SEDA-SOA: A Scalable Event-Driven context-Aware Service Oriented Architecture

Majid Ayoubi
Dept. of Computer Science and Engg.
National Institute of Technology Warangal
Warangal, India
mayoubi@student.nitw.ac.in

Sadam Ravichandra, Member IEEE
Dept. of Computer Science and Engg.
National Institute of Technology Waragnal
Warangal, India
ravic@nitw.ac.in

***Abstract*— The increase in the number of Internet of Things (IoT) devices and their vast streaming data, results in data explosion. Context awareness has also become essential in software applications for better service delivery to users. It poses the urgent need for scalable integrated software systems which can analyze, store, extract meaningful events and to deliver information to the customers in real-time considering their context. A highly scalable architecture is proposed which uses distributed queues for decomposability and comprises of three functional components. 1. Distributed complex event integrated system where each of the complex event processing engines are taking care of event detection, 2. The context detection module which extracts contextual information and match them with detected events and, 3. A notification module and an application programming Interface is provided for the bulk alert information delivery. The presented model evaluation shows that our system well suits the current trend and requirements of IoT applications.**

Keywords*— Internet of things, SOA, Context Awareness, Event Detection*

I. INTRODUCTION

Context awareness is one of trending aspects of software engineering and Internet of Things (IoT). Nowadays, most of the technologies are made mobile in nature compared to the previously fixed devices. Home automation systems which are able to control the lights, temperature and even refrigerator are used to inform the user about the daily requirements for solving and easing human problems using IoT devices [1]. Intelligent tracking of mobile systems for searching a nearby friend and traffic conditions are best use of context awareness in software engineering.

The increasing number of IoT devices and messages due to variable nature of context and the IoT challenges and usability is increasing very fast [1]. McKinsey [2] expects that one trillion IoT devices will be deployed by 2025. Therefore, a scalable software architecture is required which can deal and handle humongous data that is referred to us as Big Data. Big data is the name for huge amount of heterogeneous data comes from different sources which are analyzed and stored for better automated decision making process [3]. Big data is processed in nearly real-time for effective use of the given services. As far as the real-timing need implies, new technologies have been created to process streaming data in real-time called Complex Event Processing (CEP) systems. According to Bessis N. et al. [3], one of the IoT expected challenge is the management of its big and heterogeneous data. Hence, there is an urgent need for highly scalable software architectures to address this challenge.

Keeping this challenge in mind, heterogeneous data is received from different sources which are pulled or pushed into the system are analyzed using CEP engines and to be supplied to the customers using SOA lightweight technologies such as Representational State Transfer (REST) services or Simple Mail Transfer Protocol (SMTP). Staged Event-Driven architectures which make use of separate queues or stages and multithreading for robust and scalable data handling are being used to improve the scalability of the event-driven architectures.

The main design approach of the work presented here is to provide an integrated software architecture to face the challenge of big data handling in real-time, considering the contextual information of the customers. SEDA-SOA: A Scalable Event-Driven context-Aware Service Oriented Architecture which is facilitated by Enterprise Service Bus (ESB) and Java Virtual Machine (JVM), providing the following functionalities: First, it receives a diverse data using push and pull techniques. Secondly, it fairly distributes the incoming data into the different queues to be processed by integrated CEP systems which each of them will provide data analysis and event detection in real-time. The results of CEP engines are sent to the different queuing levels for the context detection and Finally, the necessary context-aware notifications are provided to respective customers using the present SOA lightweight communication technologies.

The rest of the paper is organized as follows: Section II provides brief introduction about the background technologies. Section III goes through related work. Section IV explains the details of the proposed software architecture. In Section V, the flow structure and algorithm of the system is discussed. Section VI provides a case study. Evaluation is done in Section VII. Finally, the paper is concluded in Section VIII.

II. BACKGROUND

A. Internet of Things (IoT)

IoT is a trending concept which can be referred as combination of "anybody, anything, anytime, anyplace, any network and any service"[2]. IoT governs the coming generation technologies. It can impact the businesses and is defined as the "Interconnection of uniquely identifiable smart objects and devices within today's internet infrastructure with extended benefits" [2]. IoT architectures provide sensors, communication networks and some context processing which needs to be highly interoperable, reliable and scalable. The main parts of IoT are sensors, communication networks and handling of events. [2].

