Complex Event Processing in Sensor-Based Environments: Edge Computing Frameworks and Techniques

A. Dhillon, S. Majumdar, M. St-Hilaire, and A. El-Haraki

Abstract By performing latency-sensitive computations at the edge and the remaining computations on a backend server, edge computing systems can effectively handle the processing of data in a timely manner. This chapter focuses on an edge computing framework that partitions the processing of sensor data at a mobile node placed at the edge and backend computations at a powerful server. The primary application of the framework is in the area of processing of complex events each of which may correspond to the simultaneous occurrence of multiple raw events generated by sensors that are monitoring the phenomena of interest. Application of such complex event processing techniques spans smart buildings, smart machinery as well as smart healthcare systems. This chapter focuses on using the proposed framework and techniques to a smart phone based remote patient monitoring system and by using prototyping and measurement presents a rigorous performance analysis of the system.

Keywords Mobile complex event processing · Remote patient monitoring system · Internet of things · Smart healthcare

1 Introduction

Data acquisition and the processing of the acquired data are two components of various computing applications. Traditionally, they have been performed by two separate system components. The data handling components that perform inputting/outputting of data send the data to another processing node that runs the data processing component and sends the results back to the data handling

A. Dhillon · S. Majumdar · M. St-Hilaire (✉)
Carleton University, Ottawa, ON, Canada
e-mail: amarjitdhillon@sce.carleton.ca; majumdar@sce.carleton.ca; marc_st_hilaire@carleton.ca

A. El-Haraki
TELUS, Ottawa, ON, Canada
e-mail: ali.el-haraki@telus.com

A. Mukherjee et al. (eds.), *Mobile Edge Computing*,
https://doi.org/10.1007/978-3-030-69893-5_20

components. Examples include systems that use sensors (actuators) for data handling and a backend server for analyzing the sensor data. The intercommunication with the backend server is often achieved with the help of an inter-communication network which can introduce significant inter-communication delays. This model in which data handling and data processing are done by two separate components is adequate for delay tolerant systems for which the latency of data processing is not a concern. It fails, however, for delay sensitive systems where the results of processing sensor data must become available within a short period of time. Examples include sensor-based remote patient monitoring systems, various types of industrial controllers and aerospace systems that must quickly react to the sensor data crossing a particular threshold. Using a multi-tiered edge computing system in which a part of the data processing is performed at the edge near the data handling device and the remaining processing on the backend server is crucial for producing the results in a timely manner and achieving the latency goals of the system. The availability of inexpensive sensing devices as well as small computing systems is fuelling the rapidly increasing deployment of such edge computing systems.

This book chapter focuses on a mobile edge computing framework that is applicable to various smart systems that are described in the next paragraph. The application of the framework and associated techniques for a real-time remote patient monitoring system that includes a mobile edge computing device connected to sensors and a backend server is described. The system uses mobile edge computing and Internet of Things (IoT) technologies to perform complex event processing for detecting an oncoming health problem for the patient being monitored.

Complex Event Processing (CEP) is the technique used to find the patterns in real time data streams. This chapter compares two CEP architectural frameworks: Server CEP (SCEP) and Mobile CEP (MCEP). The SCEP framework uses the mobile device as a gateway to forward data streams from sensors to a remote IoT server where complex events are detected. A drawback of this existing methodology is that the mobile phone always needs to remain connected to the back-end server. Also, the mobile device's network consumption is increased while transferring large volumes of sensor data streams leading to an increase in the user cost. Additionally, it leads to an increase in the workload at the back-end server that serves multiple users. In the MCEP framework, as briefly introduced in [10], the detection of complex events is performed on an edge device (such as a smart phone) that receives data from sensors. Only the detected complex events are sent to a back-end IoT server for further processing. The edge-based technique can be used in various cases such as smart home, smart building and Remote Patient Monitoring (RPM). In this chapter, a RPM use case is considered to validate and compare the two frameworks. A thorough performance analysis is performed using a synthetic workload which provides insights into system scalability and the relationship between system/workload parameters and performance. This technique can be adapted to handle various different use cases as well.

1.1 Overview of the Chapter

This section provides a short overview of the material presented in this chapter. Section 2 describes a representative set of related work and Sect. 3 discusses the system architecture for the server CEP system. Then, the architecture of the mobile CEP system is discussed in Sect. 4. Implementation details for the proof of concept prototype are discussed in Sect. 5. Section 6 presents a performance analysis of the system followed by experimental results in Sect. 7. Finally, Sect. 8 provides our conclusions and Sect. 9 outlines possible directions for future work.

2 Related Work

A representative set of works on CEP and smart healthcare systems is presented. A more detailed literature survey is available from [9].

In 2016, Higashino proposed the idea of CEP-as-a-Service (CEPaaS) in his Ph.D. dissertation [18]. The goal is to leverage the advantages of Software-as-a-Service (SaaS) to provide Complex Event Processing as-a-Service (CEPaaS) so that there is no upfront charges and maintenance cost is low. He proposed Attributed Graph Rewriting for Complex Event Processing (AGeCEP) as a language agnostic technique to model the Continuous Query Language (CQL) queries. To support his proposition for CEPaaS, Higashino designed a simulator called CEPSim that runs on top of the CloudSim simulator [5, 6]. CloudSim is a popular cloud simulator written in Java which can effectively model a public, private or hybrid cloud. It allows the users to create a data-center, cloudlet, and broker in addition to defining different policies. The CEPSim module creates a query model and supports the operator placement and the operator scheduling for performing the CEP simulation. It also provides the mechanism to compute various CEP specific metrics for performance evaluation. A major limitation of CEPSim is that it does not have single and multiple query optimization mechanisms and assumes that a submitted query is already optimized. Another limitation is that it only supports the scenarios in which the query does not fail at runtime. It is important to mention that our work compares the performance of the edge-based mobile CEP with state-of-the-art CEPaaS system considered as a baseline system.

Another work reported in [25] describes a pulse monitoring system which also used the Android application as an edge gateway and sends data to a web portal for analysis and visualization. A similar approach is described in [31] which uses an Android device as a gateway agent. Another research in [11] and [26] employed an IoT-based approach to process the health sensor data streams on the cloud. The authors have used an *Intel Galileo Gen 2* IoT agent to collect the sensor data streams from the mobile device and forward these to an IoT server deployed on the cloud. However, the authors have not used any real-time analytics system as the computation is done by a batch processing-based Hadoop system. Further,

no performance analysis is done in any of these two papers to demonstrate the effectiveness of the technique.

Woodbridge et al. have proposed an RPM system for congestive heart failure named as *WANDA* [30]. *WANDA* has a three-tier architecture in which the first tier consists of various health sensors that transmit the health sensor data streams to the second tier consisting of a web server. The third tier uses database servers to persist the health sensor data streams and perform the analysis using linear regression. Further, this system is not a real-time system and does not involve any CEP engine. However, as the authors are predicting a heart stroke, performing batch analysis seems to be appropriate. In 2017, Naddeo et al. [22] have proposed a real-time m-health monitoring system. Their system consists of an Android application which receives various physiological sensor data using the *Zephyr Bioharness BH3* sensors and performs noise filtering using various high-pass and low-pass filters. This filtered data is sent by an Android application to a remote Personal Health Record (PHR) server for analysis and visualization. A major shortcoming of this paper is that it does not describe the real-time analysis technique required for this system. Another similar work is reported in [23] where the authors proposed to integrate the CEP engine and the IoT server for smart healthcare. This paper is primarily focused on the key benefits of using CEP on the cloud. However, no actual system is designed and no performance analysis is done.

