2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems

A Secure Big Data Stream Analytics Framework for Disaster Management on the Cloud

Deepak Puthal*, Surya Nepal†, Rajiv Ranjan‡† and Jinjun Chen*
*Faculty of Engineering and IT
University of Technology Sydney, NSW, Australia
†CSIRO Data61, Marsfield, NSW, Australia
‡School of Computing Science
Newcastle University, Newcastle upon Tyne, UK
Email: {deepak.puthal, rranjans, jinjun.chen}@gmail.com, Surya.Nepal@data61.csiro.au

Abstract— **Cloud computing and big data analysis are gaining lots of interest across a range of applications including disaster management. These two technologies together provide the capability of real-time data analysis not only to detect emergencies in disaster areas, but also to rescue the affected people. This paper presents a framework that supports emergency event detection and alert generation by analyzing the data stream, which includes efficient data collection, data aggregation and alert dissemination. One of the goals for such a framework is to support an end-to-end security architecture to protect the data stream from unauthorized manipulation as well as leakage of sensitive information. The proposed system provides support for both data security punctuation and query security punctuation. This paper presents the proposed architecture with a specific focus on data stream security. It also briefly describes the implementation of security aspects of the architecture.**

Keywords—Cloud Computing; Big data analysis; data security; disaster management.

I. Introduction

Natural disasters are mostly unpredictable events and arise within very short spans of time. Therefore, technology has to be developed to capture relevant data with a minimum delay, safely deliver the data to a cloud data center for analysis and process the data in real time to detect events and develop situation awareness. There are several environmental disasters, e.g. flash flood, earthquake, land slide, cyclone, tornado, tsunami, storms, that need real-time data analysis to detect critical events and protect human lives.

Data security plays an important role in such systems. In disaster management systems, data can be originated from a variety of sources. For example, wireless sensors can be a source of data and can quickly respond to rapid changes of the environment and send the sensed data to cloud data centers. RFID sensors, surveillance cameras, social media (i.e. Twitter and Facebook) all represent sources of data for natural disasters. There is lots of work already being done on real-time data collection [11][13], stream data processing [8][9], and end-to-end security of big data streams [11][12][13]. In this paper, we are proposing an integrated framework for disaster management. We then propose a security architecture for data streams with the aim of controlling access to the streaming data by providing end-to-end security. Our solution aims to use efficient data collection, and deploy appropriate data protection within the end-to-end paradigm (source to user/query processor at the cloud), as well as using efficient data analytics technology to detect or predict the chances of any kind of disaster occurring. Our architecture focuses on secure data stream analysis and emergency alert generation. Furthermore, the proposed architecture is able to detect the emergency event/natural disaster in real time and generate alert messages to make people as well as emergency service providers aware of the situation. The main contributions of the paper can be summarized as follows:

- Development and design of a novel architecture to securely collect data and detect emergency events.
- Propose architecture maintain end-to-end security and keep data freshness during the data analysis process.
- Consider data from a variety of sources such as sensors, mobile devices and social networks to generate alerts if there is any emergency in the source area.
- Finally proposed security verification model and followed by information flow control of big data streams.

The remainder of the paper is organized as follows: Section II describes requirements and challenges to design the architecture for disaster management. Section III describes the proposed architecture and descriptions. Section IV presents our proposed security verification model followed by access control architecture and experimental results. Finally we conclude the paper in Section V.

II. Requirements and Challenges

Big data has emerged as a new area with tremendous potential in many domains, where data analytics techniques have been applied to extract actionable information from the data. A massive amount of data is collected continuously from a variety of sources. Examples of data sources include sensor networks, wireless networks, radio frequency identification (RFID), customer click streams, telephone records, multimedia data, scientific data, surveillance cameras, and social networks. In addition to its pure volume, big data also exhibits other unique characteristics in comparison with traditional data computation. For example, big data is commonly unstructured and requires real-time analysis. As a result, big data analytics in the cloud has emerged as a new popular research topic, which also brings new challenges in the data processing life cycle starting from data collection, integration, and analytics to data privacy and security. These challenges require a new system architecture for data acquisition, transmission, storage, and large-scale data processing with built in features of data security and privacy.

Many of the data sources emit data as a stream, which is an ordered sequence of data instances that can be read in real time using limited computing and storage capabilities. These sources of data are characterized by being open-ended, flowing at high-speed, and being generated by non-stationary distributions. With the increasing bandwidth in electronic devices, data streams in modern systems often transmit large amounts of data. In many situations, the information in a data stream may be used to make quick decisions in real-time that affect one or more other operations of the device processing the stream or a related business. Thus, it is often desirable for a device to process a received data stream as quickly and efficiently as possible.