B. Event-Driven SOA:

SOA 2.0 or Event-Driven SOA as its name implies, used to react and handle the events in service execution, critical conditions, etc. [5]. The Dynamic nature of business processes for different adjustments of policies and partners make an urgent requirement for modeling, composing and running these processes in an architecture capable of automation and integration of different notifications, state management and monitoring of the services [5].

C. Staged Event-Driven Architecture

SEDA is supporting high level of concurrency and can ease the service delivery systems in service oriented architectures. The SEDA systems are well designed to separate the event-driven phases using queues. The applications and services are well placed in different stages so the resource will not be overcommitted when the data delivery surpasses the service capacity of the system. The resource controllers allow the system to handle high fluctuation of resource loads and send them to appropriate operating member to be handled [6].

D. Complex Event Processing (CEP)

Data is received from multiple sources to infer events or patterns that suggest more complicated circumstances [7]. Complex Event Processing (CEP) is an engine used for tracking and analyzing of different information for the events which are happening and to originate an outcome from it [7]. CEP combines data to identify the meaningful and relevant events and respond to them in the right time.

E. Context-Awareness

Dey et al [8] define the context as: "any information that can be used to characterize the situation of an entity", where "an entity can be a person, place, or object that is considered related to the interaction between a user and the system, containing the user and applications themselves". The adjective "context-aware" is used to refer the systems that are using context. Likewise, the individuality of the system can be reflected as "context-aware", "smart", "intelligent" or other alternative expressions to state the context-awareness [9].

III. Related Work

There are many approaches providing context information for services. It has surveyed by Kapitsaki et al. [10]. A trend is created for the researches in context retrieval from environment, service design and development, usability studies, etc.

SOCAM presented by Gu et al. [11] puts forward a Service Oriented Context-Aware Middleware That quickly prototypes the services which are context-aware in universal computing environments. This middleware notifies the attained contexts physical spaces into another space in which context-aware services pull and retrieve the contexts. SOCAM is used only for context detection and does not provide event processing and notification delivery.

Truong et al. in [12] presented a peer-to-peer framework that provides context information in emergency situation. They proposed a context information management service based on SOAP messaging. The work is not considered to be useful when it comes to different applications rather than collaboration works.

Wang et al. [13] proposed a scalable architecture called LIDAS. It makes use of CEP engines for event discovery. They provided mechanisms for easy and efficient event delivery and made use of distributed system integration that each system receives the data for event detection. The analysis and knowledge based component used to extract meaningful information and to provide them for users. Though they contributed a good work and system integration but the system model does not provide any contextual information handling, which is also essential for most of the IoT applications.

Chanda et al. [14] model can provide region specific context-based information from the addresses of service consumers. performs dynamic service choreography along with regional services. The Enterprise Service Bus (ESB) is enriched by the context providers and publishing service is particularized to have location context. The performance of the model is latent for streaming data processing and it does not provide Complex Event Processing and so, the model is not Event-Driven.

Here a special attention is given to the work done by Alfonso et al. [15]. They propose a novel SOA which makes use of Complex event processing and event-driven architectures while taking context of users into high consideration. Their work provides new era for integration systems and it is not application specific. They personalized their system for the case study of Air4People, which is used to analyze the air pollution data and contextual information of users. By using CEP and context management they provide meaningful notifications to customers.

The scalability of the system is to be highly considered for today's IoT applications [2]. Most of the systems provide good integration of the diverse components but few of them considered the scalability of the system.

None of the proposed works covers all the trending IoT needs which is given in survey of Buyya et al [2] for the integrated environments.

IV. Proposed Software Architecture

The interconnectivity of different components in a distributed environment is an essential necessity. The scalability and failure recovery is to be carefully designed [2].

Moreover, with spread use of smartphones, the dynamic change of locations and mobility has become the primary part of IoT systems. Therefore, modern architectures should be adaptable to handle dynamic interactions within integrated entities [16].