More recently, several survey papers such as [17], are bridging the concepts of edge computing and healthcare. The paper by Abdellatif et al. [1] is of particular interest as it reviews the opportunities and challenges for enabling smart healthcare (s-health). They mention that edge-computing capabilities and next-generation wireless networking technologies will be the enablers to achieve this goal. One of the interesting functionalities that their architecture provides is called "edge-based feature extraction for event detection". Our work is one step in this direction. By performing latency sensitive computations at the edge and the remaining computations on a backend server, we can ensure fast response time for critical applications such as remote patient monitoring.

Table 1 shows a summary of the various techniques presented in this section along with the two proposed techniques (SCEP/MCEP) described in this chapter. The comparison is based on the following parameters:

1. Simulation/Prototype/Concept/Review: This parameter indicates the methodology that was used in the papers. Four options are possible: 'Simulation' means that the performance of the model was evaluated through simulation. Similarly, 'prototype' means that a proof of concept was implemented and evaluated. 'Concept' denotes a paper where only a high-level description of the concept is presented and 'review' designates a review paper where multiple techniques are reviewed.
2. Edge/back-end: This parameter shows whether the complex event processing is done on the edge mobile device or on a back-end server.
3. Gateway/Filter: This parameter shows the technique used to forward the health data to the back-end server. 'Gateway' signifies that the mobile device is used as

Table 1 Comparison of various techniques based on different parameters

Technique/paper	Simulation/ prototype/ concept/ review	Edge/ back-end	Gateway/ filter	Security	Cost	Performance analysis
SCEP	Prototype	Back-end	Filtering	Yes	Yes	Yes
MCEP	Prototype	Edge	Filtering	Yes	Yes	Yes
ARM7 [25]	Prototype	Back-end	Gateway	Yes	No	No
eHealthNet [22]	Prototype	Back-end	Gateway	No	No	Yes
WANDA [30]	Prototype	Back-end	Gateway	Yes	No	Yes
[31]	Prototype	Back-end	Gateway	No	No	No
[11]	Prototype	Back-end	Gateway	Yes	No	No
[26]	Prototype	Back-end	Gateway	Yes	No	Yes
[1]	Prototype	Back-end	Filtering	Yes	No	Yes
AGeCEP [18]	Simulation	Back-end	Gateway	No	No	Yes
[23]	Concept	Back-end	Gateway	Yes	No	No
[17]	Review	n/a	n/a	n/a	n/a	n/a

a gateway to forward all the sensor data whereas 'Filter' signifies that data has been reduced (filtered) by the mobile device to reduce user cost and data transfer latency.

4. Security: This parameter is 'yes' if various security related issues have been considered in the paper.
5. Cost: This parameter is 'yes' if a cost related analysis is provided in the paper.
6. Performance analysis: This parameter shows whether a rigorous performance analysis is provided.

From the comparison provided in Table 1, we can see that unlike other methodologies which perform the CEP analysis on the back-end IoT server, our proposed techniques (MCEP and SCEP) can process data on the edge and on the back-end server respectively. It is worth mentioning that authors in [1] have done data compression and edge-based feature extraction on the edge device. However, in our work, we have done complete complex event detection on the device itself. Also, various security features have been implemented in our proof of concept prototype to help insure integrity and safety of patient health data. As compared to most of the other papers, some of which are missing performance analysis, cost analysis and security features, the proposed techniques consider these factors into account.

3 Server CEP System

In this chapter, we have considered the remote patient monitoring use case. As shown in Fig. 1, the Server Complex Event Processing (SCEP) system architecture

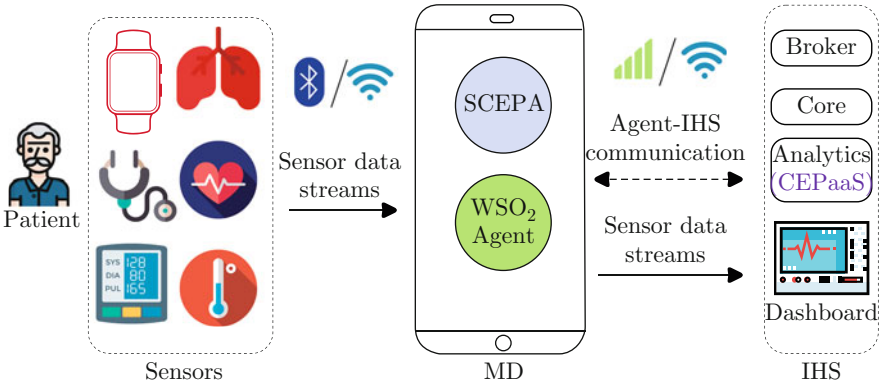

Fig. 1 Server CEP system architecture

is three-tiered consisting of multiple sensors, a Mobile Device (MD), and an IoT Hospital Server (IHS).

The mobile device along with the sensors comprise the edge system that communicates with the centralized back-end server. Multiple bluetooth and WiFi enabled wireless sensors can be used by the sensor-based system which can forward the sensor data to an Android or iOS device. For example, in a remote patient monitoring system, the sensors can be wearable health sensors worn by the patient. Such cheap and efficient sensors are provided by Cooking Hacks for example [7]. Some other commercial health monitoring sensors that can be used include the *Zeo Sleep Monitor* [13], which monitors sleep disorders, and *ViSiMobile* [29] which can measure Electrocardiogram (ECG), Heart Rate (HR), Arterial Oxygen Saturation (SpO2), skin temperature, etc. As shown in Fig. 1, the multiple sensors send the sensor data streams to a mobile device which consists of a Server Complex Event Processing Application (SCEPA) and a WSO_2 agent gateway application. The WSO_2 agent is used to register the mobile device with the IoTs server. The server complex event processing application forwards the health sensor data streams to the IHS. Communication between the sensors and the mobile device is done using bluetooth or WiFi whereas data transmission between the mobile device and the IHS is performed using either a cellular or a WiFi connection. The architecture shown in Fig. 1 can be used in other use cases such as smart buildings and smart homes as well. In the smart building use case, the wearable health sensors can be replaced by wired/wireless sensors deployed in a smart building such as room temperature sensors and light intensity sensors. In such a case, the mobile device can be replaced by a local server or a Raspberry-Pi board depending upon the workload.

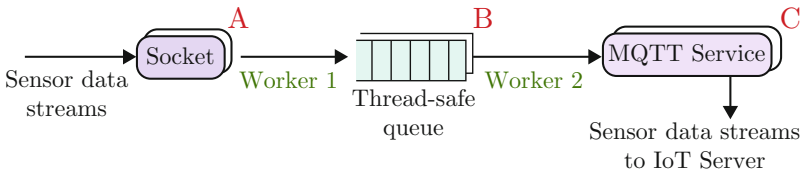

Fig. 2 Various components of the SCEP application

3.1 Components of the SCEP Application

Figure 2 shows the components of SCEPA which is used to forward the raw sensor data streams from the mobile device to the IoT server.

The various components that are stacked over one another represent multiple parallel instances of that component and a solid line represents multiple parallel sensor data streams. The various data streams are received by the Transmission Control Protocol (TCP) socket objects (one socket for each sensor) and appended to a thread-safe linked-blocking queue by a producer thread (Worker 1). A dedicated thread-safe queue is used for each sensor data stream. Further, the dequeue worker (Worker 2) retrieves the sensor data stream from a queue and sends it to the IHS using the Message Queuing Telemetry Transport (MQTT) service running on the mobile device. The MQTT protocol is used here as it is made specifically for low power devices such as sensors and mobile devices [20]. This MQTT service forwards the sensor data streams to the back-end IoT server as per the selected Quality of Service (QoS). Please note that the MQTT service also has its own queues for enabling the persistent session, and if the QoS ≥ 1 is selected, the sensor data stream tuples are temporarily persisted in case the back-end server goes offline.

3.2 Components of CEP-as-a-Service

This section discusses the various components of the CEPaaS module which is running on the IoT server.