There are two examples of big data stream processing as described in the following subsection. These examples clearly show that efficient data analysis constitutes an asset that can be used to protect human lives and infrastructure from natural hazards. One of the major concerns is to ensure that the data collected for analysis is in its original form and from legitimate sources. However, collecting data from different sources, verifying the originality of data, and processing data to extract actionable information in near real time is a challenging problem that demands a secure big data analytics framework. In this article, we present a comprehensive big data analytics architecture for disaster management that takes care of data throughout its life cycle, begins with generation of data streams from a variety of sources and ends with generation of alerts after processing. There are five key components in the architecture, namely collection, evaluation, coalition, analysis, and dissemination. These five components form a big data value chain as shown in Figure 2. One of the key aspects of our architecture is a security framework for collecting data. Though the architecture and security protocol are described using a disaster management scenario, they are equally applicable to other applications that demand secure real time stream data processing.

A. *Motivating Examples*

Supervisory Control and Data Acquisition (SCADA) systems are used to monitor and control a power plant or equipment in industries such as telecommunications, water and waste control, energy, oil and gas refining and transportation. A SCADA system gathers information, such as a leak on a pipeline, and transfers the information back to the server in the cloud, alerting the home station that the leak has occurred; it carries out necessary analysis and control, such as determining if the leak is critical, and displays the information in a logical and organized fashion [16]. Davidson et al. [17] have highlighted the need for online data analysis for alarm systems and developed a robust multi-agent system for continuous online usage within the power industry. They used multi-agent system technology to automate the management and analysis of SCADA and digital fault recorder (DFR) data.

Disaster management is another important application of stream data processing. Real-time responses to crises and disaster events, such as floods, fires, hurricanes, tsunamis, and man-made disasters, are dependent on past knowledge as well as knowledge obtained from effective real-time integration and utilization of data streaming from multiple sources including sensors, mobile device, and social media. Timely analysis of data from these sources can help rescue teams, medics, and relief workers in (i) sending early warning to people, (ii) saving lives, (iii) coordinating rescue and medical operations, and (iv) reducing the harm to infrastructures. Timely acquisition and processing of data from sources and extraction of accurate actionable information plays an important role in coordinating disaster prevention and management. Castillo-Effen et al. [20] have provided an architecture to manage flash floods by collecting data from different sources including sensor nodes deployed in the area, mobile phones (from local people?), and social media. Such collected data need to be analyzed in real time at the control center (deployed on the cloud) for timely alerts. Ramesh [18] proposed an architecture for landslide event detection using data collected from sensor networks. Tseng [19] proposed an adaptive framework for earthquake detection.

Table I. Comparison between stream processing and batch processing

	Stream processing	Batch processing
Input	Stream of new data or updates	Data chunks
Data size	Infinite or unknown in advance	Finite and known
Hardware	Typical single limited amount of memory	Multiple CPU memories
Processing	A single pass or few passes over data	Processed in multiple rounds
Storage	Not store or store non-trivial portion in memory	Store
Time	A few seconds, milliseconds or microseconds	Much longer
Application	Web mining, sensor networks, surveillance data.	Widely adopted in almost every domain

III. Architecture

The big data analytics architecture considers a number of factors such as data life cycle, real-time processing, security and privacy, and application scenarios. The application scenario of big data not only provides the application requirements, but also helps to describe the different components, protocols and functionalities of the architecture. In this article, we have chosen natural disaster management as the application. The proposed

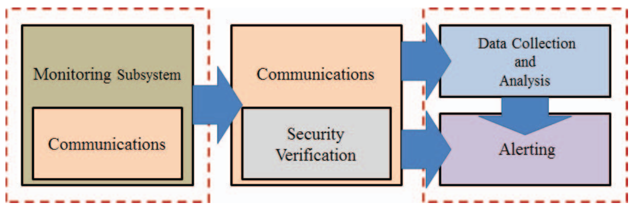

Figure 1: Block Diagram

cloud-based big data analytics system focuses on real-time emergency event detection, followed by corresponding alert message generation. We chose cloud as an infrastructure because it provides a scalable computing platform and almost infinite computation resources. The objective of the proposed architecture is to analyze the sensing data in a real-time to detect and handle emergency situations. The data are collected from various sources in different formats and the collected data are processed and evaluated in the cloud. The purpose of the data analysis is to automatically generate alerts to authorities or end users and the inhabitants at risk.