By considering the aforementioned design requirements, in this work, it is explained that how to integrate different components, maintaining the context, interoperability, accessibility and scalability of the architecture. The proposed model entails the following components:

A. Push and Pull Module

Push and Pull module is external part of the architecture. This module is used to provide the data from sensors that is considered as Big data. The pull, obtaining data from different channels by sending requests to them. Using push, the system receives the messages of channels from which it has already subscribed to. The data is sent to distributed queues to get processed.

B. Subscribers Registration Module

This module provides an API for users to submit their personal and contextual information. The contextual information is stored into the system using heuristic methods to put the users of the similar contexts in the same collection.

It provides better performance for the context matching module as it needs to match the context with only a single member of the collection. It also provides bulk sending facility for the notification alerts.

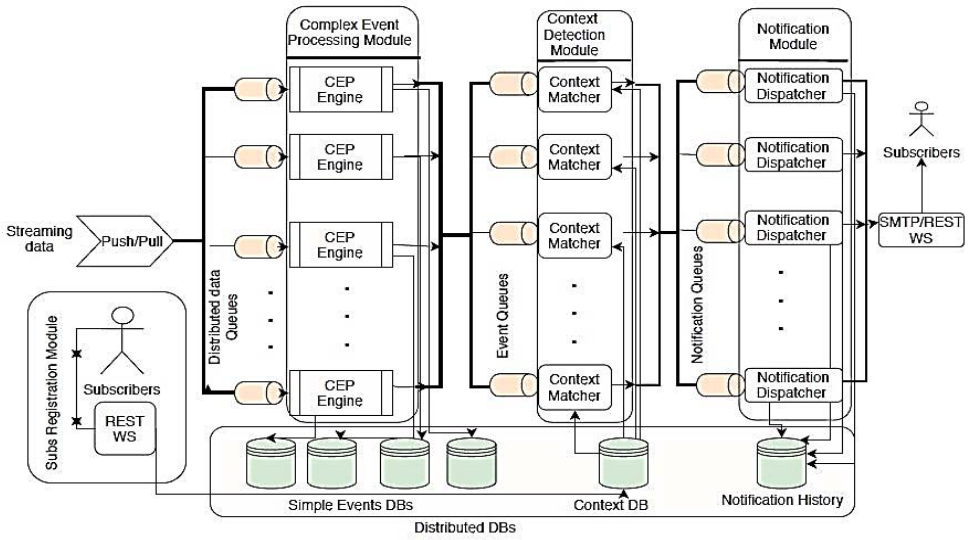

Fig. 1. SEDA-SOA Architecture

C. Complex Event Processing Module

CEP Engines take one of the functional responsibilities of the architecture. Data is received into these systems and certain events are detected using the specified patterns, and finally, the detected events are sent to event queues to be processed by Context detection module. CEP patterns should be lightweight and the pattern windows should be well designed to provide high scalability for the module.

D. Context Detection Module

This module fetches the data from event queues which have filled with results of complex events detection module. The context matchers are responsible to access context database and match the stored context information with the detected complex event contexts. This module matches the complex event context with a collection of same context provided by registration module. If matches, it sends the alert information with receiving addresses of the subscribers to the notification queues.

E. Distributed Database System

The distributed database plays a vital rule in the architecture. The integrated database management system is used to manage the storing process of the events, complex events, context of the users and related historical information for the data analysis.

F. Notification Module:

Notification module is used to receive the alert and destinations to be forwarded from the Context detection module filled up queues called notification queues. The required information is delivered to the users using lightweight web services. It is important to well design the notification module for specific case study. Distributed and separated notification module is needed especially when the number of users and alert information escalate.

G. Subscribers

Subscribers or users receive the alerts from the system. Users subscribe for alerts they want to be aware of. Users also save their location, physical and personal context. These data are used utilized by Context Matching Module.

This is worth to declare that the proposed model receives the location context of the sensors which are sending the data even if no subscriber has registered, when the situation is considered abnormal, the system sends alerts to the notification module.

V. Algorithm and Flow Structure of the System

The overall flow of the system is discussed in the Algorithm 1. The input data comes from sensors in different formats, and times. The output of the system is user alerts come from complex events detected in "CEP Engine" and context detection module.