As indicated earlier, a solid line represents multiple parallel sensor data streams whereas a dashed line represents a single sensor data stream. Each component which is shown as a box in Fig. 3 receives an input data stream and emits an output data stream as a result of the operation performed by that component. Thus, various output streams must be defined before starting the service such that an output stream contains all the attributes which have been emitted by its predecessor component. When an attribute is added or removed from an input data stream (RE.v.1 for example) as a result of an operation done by a component (MQTT receiver in this case), then the output stream can be referred to as a stream having a different version (RE.v.2 in this case). As shown in Fig. 3, a raw stream has 9 versions (RE.v.1

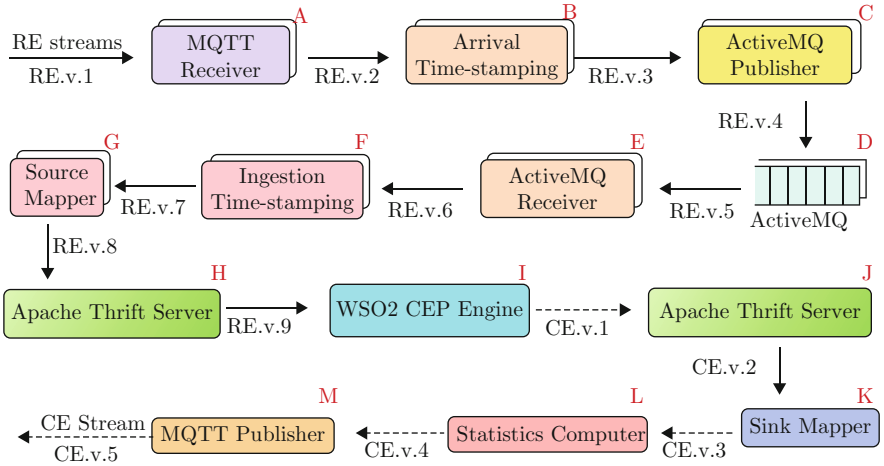

Fig. 3 Components of the CEPaaS module

to RE.v.9) whereas a complex event stream has 5 versions (CE.v.1 to CE.v.5). A brief discussion of each component is provided next in the order of the processing performed.

(A) MQTT Receiver: It receives a raw sensor stream on a particular topic after validating the content using the default/custom content validator. Multiple instances of the MQTT receivers (one for each sensor stream) receive raw sensor data streams in parallel.

(B) Arrival time-stamping: Multiple arrival time-stamping components run in parallel. Each component receives a particular stream and appends a system generated nanosecond precision time-stamps to indicate the arrival time.

(C) ActiveMQ publisher: An ActiveMQ [28] is used as a Java Message Service (JMS) queue [16]. The ActiveMQ publisher is responsible for sending the messages to a particular brokered-queue managed by an ActiveMQ broker. ActiveMQ supports both topics and brokered-queues to transfer messages, but we are using the brokered-queue in this case. For setting a JMS publisher, the various adapter properties such as JMS destination type, JMS destination name, JMS factory name, JMS provider Uniform Resource Locator (URL), JMS Connection Factory name, Java Naming and Directory Interface (JNDI) name, a username and a password need to be defined as per ActiveMQ server configurations which is running on the IoT server.

(D) ActiveMQ: ApacheMQ provides support for Advanced Message Queuing Protocol (AMQP), Streaming Text Oriented Message Protocol (STOMP), MQTT, OpenWire [2] and other protocols. The size of each ActiveMQ queue size is set to a maximum of 2 GB (restrained by the maximum value of an integer). A web-based Graphical User Interface (GUI) can be used to view the

list of all ActiveMQ queues, topics and the number of messages enqueued/dequeued in each of the queue/topic.

(E) ActiveMQ subscriber: It is used to receive the sensor data stream events from a particular ActiveMQ queue. A subscriber subscribes to a particular queue using a unique queue name identifier and then forwards the received sensor tuples as an output sensor data stream (RE.v.6 in this case).

(F) Ingestion time-stamping: This module is used to append the CEP engine ingestion time-stamps using a nanosecond precision system clock, before sending the sensor data streams to the CEP engine. Multiple ingestion time components work in parallel to time-stamp each sensor stream.

(G) Source mapper: A CEP system supports various event formats such as eXtensible Markup Language (XML), JavaScript Object Notation (JSON), key-value pairs and Health Level-7 (HL7). The role of the source mapper is to convert the type of the sensor data stream event to the format required by the CEP engine.

(H) Apache thrift server: It is the binary communication protocol originally developed by Facebook [27]. It provides a Remote Procedure Call (RPC) framework to build the cross-platform services written in different frameworks and languages [12]. WSO_2 Data Analytics Server (DAS) running inside the analytics tier provides real-time, batch and predictive analytics by using the other services such as the CEP engine and Apache Spark. Thus, the Apache thrift acts as a mediator to perform RPC on the CEP engine using the data bridge agent.

(I) CEP engine: It receives multiple sensor data streams and finds the complex events according to the CQL query which has been deployed. A single complex event stream, as shown by a dashed line, is sent to the sink mapper. The complex event detection time-stamping is done in the CEP engine.

(J) Apache thrift server: The detected complex events are sent back to the thrift server which sends them back to the data analytics server for further processing.

(K) Sink mapper: The sink mapper converts the data type of the events in CEP stream to the type required by the event publisher.

(L) Statistics computer: It computes various CEP specific metrics such as the average CEP latency by using the time-stamps taken by the IoT server.

(M) MQTT publisher: The MQTT broker component publishes the various streams to the event listener such as a dashboard, email, or a database.

4 Mobile CEP System

The mobile CEP system prototype has been designed to perform complex event detection on the edge device using an embedded CEP engine that forwards the complex events to an IHS. Although the following discussion refers to the RPM use case, the MCEP architecture can be used in the context of other use cases as well. As shown in Fig. 4, similar to the SCEP architecture, the MCEP architecture also consists of three components.

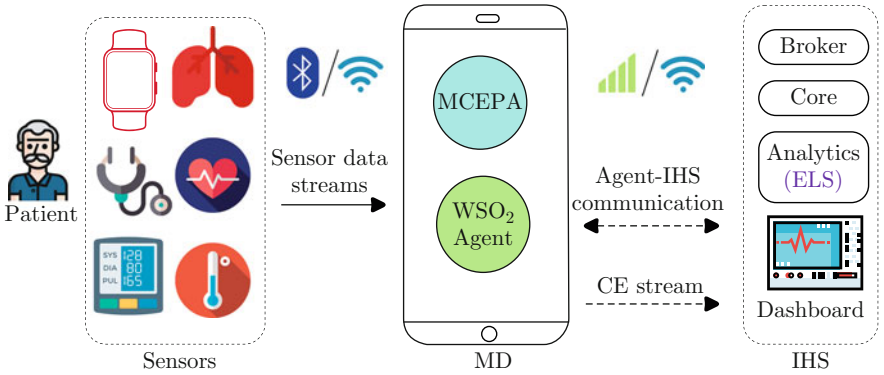

Fig. 4 Mobile CEP system architecture

1. Sensors: For the RPM use case, various wearable health sensors such as an Apple watch, Glucometer sensor, and Pulse-oximeter are used.
2. Mobile device: The mobile device uses Apache Siddhi CEP engine embedded with the mobile CEP application to perform complex event detection and sends the detected complex events to the back-end hospital server.
3. IHS: An Event Listening Service (ELS) running on the analytics tier receives the complex event alerts which are then sent to a DataDog dashboard [8] to notify the hospital staff.

The main difference between the Mobile Complex Event Processing (MCEP) system and the SCEP system is that in the mobile CEP system all the complex events are detected on the mobile device instead of processing them on a centralized IoT server. Unlike the SCEP system, which has a CEP running on the IHS, the MCEP system has an event listening service (ELS) running on the hospital server which is subscribed to listen to the complex events sent by the Mobile Complex Event Processing Application (MCEPA) running inside the mobile device.