There are several common data sources of natural disasters such as floods, fires, hurricanes, tsunamis, and man-made disasters. Sensors, surveillance cameras, social networks etc. are always the sources of the data in these incidents. By considering all these sources, we broadly divide the sources into two types: known source and unknown source. The known source belongs to sensors, surveillance cameras, where the source's identity is known to us or the query processor; social media belongs to unknown sources where a source's identity is not always known to us or the query processor.

There are two common approaches in evaluating or processing the data: batch processing and stream processing. Batch processing works in a store-and-process fashion [5], but for real-time event detection it is important to process the data on data streams. Therefore, batch processing is not capable to detect events in real time. The comparison between batch processing and stream processing features is shown in Table I. Data stream processing is an emerging computing paradigm where a huge amount of data (Big Data) must be processed in real time (with minimal delay).

Alert message generation in the cloud depends on the data set available for analysis. If the data is being modified in transit or the data is from malicious sources, there is a possibility of false alarms being generated or a failure to detect a real emergency event. In such situations, maintaining originality of the data is very important. It is of fundamental importance to develop a system which is sensitive and able to effectively recognize hazardous conditions. At the same time, a system should be intelligent enough not to overreact and trigger false alarms. So it is mandatory to deploy security verification mechanisms to collect only original data at a cloud data center for processing.

There are several solutions proposed for communication between the source and the centralized station (cloud) and data analysis on cloud for disaster management [5][18][20]. The major challenge of disaster management is to analyze the collected data in real time and protect the integrity and confidentiality of data until event detection and decision making.

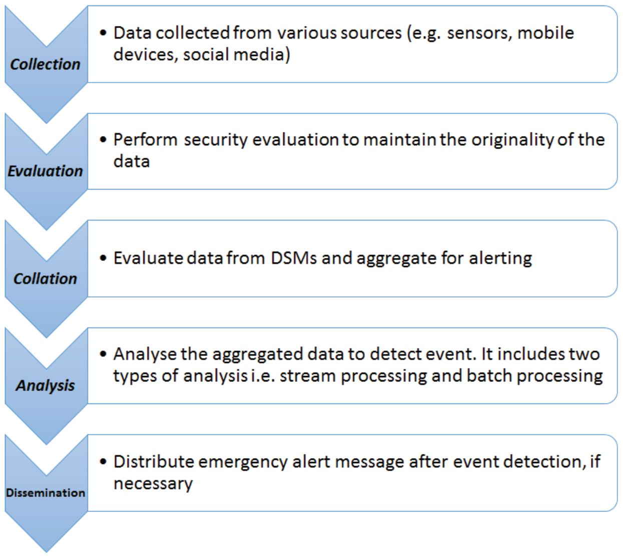
Collection
• Data collected from various sources (e.g. sensors, mobile devices, social media)
Evaluation
• Perform security evaluation to maintain the originality of the data
Collation
• Evaluate data from DSMs and aggregate for alerting
Analysis
• Analyse the aggregated data to detect event. It includes two types of analysis i.e. stream processing and batch processing
Dissemination
• Distribute emergency alert message after event detection, if necessary

Figure 2: Five sequential steps to define the architecture

The block diagram of the proposed big data analytics for disaster management on the cloud is shown in Figure 1. This figure contains three major components: data collection, communication, and data analysis/alerting. This figure shows the complete architecture from source to alert message generation with three major components and several subcomponents. The monitoring subsystem, located in the event area (source area), performs data acquisition of all relevant variables and incorporates internal communication links that allow the transmission of information from the spatially distributed locations to the cloud. Security verification should be processed in real time before data is submitted for analysis in the cloud and generate alert messages. Figure 3 shows the complete architecture of the block diagram from Figure 1. Based on the processed data, an alerting subsystem is responsible for generating alert messages which can be broadcasted by different means. We understand these perceptions via the following requirements:

- *Effectiveness*: The application should fulfil a user's needs by providing timely alert for emergencies.
- *Efficiency*: How much benefit do users get from this system (false alarm/alerting)? The alert generation should automatically be done on user's behalf.
- *Intelligibility*: Users need to understand which location information they get, time to accident, and the evacuation process. The proposed system should employ such information, and users need the above specified information.
- *Security*: Data should not be modified before reaching the cloud or end user.
- *Satisfaction*: Users should find the experience to be overall satisfactory.

We considered the above specific points while describing the proposed system model. We describe the complete architecture with five different standard steps such as collection, evaluation, collation, analysis, and dissemination as shown in Figure 2.