ALGORITHM 1: SEDA-SOA Flow Structure

Input: Sensor data
Output: alerts
(1) Load _patterns ()
(2) Load_contexts ()
(3) ***While*** data=LISTENER.data! =null, ***do***
(4) ***Scatter*** data to DATA_QUEUES
(5) ***For each*** CEP_Engine ***do***
(6) ***Store*****(**data**)**
(7) ***If***(event=match_pattern(data) ==TRUE ***do***
(8) ***Send*** **(**event, EVENT_QUEUES)
(9) ***While*** event=EVENT_QUEUES! =null, ***do***
(10) ***For each*** Event_Matcher do
(11) ***If*** **(**alert_info=match_context (events) ==TRUE ***do***
(12) ***Send*** (alert_info, NOTIFICATION_QUEUES**)**
(13) ***While*** alerts=NOTIFICATION_QUEUES! =null, ***do***
(14) ***For each*** alerts ***do***
(15) ***Send*** (alerts, SUBSCRIBERS)
(16) ***Store*****(**alerts**)**

(Line 1) Initially, the defined patterns need to be loaded into CEP Engines, so the CEP engine start use of them. These patterns are distributed into different engines according to the specific case study. (Line 2) Next, the contextual information of users or subscribers has to be loaded. The data is required when the system reaches to the line 9. (Line 3) A loop is set which is working until the listener is not empty. (Line 4) Scatter is used to distribute and supply the data to different data queues provided for each CEP engine. (Line 5) For each CEP engine in the system, it is going from line 6-8: (Line 6) The data is stored in distributed database. (Line 7) In this part the match-pattern () function is called to match the received

data with deployed pattern. If the data is specified to match a pattern in CEP engine, it will be sent to the EVENT_QUEUES used for further processing (Line 8). (Line 9) While the complex events exist on EVENT_QUEUES do from line 10-12. (Line 10) For each Event Matcher system, it fetches the context information of users and (Line 11) if a complex event context matches with the context information of the subscribed users, it then (Line 12) sends the alert information and receiving addresses of the users' collection to the NOTIFIICATION_QUEUES. (Line 13) When the NOTIFICATION_QUEUES containing alert and user information for delivery of the alert or service do (Line 14) for each alert in queue, send them to the specified subscribers (Line 15). (Line 16) Finally, the alert information is stored for data analysis and submission status of alerts.

VI. CASE STUDY

For better representation of the proposed architecture, the case study which has used by Alfonso et al. [15] being utilized here.

The case study is related to health care systems which are of great significance in IoT applications [2]. The attention towards air quality has increased as it is essential for citizens. Day by day, everyone is worried about increasing health effects of air pollution. Air pollution is affecting the health of citizens, especially when they have certain illnesses such as asthma or heart diseases [17]. The Time Magazine post on October, 17, 2017 [18] clearly shows that 16% of overall human deaths are directly related to air pollution. The research exposes that 92% of deaths happen in low to middle income countries such as India and China. It means that one out four deaths related to air pollution. In India alone in 2015, 2.5 million deaths caused by air pollution, researchers pointed.

Different IoT systems for air quality monitoring have been created due to fact that air pollution is a worldwide concern. The monitoring is essential but not sufficient, the best use of monitored data such as real-time notification systems can have strong effect on health of citizens. SAFAR [19] a ministry of Earth and Science, government of India's project is created to monitor the air pollution in different cities such as Ahmedabad, Mumbai etc. The project is to analyze and observe air quality but not providing any daily alert information for the citizens. By use of different air pollution risk prevention methods described in [17] [20] we can create awareness for the users to act upon it; especially for the risk groups.

The effect of pollution alert program is well surveyed by Chen et al. in [21], which indicates that asthma related emergency admission has reduced by 25 % as one of the benefits of the air pollution alert program.

This case study well suits the different parts of the proposed architecture. Our architecture is not application specific and can be particularized to solve diverse IoT application problems. In this case we particularized the system to handle air pollution case study named Air4People by [15].

A. *The Functional Requirements of Air4people*

The users' requirements should be considered while designing the air quality alert system. The systems have to fulfill the following requirements according to [15]: (1) The air quality alert should be provided to users in real-time (2) Notifications should be provided in a way that the users can easily understand the risk and further actions (3) the risk group should be explicitly mentioned.