5 Experimental Setup

The experimental setup used for analyzing the performance of both the MCEP and SCEP systems is discussed in this section. As shown in Fig. 5, the setup for both the server CEP and mobile CEP systems consist of five components: a timekeeper, a Sensor Simulator (SS), an IoT hospital server, a mobile device, and a wireless router.

Note that the timekeeper used in the experimental setup for performance measurement is not needed in a production system in which sensor simulator is replaced by the actual senor devices. The timekeeper is used to perform global time-

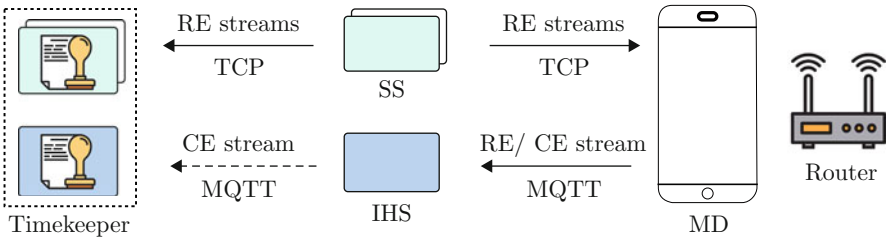

Fig. 5 System prototype setup

stamping to compute the end-to-end latency. This module is required as various components (with different un-synchronized clocks) contribute to the computing of the end-to-end latency. Therefore, a timekeeper is required to provide a global time-stamping for raw event streams (coming from the sensor simulator) and the complex event stream (from IHS) using a single clock. The various components which are stacked over one another inside the sensor simulator and the timekeeper represent multiple instances of the respective component running in parallel. The solid line represents multiple parallel data streams while a dashed line represents a single sensor data stream. As shown in Fig. 5, the data streams generated by the sensor simulator are sent in parallel to both the mobile device and the timekeeper (for global generation of time-stamping). In the SCEP system, raw event streams are sent from the mobile device to the IoT server whereas only a complex event stream is sent from the mobile device to the IoT server for the MCEP system. For both architectures, a single complex event stream is sent from the IHS to the timekeeper for global notification time-stamping. The system configuration for the aforementioned components is provided next.

1. Timekeeper: The timekeeper module is written in Java and deployed on a computer workstation having 16 GigaByte (GB) of RAM, a 2.8 GHz Intel Core i7 processor and a 1 TB Hard Drive (HD) running on Ubuntu 14.04 Long Term Support (LTS).
2. Sensor Simulator: The Java-based sensor simulator program is running on a workstation equipped with 8 GB of RAM, a 2.8 GHz Intel Core i7 processor and a 1 TB HD using Ubuntu 14.04 LTS. A multi-threaded sensor simulator program is used to simulate multiple sensors generating data at a given input rate. A nanosecond sleep time is used to generate a constant inter-arrival time for each sensor. As shown in Fig. 5, the data streams generated by the sensor simulator are sent simultaneously to the mobile device and the timekeeper using TCP sockets. All the sensor simulator daemons send data streams concurrently on separate threads, where each thread generates a stream of JSON tuples. A JSON tuple consists of both metadata and payload data. The metadata includes information such as patient id, sensor id and tuple id whereas the payload data includes the respective sensor value(s) and an event generation time-stamp (Tg). In certain

cases, the sensor data stream tuple may consist of an array of data values instead of a single value, but in our experimentation, a single value is used. Patient_{id} is required at the IHS in order to uniquely identify a patient when multiple patients are enrolled with the RPM service. Also, a combination of Patient_{id}, Sensor_{id}, and Tuple_{id} can be used to uniquely identify an event received at the IHS when multiple patients are enrolled. The sensor simulator can generate both synthetic and real data using synthetic and real datasets respectively. For simulating the real data, the sensor simulator uses sensor data available at the *slp01a/slpdb* dataset from the MIT-BIH polysomnographic database [14]. This dataset consists of 2-h duration data of 4 health signals recorded at 250 Hz. In a synthetic dataset, the tuple values are uniformly distributed integers ranging from 1 to 100. The real dataset was used to test the functional correctness of the proof of concept prototype, whereas all the other experiments were performed using a synthetic dataset. For performance analysis, a synthetic dataset is preferred over a real dataset because of the ability to control the various workload parameters including tuple values and tuple inter-arrival times.

3. IHS: An IoT server is deployed on a workstation having 16 GB of RAM, a 3.5 GHz Intel Core i7 Processor and a 1 TeraByte (TB) Solid State Drive (SSD) running under High Sierra MacOS. The MQTT broker and the MQTT subscriber are deployed on the broker and the analytics tiers of the IoT server respectively. The Java Virtual Machine (JVM) configurations for the broker, core, and analytics components of IHS used in the prototype are given in Table 2. Setting the configurations helps to dedicate the CPU resources to each component such as broker, core and analytics. Here, −Xmx represents the maximum size of the JVM heap (4 GB in this case) which can be allocated to the respective tier.
4. Mobile Device: A Google Pixel smart-phone [32] having 4 GB of RAM, 32 GB of storage, and an AArch64 quad-core processor (1.6 GHz) running Android Nougat is used as the mobile device. WSO_2 IoT server version 3.0 is deployed on IHS along with its compatible Android agent version 3.1.27 running on the mobile device. Both the mobile CEP and server CEP applications that are written using Java are built on Android Studio 3.0.1 IDE using Gradle build tools version 26.0.2 [21]. For the mobile CEP application, due to the large size of the Siddhi CEP libraries, the multidex feature has to be enabled to overcome the 64K limit of the Android Dalvik compiler. Relevant Internet, WiFi, and network permissions must be enabled for the MCEP and the SCEP applications. The MQTT publisher is deployed on the mobile device.
5. Router: A 5 GHz AC1750 Tp-Link dual-band wireless router with a maximum bandwidth of 1350 Mbps is used to transfer data between the various components.

Table 2 Java memory configurations

Parameter	Broker	Core	Analytics
−Xmx	4096 MB	4096 MB	4096 MB

6 Performance Analysis

6.1 The Complex Event Use Case Modeling

A survey conducted by World Health Organization (WHO) reported that the occurrence of fall is common among elderly people and seems to increase with age and frailty level. In accordance to this survey, each year approximately 28–35% people more than 65 years of age fall whereas this number reaches to 32–42% for 70 years old [24]. Falls lead to 20–30% of mild to severe injuries and are the underlying cause of 10–15% of all emergency department visits [24]. However, if a fall is notified to hospital staff as soon as possible, further loss can be circumvented. Fall detection can be monitored remotely using a combination of mobile sensors and physiological sensors. Mobile sensors used for fall detection include a mobile camera, accelerometer sensor, gyroscope sensor and Global Positioning System (GPS) sensor. The various physiological sensors include heart rate and respiration rate sensors [15]. In a simpler case, a fall can be identified with more certainty, if events happen in certain order for example, a fall event followed by an increase in heart rate event followed by a reduction in body movement event. Detecting the occurrence of the fall event can be done using the gyroscopic sensor as well as phone camera whereas patient's reduction in body movement can be detected by a combination of an accelerometer sensor and a Global Positioning System (GPS) sensor. Another event that indicates that the person has not responded to a call from the hospital staff within specific time can confirm the fall event.

6.2 Workload and System Parameters

The various workload and system parameters used in analyzing the performance of the SCEP and MCEP prototypes are described next.