Figure 3 shows the complete architecture of big data stream query processing with possible security attacks, and the DSM (Data Stream Manager) structure for a security framework. Data transfer to stream, clustering and Bayesian network are standard data processing, but we did not address all these in our architecture. The figure shows the complete architecture of the system from source device to cloud for data analytics, security framework and alert message generation and distribution. These five steps are defined as follows.

1. *Collection*: Data collected from different sources such as sensors, mobile devices, and social media for data analysis and event detection.
2. *Evaluation*: Stream data verified for security evaluation to maintain the originality of the data and go for online stream query processing.
3. *Collation*: Evaluate data from different DSMs and aggregate together for event detection and alert generation. Data also move to the cloud for batch processing.
4. *Analysis*: Analyze the data to detect event and generate alert messages. It includes two types of analysis: stream processing and batch processing.
5. *Dissemination*: This step is the output of the data analysis and distributes emergency alert messages if necessary.

We have described the complete architecture by considering the above five steps as follows. The description starts with data collection and ends with alert dissemination. The proposed architecture may be applicable for different applications though our description is based on a disaster management application.

A. Collection

Data are collected from various sources for analysis and event detection. As stated above we divide data sources into two types: known source and unknown source. Known sources are those whose source ID or address is known such as weather sensors, traffic sensors, and surveillance cameras. Unknown sources include social networks such as Twitter and Facebook where data sources are unknown to the data analyzer. Data are transmitted towards a cloud data center through wired or wireless connection, where sources are mobile phones, sensors, etc. The social networks data are collected through the internet. Collected data are in different formats such as video, sensor data, websites, etc. These collected data streams move to the STREAM collection system [23] before DSM for data aggregation. The aggregated data move to individual DSMs for originality of the data evaluation and the process is described in the following subsection.

B. Evaluation

There are always two types of evaluation process in big data: batch processing and stream processing. In this paper, we focus on stream processing to detect emergency events in real-time. In the evaluation step, we address the security evaluation before data analysis. Generally sources use an untrusted medium to transfer sensed data to the cloud for evaluation/analysis. So security verification is one of the important features that need to be addressed on big data streams to filter out unwanted and modified data. According to the features of big data streams, volume and velocity of data are very high, so we cannot put data streams in halt to process. We consider four important features of big data streams for security verification following the conclusions in [11].

1. Security verification needs to be performed in real time (on-the-fly).
2. The verification framework has to deal with a high volume of data.
3. Data items can be read once in the prescribed sequence.
4. Original data is not available for comparisons like store-and-process.

DSM processes data streams on-the-fly. The needs of on-the-fly processing include the amount of input data that discourages the use of persistent storage, the requirement of providing prompt results, etc. DSM is designed to handle high-volume and bursty data streams with a large number of complex continuous queries. The way DSM handles streams of tuples is similar to how a conventional database system handles relations. In addition, DSM needs to do the security verification of the data blocks on-the-fly. We implemented the real-time security verification framework called Dynamic Prime-Number Based Security Verification (DPBSV) [11]. The implementation with some results is described in the next section. The security verification is implemented before data stream query processing or data analysis as shown in Figure 3. This assures that data analytics are performed on the original data. This application is quite sensitive to maintain the originality of the data for analysis and alert generation.

Figure 3 shows an overall architecture for the big data stream process from sensing devices to the cloud data centers, including our proposed security framework. It starts with a three-step process: collection, processing, and storing. All the query and security related processes are handled at DSM. It is important to note that the security verification of stream data has to be performed before query processing and it has to be done in real time (with minimal delay) with a fixed (small) buffer size. The processed data is stored in the cloud storage for batch processing. Queries used in DSM are defined as "continuous" since they are continuously standing over the *streaming* data. Results are pushed to the user each time the streaming data satisfies the query predicate. The queries, including security verification, are defined as a direct acyclic graph where each node is an operator and edges define data flow.

C. Collation

Collation or correlation of the evaluated data from DSM is further processed for event detection, which can be processed in two ways: stream (real-time) processing and batch processing, at SMS alert management system and Hadoop, respectively. The evaluated data from different DSMs are aggregated together and sent to SMS alert management system and also to the cloud (NoSQL/Hadoop) for batch processing. Here, we correlate the data from different sources and send for analysis to generate alerts for different events. The current architectural diagram shows that the online processing/alert system and the batch processing happens in Hadoop.

