Adaptive Event Tree-Based Hybrid CEP Computational Model for Fog Computing Architecture

M.B.A. Prashan Madumal, D.A.S. Atukorale, T.M.H.A. Usoof

University of Colombo School of Computing

35, Reid Avenue, Colombo 07, Sri Lanka

prashan@compsoc.lk

aja@ucsc.cmb.ac.lk

hau@ucsc.cmb.ac.lk

***Abstract*— With the bloom of data generation devices at the network-edge, obtaining data intelligence in real-time posed problems due to network incompetence. In this paper we present a novel computation model based on Fog computing that take advantage of its low latency computing capabilities. A proof of concept is designed on top of the computation model as a layered system architecture that consists of sensors as data sources and complex event processing as the computation model in a Fog node. An event tree based scheduling system is proposed to mitigate the event dependencies. We also introduce a Fog to Cloud gateway that schedule data to the Fog or to the Cloud using a rule engine and system resource prediction. We evaluate the system against a traditional Cloud based computation model to validate its effectiveness and real-time computation capabilities.**

***Keywords*— Fog Computing, Complex Event Processing, Sensor Data Fusion, Fog to Cloud Gateway.**

I. INTRODUCTION

In recent years with the bloom of Internet of everything (IoE), we saw an exponential growth in devices that are connected to the Internet. According to Cisco [1], number of these devices will continue to grow and majority of them will belong to machine to machine devices (M2M) which will operate without human interaction.

But this advancement is not shared for networking technology which comparatively has a slower evolvement with respect to device's computation ability growth. This introduce the problem of moving data for computation, which is partially countered by the fog computing architecture as opposed to cloud computing architecture. Exponential growth in data which is about 2 Exabyte's per day gives a context of how the data is growing compared to networking architecture capabilities.

One novel way of achieving data intelligence and analytics is to use complex event processing (CEP), which is part of the Event Driven Architecture (EDA) [2]. CEP will filter, aggregate and match events into higher level events that can be consumed by applications. Main goal of CEP is to identify meaningful events and then respond and take decisions quickly as possible. By combining characteristics of fog computing and in-cooperating CEP at the edge of the network we can achieve the low latency and real-time responses that forth coming applications demand.

A. From Cloud to Fog

Fog computing can be explained as the paradigm that extends cloud computing and related services to the edge of the network [3]. Fog computing same as cloud computing will provide computation, data, storage and services for the users. Main characteristic that sets apart Fog computing from Cloud computing is the physical distance of the computation to the end user, its mobility and its wide distribution geographically. Services can ideally be hosted at the network edge or devices that are in a single level higher in the network. All the above facts greatly contribute in raising the quality of service (QoS), and reducing the response and reaction latency.

B. Fog networking

A Fog network can be considered as an underlying network architecture that can support a verity of emerging services and applications. While Fog theoretically extends the Cloud computing paradigm and even uses the Cloud technologies, Fog is distributed in a wider geographical area and much denser [4]. Also Fog devices will be much more heterogeneous, ranging from switches, routers, access points and edge end user devices.

C. Problem Definition

With the emergence of Fog Computing paradigm, need for a model that can cater the needs of exponentially growing connected devices became apparent. We aim to propose an architecture that follow a hybrid approach to compute high velocity data with the aid of CEP.

In Section II we go through the related work in areas of Fog computing and complex event processing. The Fog computation model that we propose is discussed in Section III and our proof of concept architecture is presented in Section IV and the implementation is discussed on Section V, We conduct a detailed evaluation and validation of our system in Section VI and conclude the paper in Section VII.

II. RELATED WORK

A. Fog Computing

Although a new paradigm, "Fog computing" can also be interpreted as the convergence of technologies that have been around for a while developing and maturing, most of the time independently of each other. Due to the rise of device ubiquity and demand for agile networks, Fog computing paradigm is expected to play a crucial part in service management and data privacy in the coming decade. Almost every layer of IT stack will undergo dramatic changes and shifts in regard to application development, network and

service provisioning and network traffic management with the emergence of Fog.

In the past decade, Enterprises gained many opportunities from Cloud computing with respect to computing services, thus became an efficient alternative to managing private computation sources. As Ivan and Shen [5] explains, in the coming decade due to the increase in number of connected devices, this bliss of advantages will become a critical problem. This hold true especially for latency sensitive applications. Fog computing is proposed to address the issues where Cloud computing paradigm fails to satisfy the requirements of these applications.

Cisco [6] envisioned the rise of the Fog to enable the applications of the coming era on billions of connected devices. The Cisco IOx [7] framework has been proposed as a competent architecture for the next generation IoT enabled connected devices and applications that run on top of them. By allowing to use bandwidth and storage capacity more efficiently IOx analyze and act on data right at the network edge. Data is sent to the Cloud only if the need arises. Because of having no viable way to transmit the Exabyte's of data generated through devices, proposed solution can be seen as a suitable implementation to solve the data movement to the Cloud.