- Average raw event arrival rate (λ_{RE}): It is the average rate of the raw events generated by the sensor simulator.
- Threshold for sensor stream x (Th_x): The value of the threshold parameter is used by the selection predicate (π) to filter the sensor data streams tuples which are greater than Th_x.
- $Count_x$: Count is used to specify the number of times a particular event has to occur. An exact number of occurrences can also be specified through the count parameter. We have used the $\langle min{:}max \rangle$ specifier for $Count_x$ where $\langle min{:} \rangle$ means that an event has to happen at-least min times while no upper bound is specified. In other words, the notation $\langle$min:max$\rangle$ means that the event should happen at least min times but less than max times.
- Time window (T_{win}): The time window specifies the maximum time for which event A will wait for event B to occur. Please note that this time will be different for each instance of the state machine. The time window starts as soon as event

Table 3 Workload and system parameters

Parameter	Description	Units
λ_{RE}	200, 300, **500**, 1000, 2000	Events/second
Th_x	10, 30, **50**, 70, 90, 99	–
$Count_x$	1, 5, **10**	–
T_{win}	0.005, 0.035, 0.06, 0.1, 0.2, **10**	Seconds
T_{run}	**5**, 60	Minutes

A arrives at the CEP system. Then a separate instance of the state machine is started and it waits for event B for a time less than T_{win}.

- Simulation runtime in minutes (T_{run}): It is the length of the simulation runtime in minutes.

The various values for the workload and system parameters used in the experiments are presented in Table 3. Factor-at-a-time experiments were performed on the system in which one parameter was varied in a given experiment while others were held at their default values. The value in bold for each parameter presented in Table 3 corresponds to the default value of the parameter.

6.3 Performance Metrics

The CEP specific performance metrics used in the analysis are the average CEP latency (L) and the average complex event End-to-End (E2E) latency (E). Application specific metrics are average CPU utilization and average network usage. An application profiler such as Trepn, PowerTutor or Intel Performance Viewer [3] can be used to perform system level and application level performance profiling. However, the accuracy of these applications is a concern, thus various application metrics have been calculated using a bash script which reads $dumpsys$ information using Android Debug Bridge (ADB) shell. This script reads various application and system specific metrics and parses this information using a combination of various $grep$ commands, regular expressions, awk scripts and sed expressions.

Let T_a^x and T_i^x be the arrival time and ingestion time respectively for the earliest arriving event, among all the events from the different sensor data streams that led to the complex event. Let T_g^x and T_{gg}^x be the generation time and global generation time respectively for the earliest arriving events that corresponded to the complex event. Also, let T_d^x, T_n^x and T_{gn}^x represent the complex event detection time, complex event notification time and complex event global notification time respectively. Below, we discuss how the various metrics are computed.

- Average CEP latency (L): A complex event is generated when a CQL pattern match occurs by ingesting data from multiple sensor data streams. The latency of a complex event processing is measured from the time of ingestion (T_i) for the first event (from any sensor data stream) that leads to the complex event to the time at which the complex event gets detected (T_d). If the total number

of complex events detected during an experiment is N, then the average CEP latency is given by Eq. (1).

$$L = \frac{\sum_{x=1}^{N} T_d^x - T_i^x}{N} \tag{1}$$

The average CEP latencies for the MCEP and SCEP systems are represented by L_{MCEP}, and L_{SCEP} respectively.

- Average complex event E2E latency (E): It is the average time taken by an event (which corresponds to the earliest raw event leading to a complex event) from the time it is generated by the sensor simulator (T_g) to the time it is notified at the IoT server (T_n). However, as discussed earlier, T_g and T_n are time-stamped in the sensor simulator and the IoT server respectively using clocks that are not synchronized with one another. Thus, E is computed using T_{gg} and T_{gn} (instead of T_g and T_n) both of which are time-stamped on the timekeeper module. E is computed using Eq. (2), where T_{gg}^x and T_{gn}^x represent the global generation time for the x^{th} raw event that corresponds to a complex event and the global notification time for the xth complex event, both time-stamped at the timekeeper.

$$E = \frac{\sum_{x=1}^{N} T_{gn}^x - T_{gg}^x}{N} \tag{2}$$

The average E2E latency for the MCEP and SCEP systems is represented by E_{MCEP} and E_{SCEP} respectively. A diagram showing the relationship among CEP specific metrics L, Q (complex event queuing delay), and E is presented in Fig. 6. In this figure, the multiple instances of input sensor data streams (one for each sensor) are shown in parallel such that tuples in the nth sensor data stream (where $n \in 1 \ldots y$) are denoted by T_a^n and T_i^n as arrival time and ingestion time respectively. However, as the complex event is generated from a pattern which ingests multiple sensor data events, only one complex event is shown on the right-hand side of Fig. 6.

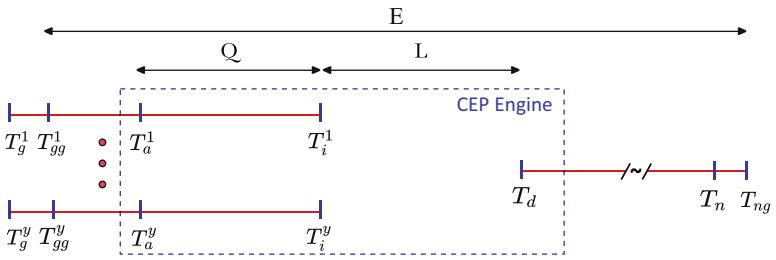

Fig. 6 CEP specific metrics

- Average CPU utilization (CU): It is the average CPU utilization by the mobile application during an experiment. CU_{SCEPA} and CU_{MCEPA} represent the average CPU utilization for the SCEP application and MCEP application respectively. The application is un-installed and installed again for each experiment.
- Average CPU utilization by IHS (CU_{IHS}): CU_{IHS} represents the average CPU utilization of the IoT server. $CU_{IHS-SCEP}$ and $CU_{IHS-MCEP}$ represent the average CPU utilization by the IHS for the SCEP system and MCEP system respectively.
- User cost (UC): The UC is the average cost (in \$/hour) by the user for using the CEP service. UC_{SCEP} and UC_{MCEP} represent the UC for using SCEP service and MCEP service respectively. Assuming that a user (patient) is using bluetooth or WiFi for connecting the sensors with the mobile device, TX can be used to compute the user cost. Here, we assume that a patient is using the mobile network for the transfer of data between the mobile device and the back-end IoT server. The user cost can be computed by as:

$$\text{User Cost } (\$/hour) = TX * \text{cost per MB} * 3600 \tag{3}$$

- Remaining Battery Life (RBL): It is the amount of remaining battery power (in %) by the application running on the mobile device during an experiment. It is an important metric representing the power consumption of an application. The different types of RBL used in the experimentation are provided next.
  - The $RBL_{SCEPA-FG}$ and $RBL_{MCEPA-FG}$ represent the battery usage for the server CEP and mobile CEP applications respectively when these applications are running in the foreground fn the mobile device and no other service is running on the background.
  - The $RBL_{SCEPA-BG}$ and $RBL_{MCEPA-BG}$ represent the battery usage for the server CEP and mobile CEP applications respectively when these applications are running in the background of the mobile device and no other application is running on the foreground.

7 Experimental Results

In this section, the performance comparison between the MCEP and SCEP systems is presented.

7.1 Comparison of Battery Usage

The impact of T_{run} on the power consumption of the MCEP and SCEP applications is presented in Fig. 7.