Data aggregation is a process where information is gathered and expressed in a summary form for purposes such as

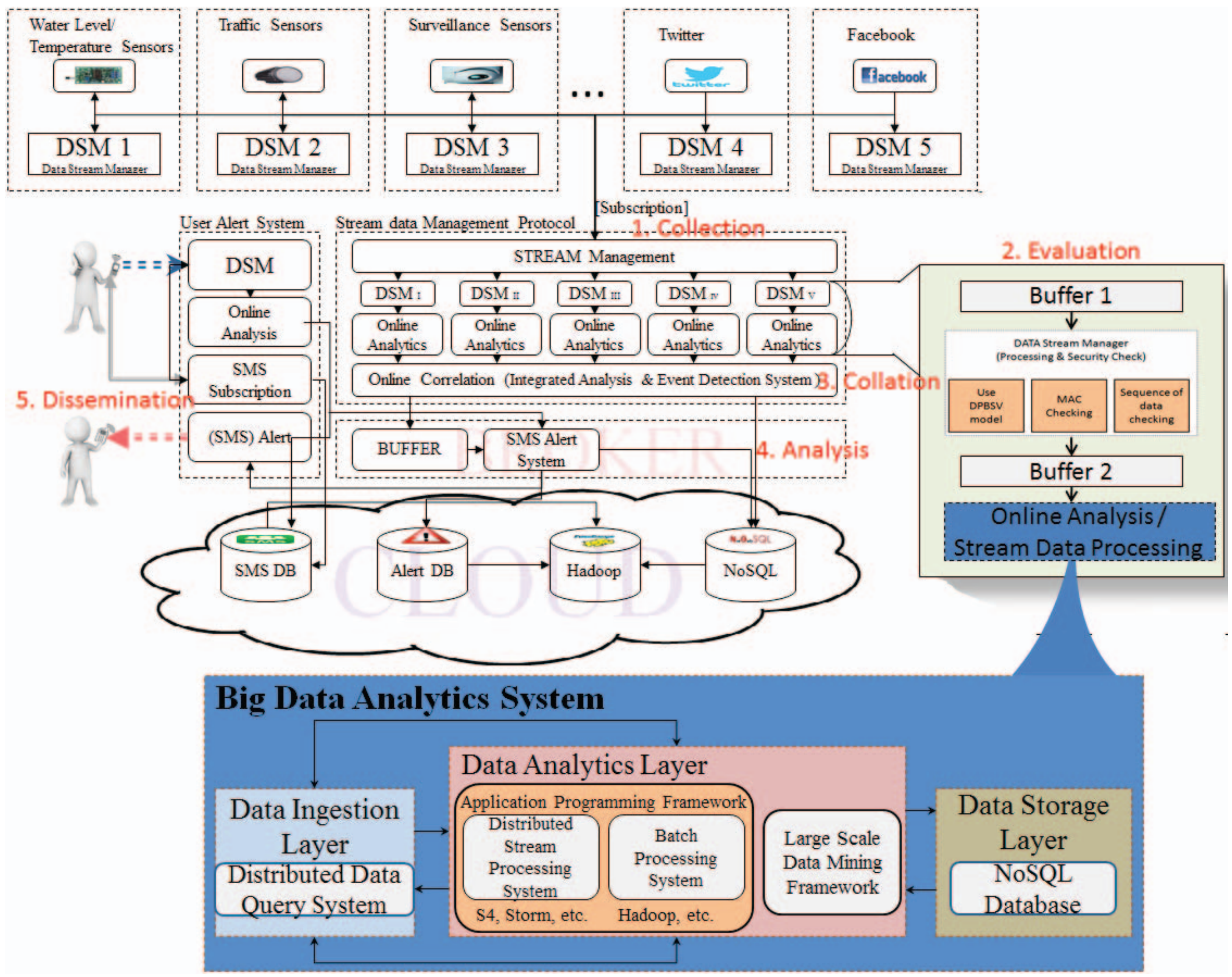
Water Level/ Temperature Sensors
Traffic Sensors
Surveillance Sensors
Twitter
Facebook
DSM 1 Data Stream Manager
DSM 2 Data Stream Manager
DSM 3 Data Stream Manager
DSM 4 Data Stream Manager
DSM 5 Data Stream Manager
[Subscription]
1. Collection
User Alert System
Stream data Management Protocol
STREAM Management
DSM
Online Analysis
SMS Subscription
(SMS) Alert
Online Analytics
Online Correlation (Integrated Analysis & Event Detection System)
3. Collation
2. Evaluation
Buffer 1
DATA Stream Manager (Processing & Security Check)
Use DPBSV model
MAC Checking
Sequence of data checking
Buffer 2
Online Analysis / Stream Data Processing
5. Dissemination
BUFFER
SMS Alert System
4. Analysis
SMS DB
Alert DB
Hadoop
NoSQL
CLOUD
Big Data Analytics System
Data Analytics Layer
Data Ingestion Layer
Distributed Data Query System
Application Programming Framework
Distributed Stream Processing System
Batch Processing System
S4, Storm, etc.
Hadoop, etc.
Large Scale Data Mining Framework
Data Storage Layer
NoSQL Database