B. Complex Event Processing

Complex event processing (CEP) model revolves around the key concept of viewing the events happening in the external world as notifications of flowing information. These events then go through a filtering process, which then have to be combined in order to understand what is happening, these combinations are termed as higher level events in the CEP model. Purpose of this model is to notify special patterns that occur in these high level events to interested parties, which is derived from lower level events. Many different communities have contributed to the formation of this model, particularly from wireless sensor networks, distributed information systems, business process automation, network monitoring, middleware systems and control systems. Eugester et al. [8] proposed novel approach that can be considered as one of the earliest CEP models. Traditional publish subscribe models consider each event separately without any relations between events, and based on their content or topic filter them to match the subscribers. In CEP, occurrences of multiple related events are identified as patterns in the event stream. This can be defined as an extension of the earlier mentioned publish subscribe models. Further most CEP systems can be categorized into two main groups, namely aggregation oriented, and detection oriented. Aggregation CEP systems calculate a result using the inbound events that comes to the system. Detection CEP systems focus on detecting a combination of events, which can match a pattern or a pattern template.

The first event processing engine can be traced back to the year 1997, introduced by Rosenblum et al. [9]. This model also supported public subscribe concept by filtering and extracting only the relevant notifications from the incoming stream. A subject system is defined as an already existing information system where event processing has been implemented. In subject systems an event is given as something that is happening or expect to happen [10].

According to M. Eckert and F. Bry [11] query languages can be used to manage complex events. They can be divided into three main categories namely, data stream query languages, composition operators and production rules. Events consist as tuples in data streams, where query languages that are similar to SQL are used for the manipulation. Tuples are then converted to database relations which in turn can be evaluated and then converted back to data streams.A. Paschke [12] explains about the designing patterns that should include in complexes event processing systems, emphasizing on categorizing CEP systems. Categorization is done according to the abstraction level, according to management level, according to the intended goal and lastly according to the good solutions and bad solutions. A novel architecture for wireless sensor networks was proposed by Pottie and Kaiser [13], which can provide Internet access and distributed networking for processors, controls and sensors that can be inside environments, equipment's and facilities highly ingrained. Applications that make use of these devices to relay on information has to be constrained on low power transceivers and low bit rates.

C. Sensor Data Fusion

Sensor data fusion has been a subject of interest for scholars in the past two decades, where the sensor development in many domains bloomed and flourished. The simplest way of defining the sensor data fusion is to describe it as the integration of data, which is generated from multiple sensors that has knowledge of their surroundings. As an effective way to minimize this vast data volume, data fusion can be used in multi-sensor environments to combine information from multiple sources and gain more insight to the problem that is not possible with just a single source. Information that can be gained from multiple sensor data fusion can greatly improve the perspective of the environment that in turn can provide a good foundation for decision making and control of the top tier system sensors belongs to [14].

Joint Directors of Laboratories (JDL) [15], which is one of the prominent organizations in data fusion describes data fusion as a multi-level and multi-faceted process that can handle the combination, correlation, association, detection and estimation automatically. R. Luo et al. [16] proposed a generalized model for data fusion that incorporate multi-sensor environments. Fusion centers are used to combine the data from multiple sensor data sources, this is done through a hierarchical process. Another data fusion model which is also a hierarchical architecture was presented by Harris et al. [17], which is still widely used in the technical community. Dubbed as the waterfall model, it mainly focus on the flow of data in a system where it can go to different level ranging from data level to decision making phase.

III. FOG COMPUTATIONAL MODEL

Fog computational model forms the basic skeletal framework of an architecture that take advantage of the Fog platform as well as the Cloud platform as computation resources. Figure 1 depicts the model, with the data source connected to the Fog node.

Fig. 1 Fog Computational Model

We model the solution around a single Fog node IoT device with multiple sensors connected which act as data sources. IoT device in turn is connected to a Cloud computation node through the proposed gateway. This forms the basis of a Fog network with only one Fog node attached. This model can be extended further to allow multiple Fog nodes, but we focus on the single Fog node structure.

A. Cloud-based Computation Model

Even though Fog computing platform possess superior response time, its computation power remains relatively low compared with other platforms like the Cloud. As this limitation should also be explored further, the selected model should be computationally intensive. Furthermore, as handling a large amount of data and processing them with low latency is a must, selected model should also be able to accommodate high throughput of data processing. Above mentioned traits can be found in computation models of image processing, video processing, simulation and event processing. All of these can be done at the edge of the network and will vastly benefit from the reduced latency of the Fog computing platform. Also the computation power required for the above remains in a high status. For our proof of concept design, we choose the event processing as the computation model. Comparatively complex event processing models are highly scalable and generalizable, which will aid our study when evaluating the Fog platform. Another main reason for choosing CEP as the computational model is its real time computational nature which converge with Fog computing features.

B. Hardware Platform

Fog computing platform can consist of different types of hardware devices, ranging from sensor motes to smart phones. Choosing the appropriate type of hardware platform can be challenging when the requirements of the computational model varies. As the suitable computation model was established as the CEP, Fog node type should be selected adhering to its requirements. The Fog node should not possess high end computation capabilities like a local desktop, nor should have inadequate computation power like in sensor motes. A serenity between computation capabilities and power usage should be maintained. Viable candidates that fulfil the above requirements would be smart phones, home theatre PC's (HTPC) and System on a Chip (SoC) computers. We chose SoC as the Fog node type as it offers substantial computation power with inherent network capabilities compared to the former 2 device types. Here we focus on IoT/IoA type devices thus a SoC platform was chosen to implement the solution.