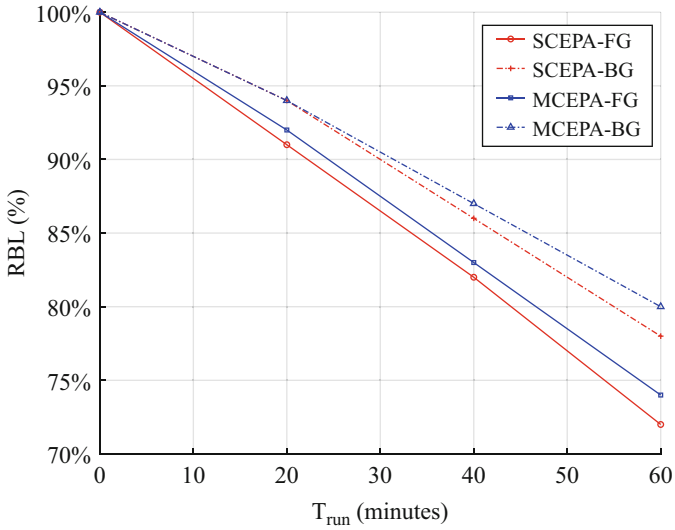

Fig. 7 Impact of runtime on battery usage

The experiment was performed for 60 min with an initial battery level of 100%. During the experiment, the values of the battery level on the mobile device were noted every 20-min interval, as shown by T_{run} in Fig. 7. Recall that scenario 1 corresponds to SCEP/MCEP application running in the Foreground (FG) and no other application running in the background. In scenario 2, the SCEP/MCEP application is running in the Background (BG) with no other application running in FG on the mobile device. It is found that the battery usage of an application for a scenario 1 is always lower in comparison to scenario 2. Also, for a given scenario the battery usage for the MCEP application is lower than that for the SCEP application. This is due to the fact that only complex events are transferred to the IoT server when the MCEP application is used. On the other hand, all the raw events (from multiple sensors) are forwarded to IoT server when the SCEP application is used, causing an increase in the battery consumption. The energy consumption due to data transfer is higher in comparison to the energy consumption due to running the CEP engine on the mobile device. This experiment shows that the proposed MCEP system provides approximately 2% power savings (both in background and foreground), in comparison to the SCEP system.

7.2 Comparison of Average CEP Latency

As shown in Fig. 8, for a particular λ_{RE}, the average CEP latency for the SCEP system is much higher than the average CEP latency for the MCEP system. For

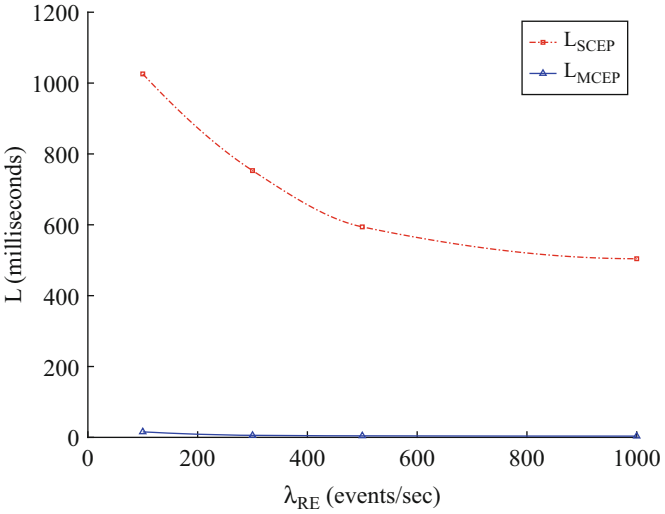

Fig. 8 Impact of the arrival rate of raw events on average CEP latency

both MCEP and SCEP systems, the average CEP latency decreases with an increase in the average raw event arrival rate. This is because, with an increase in λ_{RE}, the inter-arrival time of the event B is reduced. This led to a decrease in the waiting time of the A events in the CEP engine, resulting in the lower values of L_{SCEP}. In the case of server CEP, the data analytics server uses Apache thrift as a middle-ware to send the requests to the CEP engine using remote method invocations, causing the additional delays. This results in a higher CEP latency for SCEP in comparison to the MCEP system which does not use a middleware system. This leads to the important conclusion that there is a trade-off between security and latency for the SCEP system. Although enabling additional features in the IoT server provides more security, it also leads to a significant increase in CEP processing latency.

7.3 Comparison of Average End-to-End Latency

The end-to-end latency depends upon various factors such as the sum of various transmission times, queuing delays and event processing latencies. As shown in Fig. 9, as λ_{RE} is increased, more complex events are detected per unit time for both MCEP and SCEP systems.

This seems to increase the resource contention resulting in an increase in the transmission delay (as more complex events will be sent to the timekeeper) and the queuing delay (see [9] for an analysis on the queuing latency) leading to an increase in the average end-to-end delay. For a given λ_{RE}, the end-to-end delay for the SCEP system is higher than that for the MCEP system. Forwarding all the raw

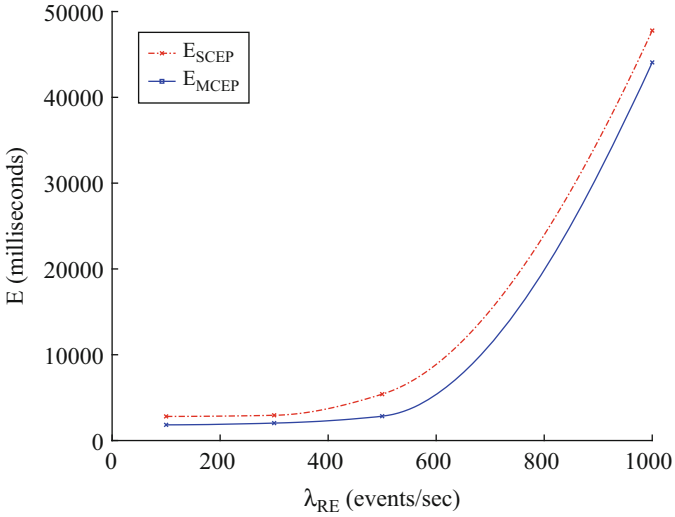

Fig. 9 Impact of the arrival rate of raw events on average end-to-end latency

sensors streams to the IoT server results in larger transmission delays that seem to lead to a higher E for the SCEP system. From Fig. 9, we can conclude that, in spite of using the large time window of 10 s (default time window) that leads to additional queuing delays on the memory constrained mobile device, E_{MCEP} achieved on the MCEP system with a given λ_{RE} is less than E_{SCEP} achieved on the SCEP system.

7.4 Comparison of IoT Server CPU Utilization

Figure 10 shows the impact of λ_{RE} on the CPU utilization of the IoT server in for the MCEP system ($CU_{IHS-MCEP}$) and SCEP system ($CU_{IHS-SCEP}$). For a given λ_{RE}, $CU_{IHS-MCEP}$ is lower than $CU_{IHS-SCEP}$. This is because of the difference in the amount of computation performed by the CPUs. In case of the SCEP system, all the raw sensor data streams are received, parsed, type converted, enqueued, dequeued and processed in the IoT server and then complex events are forwarded to the timekeeper by using the MQTT broker and metrics are sent to the DataDog dashboard by the Java Management eXtensions (JMX) agent. However, in case of the MCEP system, only CEP alerts are received by the IoT server and no further processing has to be done. The lower processing performed in case of the MCEP system leads to a lower CPU utilization. From this graph, we can conclude that the MCEP system leads to a smaller load on the IoT server, which is one of the advantages of the MCEP system.