Figure 3: Architectural diagram for classification of streaming sources to alert generation with security verifications, access control and stream data analysis

statistical analysis. One popular example of data aggregation is online analytic processing (OLAP); this is a simple type of data aggregation used for marketing where an online reporting mechanism is used to process the information. The information from such resources can then be used for early event detection and alerting.

D. Analysis

Streaming data transmissions can be unreliable in many environments. Nowadays data sources generate terabytes to petabytes of data on a daily basis [14]. Given the volume of data being generated, real-time computation has become a major challenge. A scalable real-time computation system that we have used effectively is the open-source Apache Storm tool, which was developed at Twitter and is sometimes referred to as "real-time Hadoop". The example project, called "Speeding Alert System", analyzes real-time data and raises a trigger and relevant data to a database when there is any emergency at the source side.

We described the stream query processing to detect the event. The data stream concept of continuous queries is graphs of interconnected operators that allow for rich, real-time analysis of data. A stream of data is a potentially infinite sequence of tuples, denoted as (T1, T2, …, Tn); we refer to a generic attribute Ti of tuple j as j:Ti. We assume that all tuples have a time stamp attribute set at the data sources. The data sources have clocks that are well synchronized with other system nodes as in [2]. Each query is modeled as a network of connected operators. A connection represents a data flow. Typical query operators of DSMs are filter, map, union, join, and aggregate [1]. These operators correspond to relational algebra operators. Operators can be classified as stateless (filter, map and union) or stateful operators (join and aggregates) [3]. As the nature of the data stream is infinite, stateful operators perform their computation over sliding windows of tuples defined over a period of time (e.g. tuples received in the last hour) or as a fixed number of tuples (e.g. last 100 tuples). Figure 3 shows the high level abstraction of stream data processing at DSM. The incoming streams (on the left) produce data indefinitely and drive query processing. Processing of continuous queries typically requires intermediate states, which are stored as Scratch Store in Figure 3. This state could be stored and accessed in memory or on disk. Although we are concerned primarily with the online processing of continuous queries, in many applications stream data may also be copied to an archive, for preservation and possible offline processing of expensive analysis or mining queries. Refer to [8] for further information on stream data processing in datacenter clouds.

Such large data masses termed Big Data is calling for new approaches to storage and processing of data. Scaling using hard disk parallelism is one of the design goals of scalable batch processing in the cloud. MapReduce (Hadoop) is scalable batch processing technology in the cloud. The MapReduce framework takes care of all issues related to parallelization, synchronization, load balancing, and fault tolerance. All these details are hidden from the application developer. When deciding whether MapReduce is the correct fit for an algorithm, one has to remember the fixed data-flow pattern of MapReduce. The algorithm has to be efficiently mapped to this data-flow pattern in order to efficiently use the underlying computing hardware. The data is available if at least one machine replica is up and running.

E. Dissemination

In this step, proposed framework generate the emergency alert after data analysis and send alerts to the mobile phone. Alert dissemination is always followed by data analysis (previous step). Alerts are generated from SMS Alert Management System, by following alert DB from Figure 3.

The construction and display of operator messages representative of alert conditions in a network is described as follows. Code points, which are strings of bits, are generated in response to an event in a device attached to the network [4]. The code points are utilized to index predefined tables that contain moderately short units of instant messages in administrator selectable dialects to be utilized as a part of building an administrator's data show. The messages are free of the particular warning sending item insofar as an alarm collector is concerned. The code points are progressively orchestrated so that if the alert collector does not have the most forward set of messages, the alert recipient will show a more generic message which is still illustrative of the occasion.

