert, and A. Kumar, "A Framework for Visually Monitoring Business Process Compliance," *Information Systems*, vol. 64, pp. 381–409, 2017.

[19] J. M. Perez-Alvarez, M. T. Gomez-Lopez, L. Parody, and R. M. Gasca, "Process Instance Query Language to Include Process Performance Indicators in DMN," in *IEEE 20th International Enterprise Distributed Object Computing Workshop (EDOCW)*, Vienna, 2016, pp. 1–8.

[20] M. R. N. Mendes, P. Bizarro, and P. Marques, "Benchmarking Event Processing Systems," in *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering - WOSP/SIPEW '10*, New York, 2010, p. 259.

[21] A. Baouab, O. Perrin, and C. Godart, "An Optimized Derivation of Event Queries to Monitor Choreography Violations," in *Service-Oriented Computing. ICSOC 2012. Lecture Notes in Computer Science*, vol. 7636, Berlin, Heidelberg: Springer, 2012, pp. 222–236.

[22] C. Janiesch, M. Matzner, and O. Müller, "A Blueprint for Event-Driven Business Activity Management," in *Lecture*

*Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6896 LNCS, Berlin, Heidelberg: Springer, 2011, pp. 17–28.

[23] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Scalable Process Discovery and Conformance Checking," *Software & Systems Modeling*, pp. 1–33, 2016.

[24] A. Polyvyanyy, J. Vanhatalo, and H. Völzer, "Simplified Computation and Generalization of the Refined Process Structure Tree," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6551 LNCS, Berlin, Heidelberg: Springer, 2011, pp. 25–41.

[25] A. Baumgrass, N. Herzberg, A. Meyer, and M. Weske, "BPMN Extension for Business Process Monitoring," in *Enterprise Modelling and Information Systems Architectures (EMISA 2014)*, Luxembourg, 2014, pp. 85–98.

[26] M. Hirzel, "Partition and Compose: Parallel Complex Event Processing," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems - DEBS '12*, New York, 2012, pp. 191–200.

[27] F. Nguyen, D. Tovarňák, and T. Pitner, "Semantically Partitioned Peer to Peer Complex Event Processing," in *International Symposium on Intelligent Distributed Computing*, vol. 511, F. Zavoral, J. J. Jung, and C. Badica, Eds. Cham: Springer International Publishing, 2014, pp. 55–65.

[28] O. Saleh, H. Betz, and K.-U. Sattler, "Partitioning for Scalable Complex Event Processing on Data Streams," in *East European Conference on Advances in Databases and Information Systems and Associated Satellite Events*, Ohrid, 2015, pp. 185–197.

[29] C. Inzinger, W. Hummer, B. Satzger, P. Leitner, and S. Dustdar, "Generic Event-Based Monitoring and Adaptation Methodology for Heterogeneous Distributed Systems," *Software: Practice and Experience*, vol. 44, no. 7, pp. 805–822, 2014.

[30] W. M. P. van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Brocesses*. Berlin, Heidelberg: Springer, 2011.

[31] B. Fazzinga, S. Flesca, F. Furfaro, and L. Pontieri, "Online and Offline Classification of Traces of Event Logs on the Basis of Security Risks," *Journal of Intelligent Information Systems*, pp. 108–124, 2017.

[32] A. Rozinat and W. M. P. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," *Information Systems*, vol. 33, no. 1, pp. 64–95, 2008.

[33] O. Etzion and P. Niblett, *Event Processing in Action*. Greenwich: Manning Publications Co., 2010.

[34] D. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. Hoboken: John Wiley & Sons, 2011.

[35] E. Viel and H. Ueda, "Data Stream Partitioning Re-Optimization Based on Runtime Dependency Mining," in *IEEE 30th International Conference on Data Engineering Workshops*, Chicago, 2014, pp. 199–206.

[36] K. Isoyama, Y. Kobayashi, T. Sato, K. Kida, M. Yoshida, and H. Tagato, "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 '12*, New York, 2012, pp. 123–126.

[37] Y. Kobayashi, K. Isoyama, K. Kida, and H. Tagato, "A Complex Event Processing for Large-Scale M2M Services and Its Performance Evaluations," in *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems - DEBS '15*, New York, 2015, pp. 336–339.

[38] R. Pathak and V. Vaidehi, "An Efficient Rule Balancing for Scalable Complex Event Processing," in *IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, Halifax, 2015, pp. 190–195.

[39] M. Weidlich, H. Ziekow, J. Mendling, O. Günther, M. Weske, and N. Desai, "Event-Based Monitoring of Process Execution Violations," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6896 LNCS, Berlin, Heidelberg: Springer, 2011, pp. 182–198.

[40] W. Song, X. Xia, H.-A. Jacobsen, P. Zhang, and H. Hu, "Efficient Alignment between Event Logs and Process Models," *IEEE Transactions on Services Computing*, vol. 10, no. 1, pp. 136–149, 2017.

**Mohammad Ali Fardbastani** received his B.Sc. and M.Sc. from University of Tehran in 2010 and 2012 respectively, and now is a Ph.D. candidate in the School of Computer Engineering of Iran University of Science and Technology (IUST). His research interests include Distributed Complex Event Processing (DCEP) systems, scalable distributed and parallel High Performance Computing (HPC), and real-time systems. He is experienced in system-level development of operating systems, virtualization, runtime systems, distributed adaptive resource sharing, and communication middleware for HPCs and DCEPs.

**Farshad Allahdadi** received his B.Sc. from University of Yazd in 2013 and his M.Sc. from IUST in 2015, and now is a researcher at Distributed Systems Lab in the School of Computer Engineering of Iran University of Science and Technology (IUST). His research interests include Distributed Complex Event Processing (DCEP) systems, scalable distributed and parallel High Performance Computing (HPC), and NoSql Databases.

**Mohsen Sharifi** received his B.Sc., M.Sc. and Ph.D. in Computer Science from the Victoria University of Manchester in the United Kingdom in 1982, 1986, and 1990, respectively. At present he is a Professor of System Software Engineering at the School of Computer Engineering of Iran University of Science and Technology. He directs a Distributed Systems research group and laboratory. His main interest is the development of distributed systems, solutions, and applications, particularly for use in various fields of science. He has developed a high performance scalable cluster solution comprising any number of homogeneous or heterogeneous COTS computers for use in scientific applications requiring high performance, availability and scalability. The development of a true distributed operating system is on top of his wish list.