C. Data Source

Data source of the model should be compliant with the computation model and the Fog node type. Numerous data sources exist that can use CEP as the computation model namely, social media feeds, stock market feeds, supply chain feeds etc. But the main criteria for the data source selection is its proximity to the computation model. Data source should originate at the network edge itself and also it should be able to generate large bulks of data continuously. Sensor data feeds classify into this category, as it satisfy the above said conditions. Thus for our model we use sensor data feeds as the primary data source.

Fog computation model lays the foundation for our proof of concept architecture, which extends the computation model into a layered design. As discussed in above sections, we chose the best models for the data source, computation model and the Fog node type to demonstrate the capabilities and weaknesses of the Fog computing platform. Even though our proposed architecture is built upon these selected models, one can freely choose other suitable models as a proof of concept architecture that can evaluate different aspects of the Fog computing platform.

IV. Proof of Concept Architecture

Three key areas will be highlighted in this section that ultimately build the system architecture, which is largely based on the waterfall data fusion model.

A. System Architecture

Whole system can be presented as a layered architecture that have independent functionality in each layer. This enables the high modularity and customizability in the implementation which will ensure pluggable nature in the architecture as we intended. End users will be able to develop components for these layers separately without worrying about other layers, which can be done according to the standards and constraints in the system. As depicted in Figure 2, proposed architecture consist of seven layers. A 7 layer design was used to clearly separate out the data flow in the system, where the 3 topmost layers will be merged in the actual implementation.

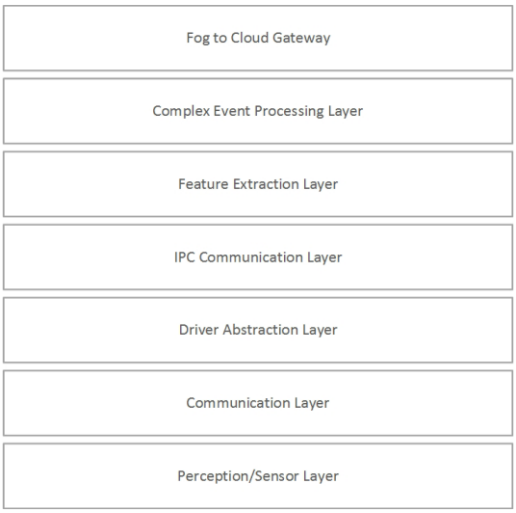

Fig. 2 Layered system architecture

Above architecture is divided into three main areas, the lowermost layers which are the perception layer, communication layer and driver abstraction layer belongs to the low level hardware component module. Task of the inter process communication (IPC) layer is to act as the messenger between lower layer and the above application layer, which can also be taken as a part of the application layer itself. Other sub layers of the application layer are the complex event processing layer and the feature extraction layer. Top most layer consist of the gateway module that act as the messenger between the Fog and the Cloud.

B. Fog to Cloud Gateway

Fog to cloud gateway module can be named as the reason for the uniqueness of the proposed architecture. There have been previous studies that incorporated complex event processing and sensor data fusion but none of them have a mechanism to take advantage of Fog computing platform as well as the Cloud computing platform. Presented Module in Figure 3 has four main components, having the Rule engine module as the primary component. Other three modules connect with the rule engine to carry out the scheduling of events to either the Cloud CEP engine or to the local CEP engine.

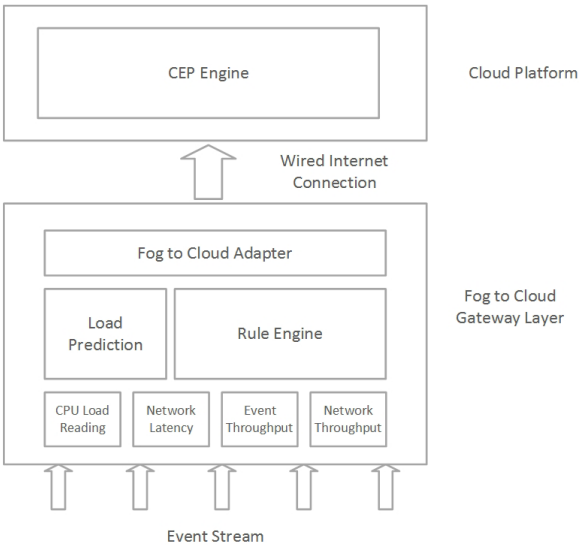

Fig. 3 Fog to Cloud Gateway Module

Resource Monitoring: This layer of the module consist of four resource monitoring components that will take readings of the system's following resources namely CPU load, network latency, network throughput and event throughput of the local CEP engine. These values will be read in short time intervals to take accurate system status to feed into the upper layer.