IV. Security Framework for Big Data Streams

We are analyzing the data in two different modules i.e. stream processing and batch processing, as described in the previous architecture description. Table I indicated the basic difference between batch processing and stream processing features. Possible attacks and their classifications of the data streams are described in [11]. The streaming data security can be broadly divided into two types of security punctuations: (i) the "data security punctuations" (dsps) describing the data-side security, and (ii) the "query security punctuations" (qsps) representing the query-side security [21]. We introduced a new module called Data Stream Manager (DSM), where we perform security verifications of data streams for *dsps* before data analysis. We proposed Dynamic Prime Number Based Security Verification (DPBSV) and Dynamic Key Length Based Security Framework (DLSeF) methods for big data streams based on the shared key derived from synchronized prime numbers in our earlier works [11, 13]. The proposed DPBSV scheme for big data stream processing is based on a common shared key that is updated dynamically by generating synchronized pairs of prime numbers [11]. Later, to make it more efficient by reducing the computational overhead and buffer size, we proposed DLSeF which is based on the shared key derived from synchronized prime numbers [13]. These two techniques were proposed to maintain end-to-end security (source to processing unit, i.e. DSM) of big sensing data streams and perform security verification at DSM. These models introduced a small buffer before DSM, because we need to satisfy the four features of big data streams (from Section III). A buffer can be used to halt the data packets before processing. These methods focus on reducing the buffer size as well as the halting time of data blocks with a fast security verification model.

A. Security verification model

Here we proposed a security model by following the system model of DPBSV and DLSeF. We follow [11][13] to design a DSM which is capable of handling high volume, velocity and variety data streams from multiple sources. In addition, the DSM is responsible for performing the security verification of the incoming data streams in near real time to synchronize with the processing speed of Stream Processing Engines (SPE). In this security model, source sensors are deployed with Intrusion Detection Systems (IDS). Sensor-based IDS monitor a sensor's behavior and generate alerts on potentially malicious activities onboard and network traffic [24]. IDS can be set inline, attached to a spanning port of a sensor. The idea here is to allow access to all packets we wish the IDS to monitor. LEoNIDS (low-latency and energy efficient network IDS) is a system that determines the energy expectancy trade off by giving both lower power utilization and lower recognition expectancy in the meantime [25].

In our architecture, the data streams are always in an encrypted format when they arrive at the DSM. Our idea is that while encrypting the data packets at the source sensors, we attach a sensitivity level of data to each individual data packet. We apply different keys to encrypt the data packets for different data sensitivity levels to maintain data security based on data sensitivity. This ensures data streams maintain authenticity and integrity, whereas confidentiality is based on data sensitivity level. In a very generic representation, if we need *n* levels of data security then *n-1* keys ($Shared\ Key(K_{SH})$) are required for encryption/decryption. Here we are using longer keys for strong encryption whereas a shorter key length can be used for weak encryption (See Table II).

Here we consider three levels of security i.e. strong confidentiality, partial confidentiality, and no confidentiality; and two keys (i.e. *k1, k2*) for encryption methods. The strong encryption method uses *k1* and is used to provide strong confidentiality, and the weak encryption method uses *k2* to support partial confidentiality. Note that we do not need to encrypt the data packets for no confidentiality. Data packets can

Table II. Notations Symmetric key (AES) algorithm takes time to get all possible keys using most advanced Intel i7 Processor.

Key Length	**32**	**64**	**128**
Key domain size	4.295e +09	1.845e +19	3.4028e +38
Time (in nanoseconds)	7.301e +09	3136e +19	5.7848e +35

be transmitted to DSM by encrypting the data stream, where deployed sensors are always associated with sensitivity level. We are going to apply encryption methods (strong/weak encryption) based on the data sensitivity or confidentiality level. These two shared keys used for encryption and decryption are always initialized and distributed by the DSM. DSM always distributes the shared keys to source sensors before it expires.

$$Shared\ Key(K_{SH}) = \begin{cases} security\ level - n & key\ (1) \\ \vdots & \vdots \\ security\ level - 3 & key\ (n-2) \\ security\ level - 2 & key\ (n-1) \\ security\ level - 1 & no\ key \end{cases}$$

Every encrypted data packet is always associated with a flag value of 1 or 0 which represents the data sensitivity level and shared key used for data encryption. Here we consider 1 is for strong encryption i.e. high sensitivity data, 0 is for weak encryption and no flag value represents no encryption. The security verification needs to be done on-the-fly (i.e. near real-time) with smaller buffer size. The queries, including security verification, can be defined as a directed acyclic graph and each node is an operator and edges defined as data flows between the nodes.