In order to make users understand the alerts being given to them, it is needed to use standard air quality measurement. Here we use the US Air Quality Index (AQI) [20]. It consists of values from 0-500 which fits into 6 levels as follows:

TABLE I: Air Quality index levels and respected colors

No.	Air Quality Index (AQI)	Levels of Health Concern	Colors
1	0 to 50	Good	Green
2	51 to 100	Moderate	Yellow
3	101 to 150	Unhealthy for Sensitive Groups	Orange
4	151 to 200	Unhealthy	Red
5	201 to 300	Very Unhealthy	Purple
6	301 to 500	Hazardous	Maroon

AQI categorizes each level with specific health concern and correspondent action to protect the health. General AQI alerts are not sufficient as they do not provide the details for each pollutant which is highly useful for specific customers. For example, we can provide the health concern and correspondent action for specific level of Ozone (O3) or CO2 for customers having diseases related directly to specific gas or pollutant [22].

B. *Context Definition for Air4people*

For the context, here we need to set some characteristics which defines it. Here, context is referred to be put in 3 categories with their specific characteristics as follows: 1. Location context, 2. Physical context and 3. Personal context.

1) Location Context

Each user saves his/her location and specific places s/he lives and exchange between them throughout the day.

2) Physical Context

This refers to the physical activities each user is doing. It can be a fix timetable which provides an hourly schedule of activities such as going for jogging etc. We provided whether a person is heavy-outdoor worker or not as physical context matching for this case study.

3) Personal Context

In this type of context, we consider the diseases and age of the users. There are two diseases to be considered, Lung diseases and heart diseases. According to the age of users we have two types: 1. Children, we consider anyone under 19 years as child. 2. Mature people, from 19 years and above.

C. *Particularization of SEDA-SOA for Air4People case study*

1. Queuing system

The backbone of the architecture is the queuing system. Three different stages of queues being used to decompose and provide high scalability for the system.

a. Queues for the data which is received by the system from sensors, storing different pollution measurements with the specific locations provided.

b. Queues for the intermediate results of complex event processing engine which used to detect events and send them to appropriate pollution level queues for context detection.

c. Queues to hold the results of the context matching phase for each notifiable event and receiving addresses of the users to be sent. The notification module uses the alert and receiving addresses of users for bulk sending.

2. Databases

The database is well particularized to support huge number of concurrent queries which are coming from architecture for different proposes. In order to speed up the extraction of users' contexts, each location has got a separate database containing users' different context properties. The corresponding alert databases has created for different levels of each pollutant. And for each kind of pollutants, a separate database is created to support concurrent database writing/reading functions.

3. Complex Event Patterns

The sensor data which is coming from the air pollution stations are called simple events. These simple events are matched with deployed patterns in CEP engine. The patterns have designed according to guideline provided in [20] and the structure of patterns being verified by Esper Online EPL.

For each level of the different pollutants (Ozone, Particulate Matter 10, Particulate Matter 2.5, Sulfur Dioxide, Nitrogen Dioxide and Carbon Monoxide) we have defined a pattern except for level 1 which needs no notification, so there is no need to design a complex events pattern for it.

The patterns have been designed to have less elements, so the extra elements which are not needed at the time of event detection are discarded. The code listing bellow shows the event pattern for Particulate Matter 2.5 of level 3:

CODE LISTING 1**.** Complex event pattern for PM2.5 level 3

(1) "***select*** 3 ***as*** levelNumber,"
(2) +"'PM2_5' ***as*** levelIndicator,"
(3) + "a1._pm2_5 ***as*** pm2_5,"
(4) + "a1._location ***as*** location "
(5) + "***from pattern*** [(***every*** a1= "
(6) + "esper.AirMeasurment ((a1._pm2_5>=35.5 ***and*** a1._pm2_5 <=55.4)))].***win:time(10 minutes)***"

(Line 1) If the pattern in line 6 has detected, so we specify the level of pollutant as 3, line 2 sets the level pollution indicator as PM2.5 to be sent to the queues, line 3-4 sets the event pollutant and location for event. Finally, at line 6 it specifies that in sliding window time of 10 minutes if it receives a pollution which is between specified range for each air measurement, so send the event to the event listener for further processing.