Rule Engine: Rule engine module will process the readings of the Resource monitoring module and based on rules that is applied to priority classes of the events, will make the decision of scheduling. Scheduling can either be to the Cloud based CEP engine or to the local CEP engine. After the decision is made for each event it will be passed into the upper layer or directly to the local CEP engine.

Load Prediction: Even though the Resource monitoring layer reads the system status, Rule engine can only schedule the events based on current or past resource values. This is a disadvantage as the system won't be able to take precautions and adjust ahead of time. Load prediction module will predict the system status based on history values of the resource monitoring data. This can enable the rule engine to schedule events more effectively as it can anticipate the future changes of the system resources and adapt to those changes.

Cloud Connector: Fog to Cloud adapter handles the communication between the architecture and the Cloud platform. Communication instances exist in the scenario of overload at the local system. Event stream will be transported to the Cloud platform through a WAN connection.

V. Architecture Implementation

Implemented system is divided to three independently functioning subsystems. From lower levels to the higher level, they are the sensor data reading system, Fog computing based CEP engine and Cloud computing based CEP engine. Because each subsystem is independent from one another it was possible to use different technologies and programming languages to develop each system. The only requirement was to keep the data communication protocol and format to remain consistent between subsystems. The lowermost subsystems were developed in a single hardware platform that has capabilities of supporting a high level OS as well as low level sensor communication hardware support.

A. Hardware Layer

Implementation of the hardware layer consists of the sensor drivers that is needed to interact with the sensors that take environmental readings. Sensors can be categorized into two classes on how they interact with the hardware platform, namely stand-alone sensors and sensors hubs. These 2 classes are connected to the hardware platform through different mediums. Hardware layer was implemented in the PcDuino SoC (System on a Chip) taking advantage of its Arduino-like header interface and its on-board Wi-Fi module.

B. Fog Based Complex Event Processing Engine

Fog based event processing engine is implemented on the PcDuino v2 board which has an ARM architecture based processor namely the ARM Cortex A8 running at 1GHz frequency. Other hardware specifications include 1GB DRAM, RJ42 Ethernet connection, on board Wi-Fi module and a 4GB ROM. Operating system is Ubuntu 12.04 with the LXDE desktop environment. CEP engine was developed using the Esper complex event processing engine as the core to process the queries. We chose Esper over other CEP engines like Siddhi because of its high event throughput. Java 8 JRE was used as the run-time environment for the engine and various Linux tools was used to get system status and resource usage of the system.

1) *Events:* Events are the basic building blocks of a CEP engine. Events can range from basic primary events that only have raw data to complex events that are generated from multiple events with rules applied to them. The data that is required for queries and rules are carried within events and can be extracted when necessary.

TABLE I
Event Priority Classes

Event Name	Event ID	Priority Class	Priority Value
Acceleration	ACCE	Primary	5
Gravity	GRAV	Primary	5
Rotation	ROTA	Primary	5
Orientation	ORIE	Primary	5
Luminous	LUMI	Primary	5
Temperature	TEMP	Primary	5
Humidity	HUMI	Primary	5
Distance	DIST	Primary	5
Critical Luminous	WLUM	Critical	6
Critical Temperature	WTEM	Critical	6
Critical Humidity	WHUM	Critical	6
Average Luminous	ALUM	Noncritical	3
Average Temperature	ATEM	Noncritical	3
Average Humidity	AHUM	Noncritical	3
Entered	ENTE	Normal	4

In our System event are categorized according to their priority level namely, primary events, critical events, noncritical events and normal events. Table I depicts the event names, Id's and their classes. We selected the events based on availability of sensors in the simulated environment and their ability to derive high level events and decisions.

2) *Event Hierarchy:* Every primary event in the system can lead to secondary, tertiary and higher level events forming an event hierarchy. To better understand the event hierarchy of our system, following example can be taken into consideration. A person enters the room, which triggers the "DIST" event and through that the "ENTE" event because the threshold of distance was met. At the same time the temperature of the room is at a critical level of 40 Celsius which makes the "WTEM" event to occur. Both of these "ENTE" and "WTEM" event will be the origin point of the "EWTE" event which will warn or take necessary actions to the situation.

C. Event Tree Based Event Dependency Resolution

When scheduling the events through the getaway to either the Fog CEP engine or to the Cloud CEP engine, we encountered the critical problem of tightly coupled dependencies between events. Our system's events are highly dependent on each other, in most cases one event is the origination point of another event. We introduced a novel method of solving dependencies of events, taking account the priority of the events and event hierarchy. At the first stage of our algorithm, event tress are formed by having the end complex event as the root and omitting the primary events. Figure 4 shows the resulting event trees with their respective priorities in each event.