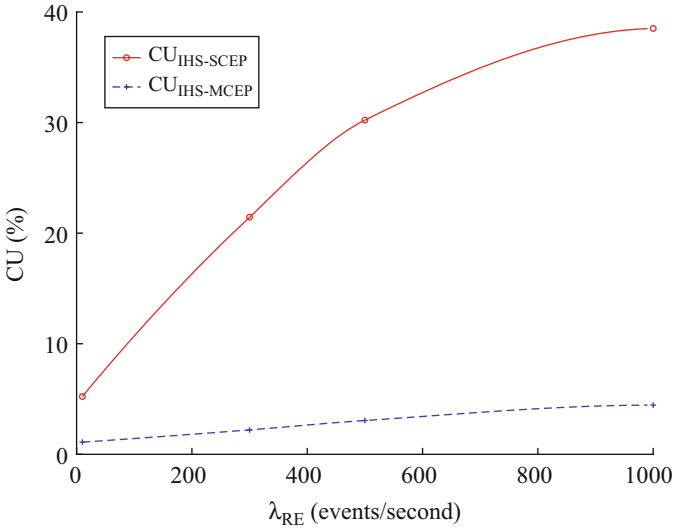

Fig. 10 Impact of the arrival rate of raw events on the IHS CPU utilization

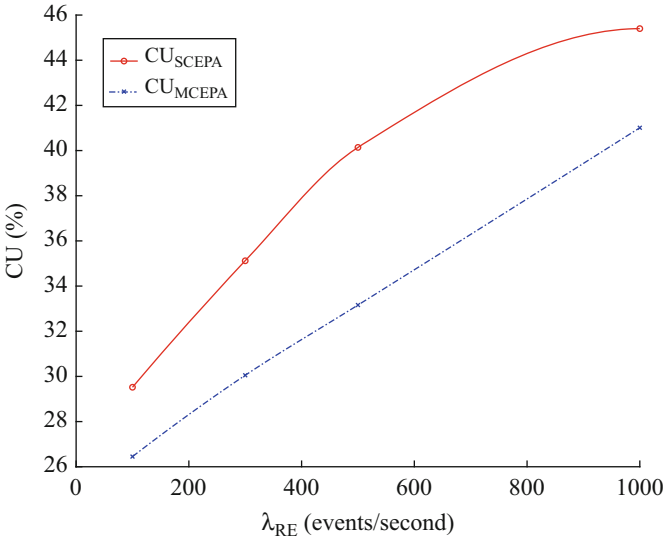

Fig. 11 Impact of the arrival rate of raw events on mobile device CPU utilization

7.5 Comparison of Mobile Device CPU Utilization

Figure 11 shows the CPU utilization observed for the MCEP application and the SCEP gateway application.

For the MCEP application, CU_{MCEPA} seems to increase steadily with the increase of λ_{RE} as more processing is done inside the CEP engine for the higher raw event arrival rates. Also, it is interesting to note that for any given value of λ_{RE}, the CPU utilization of the MCEP application is lower than the one for the server CEP application. For example, the CU_{MCEPA} is 41.01% when 2 sensors are sending data streams at 1000 Hz, which is lower in comparison to the 45.40% CPU utilization reported in the case of the SCEP application.

7.6 Comparison of User Cost

As shown in Fig. 12, for any value of raw event arrival rate, the amount of data transferred per second (TX) is more for the server CEP in comparison to the mobile CEP. This is because the SCEP application forwards all the raw events to IoT server. Equation (3) (discussed in Sect. 6.3) is used to compute the data transfer cost incurred by the user for using the MCEP and SCEP systems. The rate of $0.05/MB offered by Bell (a major telecommunication company in Canada) is used [4]. For any given λ_{RE}, a significantly lower data transfer cost is observed for the MCEP system in comparison to the SCEP system as in case of MCEP system only the complex events are sent while in case of the SCEP application the entire raw event streams are forwarded. It is interesting to note that at an arrival rate of 1000 events/second, the MCEP system provides a significant savings of $12.74/h ($13.32/h–$0.58/h).

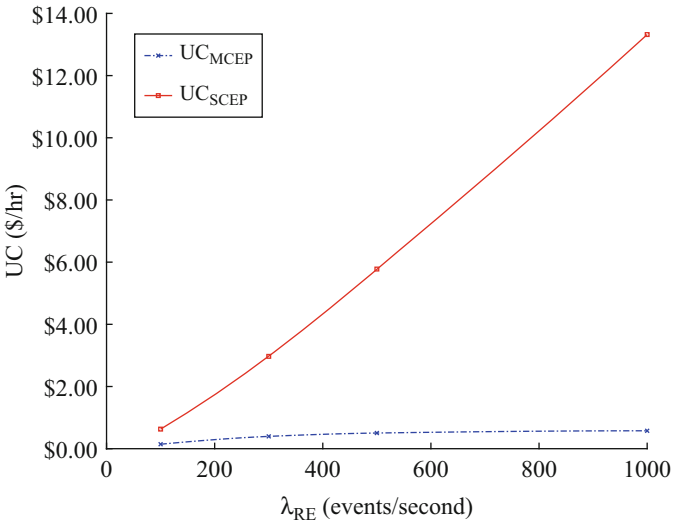

Fig. 12 Impact of the arrival rate of raw events on user cost

8 Conclusions

The availability of mobile devices and sensors at a reasonable price is rapidly increasing the use of mobile edge computing systems that are deployed in various applications that include smart homes, smart industrial machinery and smart healthcare systems. This chapter presents an edge computing framework and complex event processing technique for such systems. It includes a description of an application of these to real-time remote health monitoring that leverages both mobile system technology and edge computing techniques for the detection of complex events which typically indicate the potential occurrence of a health problem for the person being monitored.

One of the goals of this chapter was to compare two different architectural frameworks for performing complex event processing in sensor-based systems: SCEP (centralized server-based approach) and MCEP (edge device-based approach). Unlike the high-level simulation-based SCEP approach provided in [19], we have described an SCEP architecture and implementation of its prototype (in Sect. 3) that also has more features. However, such an SCEP system has some disadvantages including the necessity of a persistent network connectivity, high data transfer cost for the user, and a larger mobile device power consumption as shown in Sect. 7.1. On the other hand, the MCEP system can effectively handle the network unavailability problem by performing CEP on the edge device instead of processing the sensor data streams on a remote cloud. This system has been realized by successfully embedding a CEP engine on the mobile device to perform the complete complex event detection on the edge device and send various complex events (alerts) to a remote back-end server to notify the concerned personnel. The proof of concept prototype for the proposed technique has been built successfully and tested using a synthetic workload on a Google Pixel mobile device running *Android Nougat*. As discussed in Sect. 7.6, the MCEP system leads to a reduction in the user cost and the mobile device energy consumption and improves the overall latency of the system. A thorough experimental investigation based on measurements made on the prototype has led to a number of insights into the impact of system and workload parameters on performance. The key insights are summarized.

- Network connectivity requirement: The MCEP system does not mandate a persistent Internet connection with the back-end IoT server. Thus, if the network is not available temporarily, the user can still receive local alarms generated by the mobile device.
- User cost: As shown in Fig. 12, the user cost for the MCEP system is significantly lower compared to the cost of the centralized server CEP system. For the typical pricing data available at [4], the MCEP system provides savings of approximately $13/h, over the central server-based SCEP system. This is because the data transfer is reduced in the MCEP system, as only complex events are sent to the IoT server.
- Security and data privacy: As the mobile CEP system processes the sensor data streams locally, the user has better data privacy in comparison to the SCEP

system. In order to ensure the data privacy and security for the SCEP system, various authentication and authorization methods have to be employed on the IoT server which can lead to additional delays. Ensuring data privacy and security of a centralized server comes at the expense of processing latency. Thus, the MCEP system has an advantage over the SCEP system as it requires relatively lesser security mechanisms to be imposed on the system for ensuring data privacy.

- Out-of-order message delivery: As the SCEP gateway application forwards all the sensor data streams, this can lead to synchronization issues among various sensor streams at the back-end server. This issue is less evident in the MCEP system as the sensor devices are locally connected to the edge device using Wi-Fi or bluetooth connections.

These characteristics lead to the conclusion that the MCEP system has a significant number of benefits over the SCEP system. However, the SCEP system also has a few benefits over the MCEP system as described next.

1. Predictive analytics: In the MCEP system, only the complex events are sent to the IoT server. This means that the historical data of the patient is not retained. However, in the case of the SCEP system, the historical data can be further used by various predictive analytics algorithms using machine learning to predict future alerts.
2. Easier to deploy security mechanisms: The IoT server comes with off-the-shelf authentication and authorization features which are easily configured. However, in the case of mobile CEP, such features have to be manually added and customized.