4. User Deployment

The user subscription is well designed to put each user in location and pollutant specific databases and collections. An API provided to send messages via a deployment interface. The data is sent to a java file which is responsible to check the users' context information and send them to the appropriate collection of a database. According to different contexts of a user, it is possible for a single user to be sent to many collections. Following pseudo code is provided for the users' collection allocation:

CODE LISTING 2. User deployment allocation:

(1) getDatabase(user.location)
(2) ***if*** (user.lungDisease=true or user.child=true or user.heavyOutdoor=true) ***do***
(3) ***send*** (user, O3)
(4) ***if*** (user.heartDisease=ture or user.lungDisease=true or user.child=true) ***do***
(5) ***send*** (user, PM10 and PM2.5)
(6) ***if*** (user.heartDisease=true) ***do***
(7) ***send*** (user, CO)
(8) ***if*** (user.lungDisease=true, user.child=true)
(9) ***send*** (user, NO2 and SO2)
(10) ***else send*** (user, Normal)

(Line 1) It selects the database according to user's location, (Line 3) if the condition in Line 2 matches, it sends the user information to the collection named O3. The rest of the code does the same. Finally, if no condition matched, the system sends the user to the normal category which will be notified with different alerts rather than risk groups.

It provides a great achievement in scalability of the system. The system needs to match the context once for each database and send bulk emails for all users of the specific category at once rather than finding and sending the alerts for the users one by one.

5. Email Notifications

The users of the system are being notified using SMTP bulk sending. The Alert queue was defined to have the comma separated list of emails and particular alerts to be sent. A local email server was deployed to receive the bulk emails with specified pollution alert. The system was also tested with Gmail SMTP service for sample email sending.

VII. EVALUATION

The dataset of air pollution of northern Taiwan [23] is being used to push the data into the system. The system also tested with random data pulled from the ThingSpeak public cloud.

The architecture different characteristics have been evaluated for stress tests. The whole architecture was deployed in an Intel core i7 Desktop computer with 8 GB of RAM. The system is deployed with 150 users of different contexts. It has tested with 36 patterns which are 6 level for each of 6 pollutants defined by EPA (Ignoring the 1st level and defining a level as pollutant level above 6).

The stress tests for the system has evaluated by the push model which is used to fetch data from CSV file of air pollution measurements and send them to different queues provided for each pollutant. We have used different configuration to test how the system reacts in different situations. In all configurations the system was tested in a period of 15 minutes (enough time to check the RAM overload). The different configuration information is shown as follows:

TABLE II. Different configuration characteristics

Configuration No	DB	Patterns	Windows	Queues
Configuration 1	No	6	Sliding	All
Configuration 2	No	All	Sliding	All
Configuration 3	Yes	All	Sliding	All

In Configuration 1 we have checked the system throughput if there is less number of patterns (6) to be processed. In Configuration 2 the system has checked with all patterns and queues without storing the simple events. In the Configuration 3 the system was tested with all patterns and queues while storing the simple events in the database which is an extra burden rather than previous configurations. For stress test purpose, we enqueued as much as data the system could properly respond. The performance tests result for the above configurations are shown as below:

TABLE III. The number of Events submitted, detected and notified for each configuration

Configuration No	Stages of Events	Number of events
Configuration 1	Total Events Submitted	2520000
	Event per minute	168000
	Relevant events detected	1237275
Configuration 2	Total Events Submitted	9429378
	Event per minute	628625
	Relevant events detected	2568958
Configuration 3	Total Events Submitted	8001870
	Event per minute	533458
	Relevant events detected	2156062

In Configuration 1 the system has tested using a single pollutant's (Ozone) 6 patterns, which contained more number of events in dataset. The system could work properly up to 168000 events per minute for sliding window, at the speed in which Ozone queue's data was increasing and not getting processed by the system. Comparing with CARED-SOA by [15] our architecture can provide more than 10x improvement as the CARED-SOA has stopped working at 7272 detected complex events from 54000 submitted events per minute, but SEDA-SOA could detect up to 82485 detected complex events from 168000 submitted events per minute.