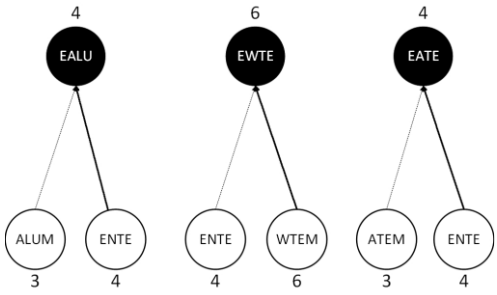

Fig. 4 Event Tree Structure

The highest priority value of the children will propagate to the root node of the tree. As seen in the 1st tree, priority value 4 of "ENTE" was propagated to the root node "EALU" as it was the max from the 2 children it has. Even though Figure 4 only depicts the event trees of our system, this algorithm can freely scale up to accommodate any number of trees with more children in each tree. Main goal of our introduced algorithm is to have all the dependent events process in the same instance of the CEP engine, either in the Fog based engine or in the Cloud based engine. For an example according to Figure 4, we want to process ALUM and ENTE in the same engine to get the resulting event EALU. But at the same time we want to process ENTE and WTEM in the same engine too, to the get the EWTE event. These 2 event trees can be scheduled in different engines, but because they both have the ENTE event as a common child we have to schedule both of the trees in the same CEP engine or else the last complex event won't be generated. In this example all 3 trees will be scheduled in the same engine as they have ENTE event as a common child.

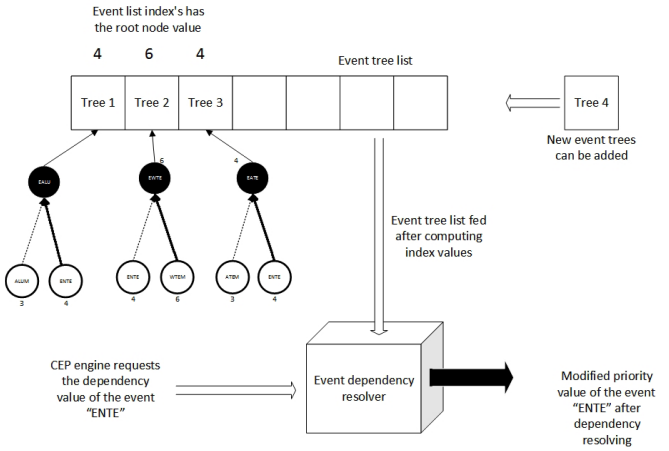

Fig. 5 Event Tree Algorithm Flow

As shown in Figure 5, Event trees are added to an event tree list. Event trees are implemented with hashmaps with children having entries in the map, and event tree list is implemented as a list of hashmaps. Event tree list has to be initialized and index values calculated before dependency solving. Index values will be the value of the root node of the tree which is the max event priority of the children of that tree. CEP engine can request dependency solving whenever an event scheduling is happening, and the output will be a modified priority value of the requested event. In our implementation that has 3 event trees, if the Engine requests the dependency for ALUM event output value will be 6 as the modified priority. ALUM is dependent on ENTE, but ENTE also dependent on WTEMP. This will cause the output value to be 6 as WTEM has priority 6 and all of the above events should be scheduled in one engine to make decisions accurately.

D. System Resource Prediction

Prediction model was used to predict the 4 system resources that was discussed in above sections, namely CPU usage, network throughput, network latency and event throughput. When implementing, we opted for a regression based prediction model as they offer robust performance. We also devised an algorithm to collect the system resource data at run-time and predict the relevant values for certain amount of seconds to the future.

E. Rule Engine

As the core of the Fog to Cloud gateway, we implemented the rule engine divided into 3 parts in accordance with priority classes of events. Data was fed as input for every part of the engine that comes from the System resource monitoring module, system resource prediction module and event dependency resolving module. Input to the rule engine will be the event ID of the event to be scheduled and its priority.

F. Cloud Based CEP Engine

Our implementation of Cloud based CEP engine is largely the same as Fog based CEP engine with few notable omissions. The whole Fog to Cloud gateway has been removed and a simple Cloud to Fog connector was added to

handle the communication between the Cloud and Fog. This includes the removal of System resource monitoring, resource prediction, rule engine event tree module. We choose Windows Azure IaaS cloud platform to host our Cloud based CEP engine. A VM with a virtualized Intel(R) Xeon(R) CPU E5-2673 v3 @ 2.40GHz processor with 4 cores available and with 7GB of RAM was used. Actual location of our VM is listed as Central US.

VI. Evaluation and Validation

Each subsystem and module will be subjected to a number of experimental evaluation procedures where results will be compared and evaluated. We carried out the evaluation experiments in the relevant hardware platforms they were implemented on.

A. Dataset

Our implemented system gather data from sensors in a highly dynamic environment, where sensors gather data from a room that resembles a busy classroom. Event simulator will collect the event data and their occurrence times since the system start, these event data will be the actual data that the sensors feed to the CEP engine. Then the simulator will input the replicated sensor data in the next run, with the correct arrival times and time differences between arrivals.

B. Analysis of Fog Based CEP Engine with Fog to Cloud Gateway

This section will focus on analyzing, evaluating and benchmarking our proposed novel architecture, in areas of performance, effectiveness and accuracy. Different datasets were used for some evaluations where measuring performance was of importance. Our introduced prediction module and event tree module will also be analyzed and evaluated in forthcoming subsections.