9 Future Work

Directions for further research include the following:

- The MCEP system can be extended to form a hybrid CEP system such that real-time analytics is performed on the mobile device and the predictive analytics is being performed on the IoT server using the stored historical data. Investigation of such a system forms an important direction for future research.
- The performance of the current system can be analyzed when multiple devices (one device per user) are enrolled with the IoT server. This would test the scalability of the system as the number of users using sensor-based systems is expected to grow.
- MCEP leads to a lower battery usage in comparison to SCEP. Irrespective of the system type, a mobile device based remote health monitoring can be performed only for a number of hours after which the device needs to be recharged. This is acceptable for a number of different situations. Using multiple mobile devices with one serving as the primary device and the other(s) serving as backups may be helpful when the system is continuously used without an opportunity for recharging the battery of the mobile device. The secondary device can replace

the primary device when it runs out of battery power for a duration during which the primary device can get charged. The investigation of such a system focusing on how to perform an effective hand-off from one device to the other devices forms an interesting direction for future research.

Acknowledgments We are grateful to TELUS and Natural Sciences and Engineering Research Council of Canada (NSERC) for providing financial support for this research.

References

1. Abdellatif, A.A., Mohamed, A., Chiasserini, C.F., Tlili, M., Erbad, A.: Edge computing for smart health: Context-aware approaches, opportunities, and challenges. IEEE Network **33**(3), 196–203 (2019)
2. Apache Software Foundation: OpenWire Protocol. [Online available at]: http://activemq.apache.org/apollo/documentation/openwire-manual.html, [Accessed: 13-Jan-2019]
3. Bakker, A.: Comparing energy profilers for android. In: 21st Twente Student Conference on IT, vol. 21 (2014)
4. Bell Canada: Bell pay per use rates. [Online available at]: https://support.bell.ca/Mobility/Rate_plans_features/What-are-Bell-Mobilitys-current-pay-per-use-rates, [Accessed: 12-Jan-2019]
5. Buyya, R., Ranjan, R., Calheiros, R.N.: Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities. In: International Conference on High Performance Computing Simulation, pp. 1–11 (2009). https://doi.org/10.1109/HPCSIM.2009.5192685
6. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R.: CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. Journal of Software—Practice & Experience **41**(1), 23–50 (2011). https://doi.org/10.1002/spe.995
7. Cooking Hacks: MySignals changes the future of medical and eHealth applications. [Online available at]: http://www.my-signals.com/, [Accessed: 13-Jan-2019]
8. DataDog: Modern monitoring and analytics. [Online available at]: https://www.datadoghq.com/, [Accessed: 13-Jan-2019]
9. Dhillon, A.: An Edge Computing-based Complex Event Processing Technique for Sensor-based Systems. Master's thesis, Carleton University, Ottawa, ON, Canada (2018)
10. Dhillon, A., Majumdar, S., St-Hilaire, M., Haraki, A.E.: MCEP: a Mobile device based Complex Event Processing System for Remote Healthcare. In: the International Conference on Internet of Things (iThings), pp. 203–210 (2018)
11. Dineshkumar, P., SenthilKumar, R., Sujatha, K., Ponmagal, R.S., Rajavarman, V.N.: Big data analytics of IoT based Healthcare Monitoring System. In: 2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON), pp. 55–60 (2016). https://doi.org/10.1109/UPCON.2016.7894624
12. Foundation, A.S.: Apache thrift™. [Online available at]: https://thrift.apache.org/. [Accessed: 13-Jan-2019]
13. Gibson Research Corporation: Zeo Sleep Manager Pro. [Online available at]: https://www.grc.com/zeo.htm, [Accessed: 14-Jan-2019]
14. Goldberger, A.L., Amaral, L.A., Glass, L., Hausdorff, J.M., Ivanov, P.C., Mark, R.G., Mietus, J.E., Moody, G.B., Peng, C.K., Stanley, H.E.: PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. Circulation Journal **101**(23), e215–e220 (2000)

15. Habib, M.A., Mohktar, M.S., Kamaruzzaman, S.B., Lim, K.S., Pin, T.M., Ibrahim, F.: Smartphone-based solutions for fall detection and prevention: challenges and open issues. Sensors Journal **14**(4), 7181–7208 (2014)
16. Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K.: Java™ Message Service API Tutorial and Reference: Messaging for the J2EE™ Platform. Addison-Wesley Professional (2002)
17. Hartmann, M., Hashmi, U.S., Imran, A.: Edge computing in smart health care systems: Review, challenges, and research directions. Transactions on Emerging Telecommunications Technologies **n/a**(n/a), e3710. https://doi.org/10.1002/ett.3710
18. Higashino, W.A.: Complex Event Processing as a Service in Multi-Cloud Environments. Ph.D. thesis, Department of Electrical and Computer Engineering at University of Western Ontario (UWO) (2016). URL [Online available at]: https://ir.lib.uwo.ca/etd/4016
19. Higashino, W.A., Capretz, M.A.M., Bittencourt, L.F.: CEPaaS: Complex Event Processing as a Service. In: IEEE International Congress on Big Data (BigData Congress), pp. 169–176 (2017). https://doi.org/10.1109/BigDataCongress.2017.31
20. Hunkeler, U., Truong, H.L., Stanford-Clark, A.: MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks. In: 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08), pp. 791–798 (2008). https://doi.org/10.1109/COMSWA.2008.4554519
21. Kousen, K.: Gradle Recipes for Android: Master the New Build System for Android. "O'Reilly Media, Inc." (2016)
22. Naddeo, S., Verde, L., Forastiere, M., De Pietro, G., Sannino, G.: A Real-time m-Health Monitoring System: An Integrated Solution Combining the Use of Several Wearable Sensors and Mobile Devices. In: International Conference on Health Informatics (HEALTHINF), pp. 545–552 (2017)
23. Naqishbandi, T., Imthyaz Sheriff, C., Qazi, S.: Big data, CEP and IoT: redefining holistic healthcare information systems and analytics. International Conference on Advances Research in Engineering and Technology **4**(1), 1–6 (2015)
24. Organization, W.H.: WHO Global Report on Falls Prevention in Older Age. [Online available at]: http://www.who.int/ageing/publications/Falls_prevention7March.pdf, [Accessed: 8-March-2018]
25. Reza, T., Shoilee, S.B.A., Akhand, S.M., Khan, M.M.: Development of Android based Pulse Monitoring System. In: Second International Conference on Electrical, Computer and Communication Technologies (ICECCT), pp. 1–7 (2017). https://doi.org/10.1109/ICECCT.2017.8118045
26. Senthilkumar, R., Ponmagal, R., Sujatha, K.: Efficient Health Care Monitoring and Emergency Management System using IoT. International Journal of Control Theory and Applications **9**(4), 137–145 (2016)
27. Slee, M., Agarwal, A., Kwiatkowski, M.: Thrift: Scalable cross-language services implementation. Facebook White Paper Journal **5**(8) (2007)
28. Snyder, B., Bosnanac, D., Davies, R.: ActiveMQ in Action. Manning Publications (2011)
29. Sotera Wireless: About Visi Mobile. [Online available at]: https://www.soterawireless.com/visi-mobile/, [Accessed: 13-Jan-2019]
30. Suh, M., Chen, C., Woodbridge, J., Tu, M.K., Kim, J.I., Nahapetian, A., Evangelista, L.S., Sarrafzadeh, M.: A Remote Patient Monitoring System for Congestive Heart Failure. Journal of medical systems **35**(5), 1165–1179 (2011)
31. Wahane, V., Ingole, P.: An Android based wireless ECG Monitoring System for Cardiac Arrhythmia. In: Healthcare Innovation Point-Of-Care Technologies Conference (HI-POCT), pp. 183–187 (2016)
32. Wikipedia: Google Pixel. [Online available at]: https://en.wikipedia.org/wiki/Google_Pixel, [Accessed: 13-Jan-2019]