In Configuration 2, the system was tested with all patterns and queues without storing the simple events. The system was supporting up to 171263 detected complex events from 628625 events per minute (36 patterns). The CARED-SOA has already stopped working properly at 11631 detected complex events from 26607 events submitted (47 patterns). In order to test the effect of batch windows on the architecture we defined 15-minute batch windows which is equivalent to have more than 1-million events in windows before sending actual complex events to different levels. Although, the system RAM could still support the batch windows, but the testing was trivial due to less number of complex events detected. Finally, the system was checked with all patterns and queues with storing the simple events into NoSQL (MongoDB) database. Surprisingly, the system could support huge number of event detection, storage and notification. SEDA-SOA could support up to 143737 detected complex events from 533000 submitted events per minute that is extremely more than CARED-SOA which was not responding properly at just 11523 detected complex events from 13700 events submitted per minute.

A. Model Discussion

The proposed model provides following enhancements compared to existent architectures:

- The powerful building block of the system is different queues which is providing high scalability.
- Each pollutant is processed simultaneously from different queues and the result is distributed to different levels and so, alerts.
- The use of multithreading makes the system enable to process next event, even if the previous event is under process and makes best use of resources of computer.
- Once the data is received into the CEP engine it is initially matched with patterns before context matching. If the atom event is created, then it further sends it to the queue to match with context.

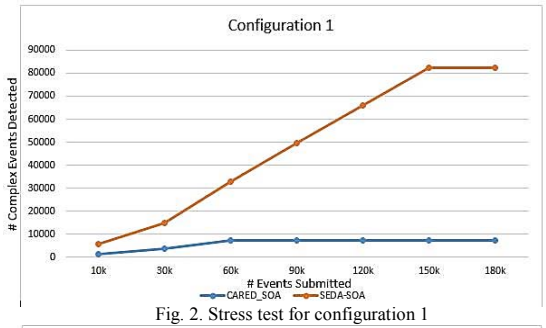

Fig. 2. Stress test for configuration 1

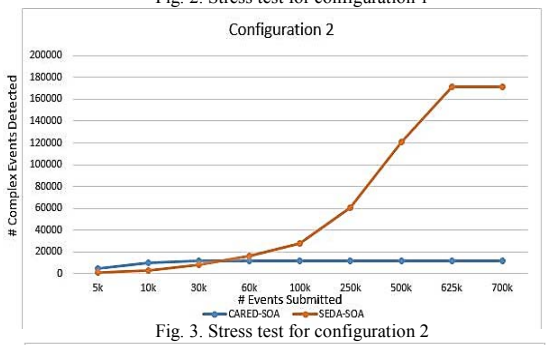

Fig. 3. Stress test for configuration 2

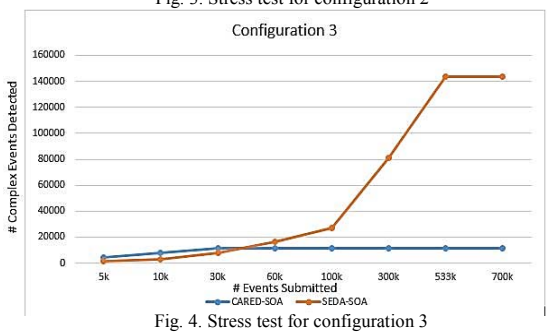

Fig. 4. Stress test for configuration 3

- The data is stored to and retrieved from distributed databases, so the processing system is not postponed to wait for the data retrieval and storage.
- Finally, the notification module is supported by sufficient queues and lightweight communication software to send the bulk email alerts to the right user in the right time.

SEDA-SOA uses lightweight java implementation rather than Enterprise Service Bus which has complex architecture. Though ESB provides enough facilities for program deployment, but it lacks to provide high scalability. We have used ESB in two parts, 1. For user deployment and 2. For notification alerts, as ESB could give better alert delivery for the bulk users rather than simple java implementation.

VIII. CONCLUSION

In this paper we proposed a highly scalable context-aware event-driven SOA which can satisfy the current IoT applications requirements stated in [2]. The data is being received either by Push or pull module, dividing the input into different CEP engines to be processed simultaneously and the results are sent to the context detection module. The resulting complex events are notified through the notification module by lightweight API to the end users considering their contexts. The Evaluation and model discussion illustrates that the system can provide highly scalable event detection and service delivery and can accomplish the current trend of IoT and integrated architectures requirements.