1) CEP Engine Benchmark: To accurately evaluate our Fog based CEP engine's capabilities with regard to performance, we did benchmarking using different platforms. As our system was implemented in specific hardware, the engine is handy-capped by its hardware and architecture limitations. We ran the benchmarks 3 runs in each platform to eliminate any anomalies. We chose one example from each platform to clearly highlight the differences between them. Obtained results are listed in the Table II below.

TABLE III
Benchmark Run times

Platform Name	First Run (ms)	Second Run (ms)	Third Run (ms)
Fog Node	342595	353469	359019
Cloud VM	8151	8060	8112
Local Desktop	2447	2537	2570

2) *Event Throughput and CPU Load Analysis:* Analysis will be done on Fog node platform as it is the most significant of the tested 3 because our system was implemented in the same hardware platform. Figure 6 shows how CPU load vary with the event throughput below.

CPU load can be seen to steadily rise into upper 80% and hover around that limit in the tested time duration of 355 seconds. Note that these results are also based on the amplified dataset we used for benchmarking, thus only represent the CEP engine in a stressed state. Furthermore event throughput seems to climb straight up to 700 events per second and oscillate around 700 in the course of the test run. We can observe a relation between the event throughput and CPU load where dips of event throughput result in lowered CPU loads.

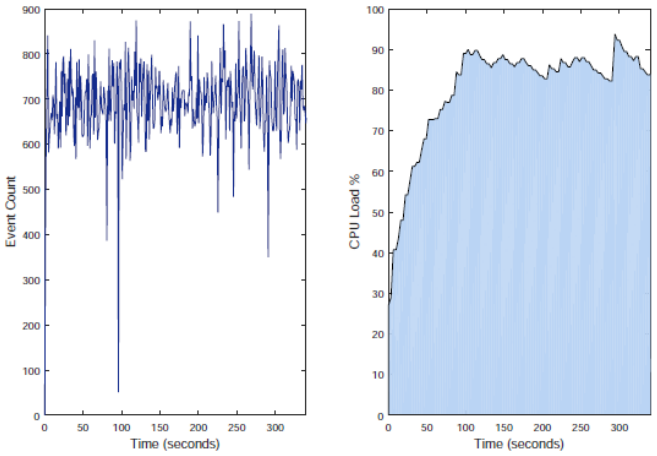

Fig. 6 Fog Node Event Throughput and CPU Load vs Time

3) *Event scheduling vs CPU Load Variance:* This phase was done by simulating the network latency artificially as we cannot use real values to experiment with the rule engine. CPU load values were set to be static having 95% as the value.

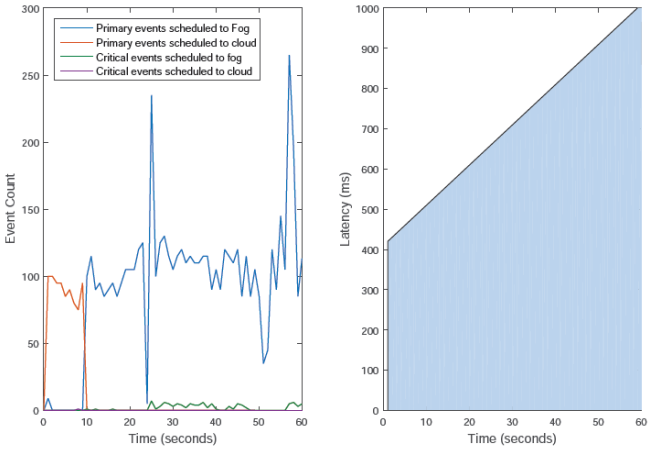

Fig. 7 Critical Class Event Scheduling vs Latency Variance

As other resources has no effect on the rules of critical class, we do not tamper with their values for this test run. Figure 7 graphically represent our observed values in event scheduling. Here we simulate latency variance discarding the effects of traffic priority types and loading conditions. Latency was incremented from 400 to 1000ms programmatically in the period of analysis. From the depicted graphs, primary events and critical events schedule to cloud until 10 seconds from system start time. At this point Latency has been increased from 400 to 500ms, which is the threshold value we have given for this test for the rule class algorithm. From then onwards all the given events can be seen to schedule into the Cloud.

4) *Unstimulated Event Scheduling Analysis against System Resources:* To evaluate the real world behaviour using actual resource values obtained from the platform at run-time we ran a test run with the original sensor data. In this section,

results obtained from the test run is analyzed and evaluated with the expected outcome of the architecture. Figure 8 represent all the system resources and the event scheduling happened in a time duration of 60 seconds from system start. In a real world scenario CPU load of our system hovers around 30% as seen in Figure 8, and according to event throughput spikes it can jump up to around 60%. In our test run we observed a network latency spike which can be attributed to due to sudden CPU load jump in the time interval of 30 to 40 seconds.