REFERENCES

[1] Gubbi J, Buyya R, Marusic S, Palaniswami M. . Internet of Things (IoT): a vision, architectural elements, and future directions. Fut Gen Comput Syst 2013;29(7):1645–60.

[2] R. Buyya, A. V. Dastjerdi, Internet of Things: Principles and Paradigms, San Mateo, CA, USA: Morgan Kaufmann, 2016.

[3] Bessis N, Dobre C, editors. Big data and Internet of Things: a roadmap for smart environments, vol. 546. Cham: Springer International Publishing; 2014.

[4] Guinard D, Trifa V, Mattern F, Wilde E. From the internet of things to the web of things: resource-oriented architecture and best practices. Architecting the Internet of Things. Berlin Heidelberg: Springer; 2011. pp. 97–129.

[5] Zakir Laliwala, Sanjay Chaudhary, "Event-Driven Service Oriented architecture", IEEE, Conference, Software Engineering 2008.

[6] Matt Welsh, David Culler, Eric Brewer, "Staged Event-Driven Architecture" Harvard University, 2013.

[7] Schmerken Ivy, "Deciphering the Myths Around Complex Event Processing", Wall Street & Technology, May, 2008.

[8] A.K. Dey, "Understanding and Using Context", Pers. Ubiquitous Comput., vol. 5, no. 1, pp. 4-7, Jan. 2001.

[9] Unai Alegre et al, Engineering context-aware systems and applications: A survey, The journal of system and software, Feb 2016.

[10] G. M. Kapitsaki, G. N. Prezerakos, N. D. Tselikas, I. S. Venieris, Context-aware service engineering: A survey, J. Syst. Softw., vol. 82, no. 8, pp. 1285_1297, 2009.

[11] Gu T., Pung H.K., Zhang D.Q., 2004. A middleware for building context-aware mobile services. Vehicular Technology Conference 5, 2656–2660.

[12] H. Truong, L. Juszczyk, A. Manzoor, S. Dustdar, Escape_An adaptive framework for managing and providing context information in emergency situations, 2nd Eur. Conf., (EuroSSC), Kendal, U.K., 2007, pp. 207_222.

[13] Dong Wang et al. A Novel Complex Event Processing Engine for Intelligent Data Analysis in Integrated Information Systems, International journal on sensor networks, 2016.

[14] J. Chanda, S. Sengupta, A. Kanjilal, and K. India, ``CA-ESB: Context-aware enterprise service bus," Int. J. Comput. Appl., vol. 30, no. 3, pp. 1 8, 2011.

[15] ALFONSO GARCÍA DE PRADO et al. CARED-SOA: A Context-Aware Event-Driven Service-Oriented Architecture, IEEE Access, 2017.

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

[17] Review of evidence on health aspects of air pollution REVIHAAP project, World Health Organization, Copenhagen, Denmark, Tech. Rep., 2013. [Online]. Available: http://www.euro.who.int/__data/assets/Pdf_file/0004/193108/REVIHAAP-Final-technical-report.pdf

[18] ALEXANDRA SIFFERLIN Here's How Many People Die from Pollution Around the World, The Time magazine, 2017. [Online]. Available: http://time.com/4989641/water-air-pollution-deaths/

[19] Ministry of earth and science, Govt. of India, fetched: March, 2018 [Online] Available: http://safar.tropmet.res.in/MONITORING%20SYSTEM-10-3-Details

[20] U.S. Environmental Protection Agency. (2014). AQI Air Quality Index. A Guide to Air Quality and Your Health, accessed on Jan. 17, 2017. [Online]. Available: https://www3.epa.gov/airnow/aqi_brochure_02_14.pdf

[21] Hong Chen et al. Effect of air quality alerts on human health, lancet planet earth 2018.

[22] S. Peter et al., Nutritional solutions to reduce risks of negative health impacts of air pollution, Nutrients, vol. 7, no. 12, pp. 10398_10416, Dec. 2015.

[23] Air Quality for Northern Taiwan, accessed: March, 10 2018. [Online] Available: https://www.kaggle.com/nelsonchu/air-quality-in-northern-taiwan/data