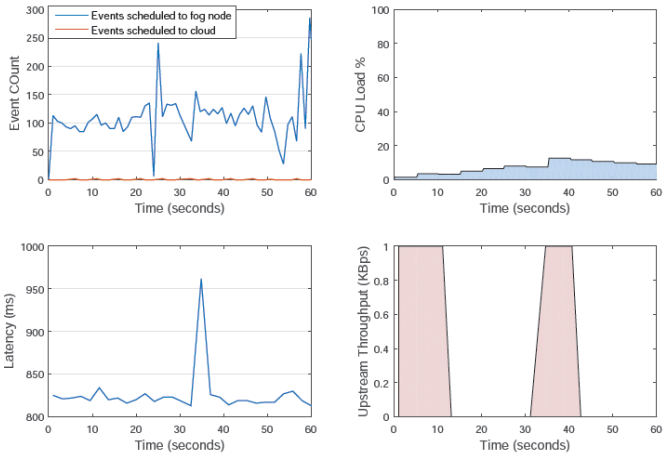

Fig. 8 Unstimulated Event Scheduling vs System Resource Variance

5) *Event Tree Module Performance Analysis:* Our novel implementation for event dependency solving using event trees will be analyzed in this section, focusing on the performance of our algorithm. The actual event trees from the original dataset was used, which consists of average luminous (ALUM), entered (ENTE), critical temperature (WTEM) and average temperature (ATEM) events.

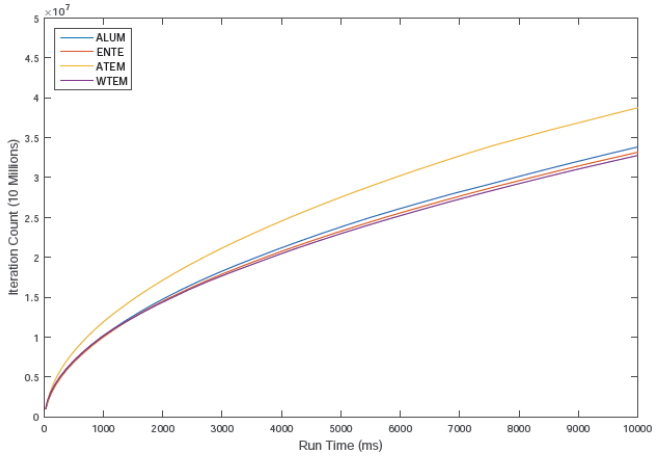

Fig. 9 Event Tree Module Performance Analysis

As depicted in Figure 9, algorithm was run for 1 million times per iteration, for 50 iterations and run time for each iteration was logged.

C. System Comparison and Analysis

An in-depth comparison of the CEP engine based only in Cloud and our implemented novel architecture of Fog based CEP engine with the Cloud gateway will be carried out in this section. The vital element that we focus here is the latency and the responsiveness of the system. As a real-time system, our implemented solution needs to be swift in handling the responses from events that occur.

1) *Primary Event Latency Comparison:* Original dataset was used as the event feed to the CEP engines in both Cloud and the Fog node. We carried out the experiment by analyzing the behaviour of luminous event taken as the measure point for primary events. Figure 10 graphically represent the event occurrence times and response differences in each engine. Here event value corresponds to the actual sensor data value of the particular event.

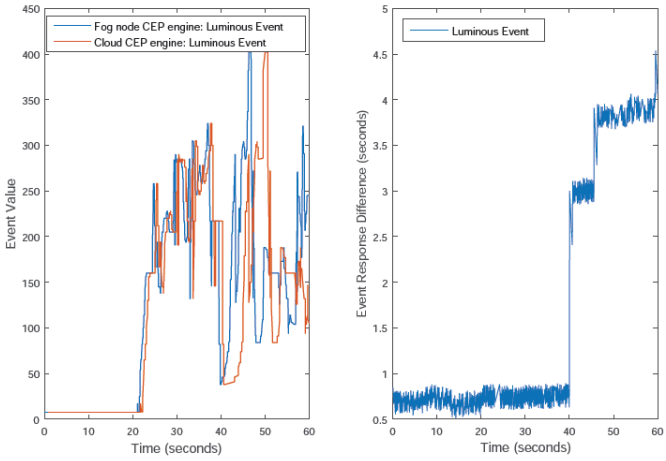

Fig. 10 Primary Event Latency Comparison

We can observe the event response difference hovering around 800ms range until 40 seconds after system start. But after that the response difference increase up to 4000ms. This can be attributed to the increased number of events transferred thus increasing the latency between the Cloud and the local event sender. In a real time system, even having 800ms as the response difference is unsuitable as the actual response time can be higher than that. Here we only measure the response difference, omitting the actual response time. Our engine correctly schedule primary events into the Fog node, when latency is high, thus our solution eliminate the undesirable high event response time present in the above graphs.

VII. CONCLUSION

By effectively exploiting the ever evolving computation power of edge devices, we can obtain data intelligence right at the edge of the network. Our work focused on developing a "pluggable" architecture that is based on the novel Fog computation model we introduced. This new model's uniqueness lies in its bridged computation link between the Fog node and the remote computation source, where each source can be used simultaneously to process data as needed. This key feature allowed our introduced architecture to make use best of both worlds for data processing.

We designed our layered architecture on top of this established Fog computational model, where each layer has independent functionality from one another. As the core computational model for the system, we choose complex event processing considering its real-time nature when processing data. Sensors that sample environmental changes were taken as data sources for the computation model. Our architecture was refined through iterative implementations that optimized the overall system. We presented a unique gateway module that request the aid of remote computation

sources by evaluation Fog node's resources and other factors like event throughput.

The designed layered architecture was completely implemented, where the Fog based CEP engine was hosted in the PcDuino V2 platform and the Cloud based CEP engine in Windows Azure IaaS Cloud platform. In our iterative implementation process we introduced a novel concept for dependency solving between events in the form of event trees. This concept proved crucial when scheduling the events to the corresponding computation source. A modified polynomial regression model was also presented that aided the prediction of dynamic resources of the Fog node.

We carried out a comprehensive evaluation and validation process, measuring the accuracy, performance and the effectiveness of the implemented system. Capabilities of the implemented CEP engine was explored through benchmarking in different platforms. We observed the effectiveness of the rule engine through our experimental evaluation methodology, which proved to be highly effective and performed as expected. Event tree module's performance exceeded expectations in the evaluation process, confirming its validity in a real time environment. In our experiments with the prediction model it proved to be inaccurate in certain situations of resource predictions where the usage of the said resource depend on a high range of variance. This heightened the reliability of a bare rule engine existence without a prediction module. Further approaches can be explored for prediction module to obtain higher accuracy in areas of machine learning and statistical approximation.

VIII. Future Work

Distributed Virtualized Fog Computation Model: In our study we presented our architecture regarding for Fog node as the computation source. As the Fog computing platform is a highly virtualized one, one can take advantage of this and extend the architecture to support multiple Fog nodes. Multiple CEP engine instances can be run on each Fog nodes available in the system. Different types of Fog nodes are connected through a single gateway with the Cloud. Here main focus should be given to how the events are scheduled in each local CEP engines that exist in the Fog nodes, and how different types of Fog nodes can be virtualized to better utilize their resources.

Acknowledgment

We sincerely thank WASN Lab of UCSC for providing the necessary development environment for our test bed.

References

[1] T. Cisco, Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update , 2010 2015, Growth Lakel., vol. 2011, pp. 20102015, 2011.

[2] David B. Robins. Complex Event Processing. 2010 Second International Workshop on Education Technology and Computer Science, page 10, 2010.

[3] M. Firdhous, O. Ghazali, and S. Hassan, Fog Computing : Will it be the Future of Cloud Computing?, The Third International Conference on Informatics & Applications, pp. 815, 2014.

[4] L. M. Vaquero, L. Rodero-merino, L. M. Vaquero, and L. Rodero-merino, Finding your Way in the Fog : Towards a Comprehensive Definition of Fog Computing Abstract : Finding your Way in the Fog : Towards a Comprehensive Definition of Fog Computing, 2014.

[5] I. Stojmenovic and S. Wen, The Fog Computing Paradigm: Scenarios and Security Issues, vol. 2, pp. 18, 2014.

[6] Fog computing, ecosystem, architecture and applications - research at cisco, 2015. URL http://www.cisco.com/web/about/ac50/ac207/crc new/ university/RFP/rfp13078.html.

[7] Roberto Mora. Cisco iox: Making fog real for iot, blogs@cisco -cisco blogs,June 2015.URL http://blogs.cisco.com/ioe/cisco-iox-making-fog-real-for-iot..

[8] P. T. Eugster, P. a. Felber, R. Guerraoui, and A.-M. Kermarrec, The many faces of publish/subscribe, ACM Comput. Surv., vol. 35, no. 2, pp. 114131, 2003.

[9] David S. Rosenblum and Alexander L. Wolf. A design framework for internetscale event observation and notication. SIGSOFT Softw. Eng. Notes, 22(6):344-360, November 1997.

[10] David Luckham and Roy Schulte. Event Processing Glossary - Version 1.1. Processing, (July):1-19, 2008.

[11] M. Eckert, S. Oriented, A. Soa, and E. A. Eda, Complex Event Processing (CEP) Types of Complex Event Processing Event Queries, Informatik-Spektrum, vol. 32, pp. 18, 2009.

[12] A. Paschke, Design Patterns for Complex Event Processing, October, 2008.

[13] G. J. Pottie and W. J. Kaiser, Wireless integrated network sensors (WINS): Principles and practice, Cacm00, pp. 110, 2000.

[14] G. Xing, R. Tan, B. Liu, J. Wang, X. Jia, and C.-W. Yi, Data fusion improves the coverage of wireless sensor networks, Proc. 15th Annu. Int. Conf. Mob. Comput. Netw. - MobiCom 09, p. 157, 2009.

[15] Alan N Steinberg, Christopher L Bowman, and Franklin E White. Revisions to the JDL data fusion model. Proceedings of SPIE, 3719:430-441, 1999.

[16] R. Luo, C. Yih, and K. Su, Multisensor fusion and integration ap-proaches, applications, and future research, IEEE Sensors J., vol. 2. pp. 107119, 2002.

[17] C.J. Harris, A. Bailey, and T.J. Dodd. Multi-sensor data fusion in defence and aerospace. The Aeronautical Journal, 102(1015):229-244, 1998.