B. *Information flow control model*

According to Sandhu's definition on lattice based access control, users are defined as humans, subjects are processes and objects are files [22]. We follow the same way to define our system, where users are humans and subjects are query processors (QP) and objects are data blocks after security verification at DSM. We use a standard five steps/stages process for the information flow control model. The five stages are Stage 0: structure module; Stage 1: information flow between the levels; Stage 2: recursive lattice construction; Stage 3: conflict of interest; Stage 4: decision over data access. By following the above steps, information flow control policies specify under which conditions information may be exchanged or accessed by the users and query processor.

From the previous description of security verification, sensors always generate the data packets with the format {DATA; 1/0; S_i, S_i/DSM}. DATA means encrypted data packets, 1/0 means the flag value (FV) to define the data sensitivity level, S_i means the source of the data and finally S_i/DSM shows who has the influence to modify the data packets. After security verification at DSM, we check the flow model to define the access control. We made the flow model simple and defined the static lattices for lightweight processing

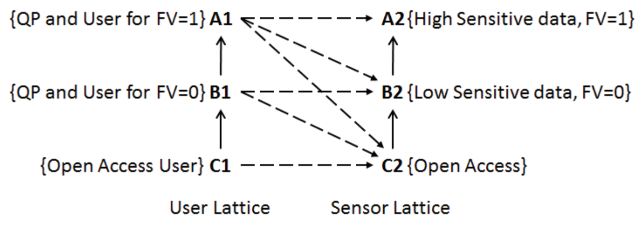
{QP and User for FV=1} A1
A2 {High Sensitive data, FV=1}
{QP and User for FV=0} B1
B2 {Low Sensitive data, FV=0}
{Open Access User} C1
C2 {Open Access}
User Lattice
Sensor Lattice

Figure 4: Lattice model for data access

over big data streams. There are three different ways of flow management, namely no management, centralized management and distributed management. We follow centralized management at DSM after security verifications. We defined our flow model (FM) as follows

FM = <S, O, SC, →>
Where: S = Subjects
P = Processes
SC = Security Classes
→ = Can-flow relation on SC

Here we did not add an operations option in our FM, because our focus is only to read or access the data stream instead of writing. We define a static lattice for sensors, which will label incoming data streams and a static lattice for users to define the access class for both user and query processor. The lattice structure with access policy is shown in Figure 4. The lattice is a Directed Acyclic Graph (DAG) with a single source and information is permitted to flow from a lower class to upper class. We have divided our lattice into three classes i.e. {A, B, C}, where 1 is for user lattice i.e. {A1, B1, C1} and 2 is for sensor lattice i.e. {A2, B2, C2}. We defined A as the highest class (i.e. for high sensitivity information), followed by B defined as lower class (i.e. low sensitivity data) and finally C is defined as lowest class for open access information.

Figure 4 shows the access policy, where a class of user lattice has access to the same and lower level classes of sensor lattice. We follow a modified Chinese Wall model for information flow control by Snadhu [22] to define the conflict of interest between the classes. This access policy always satisfies the properties of reflexive, antisymmetric and transitive (i.e. partial order). Partial ordering → on a set L is a relation where:

$\forall\, a \in L,\ a \to a$ holds (reflexive)

$\forall\, a,b \in L$, if $a \to b,\ b \to a$, then $a = b$ (antisymmetric)

$\forall\, a,b,c \in L$, if $a \to b,\ b \to c$, then $a \to c$ (transitive)

We follow this partial order relation between the classes of lattice to define access control of big data streams. This tends to query security punctuations (*qsps*) of data streams.

V. Conclusions

Diversity of existing big data analytics frameworks in cloud makes the process of emergency decision making for software engineers, solution architects, or infrastructure administrators challenging. To address such an issue, this study explores the holistic system framework from data collection to security framework, data analytics and alert message distribution. Here we also explored the security framework of big data streams at DSM followed by access control of data stream by information flow model.

VI. Acknowledgement

This research is funded by the Australia India Strategic Research Grant titled “Innovative Solutions for Big Data and Disaster Management Applications on Clouds (AISRF – 08140)” from the Department of Industry, Australia.

References

[1] Gulisano, Vincenzo, Ricardo Jimenez-Peris, Marta Patino-Martinez, Claudio Soriente, and Patrick Valduriez."Streamcloud: An elastic and scalable data streaming system.” *IEEE Transactions on Parallel and Distributed Systems, 23*(12), pp. 2351-2365, 2012.

[2] N. Tatbul, U. C¸ etintemel, and S.B. Zdonik, "Staying Fit: Efficient Load Shedding Techniques for Distributed Stream Processing." *Proc. Int’l Conf. Very Large Data Bases (VLDB)*, pp. 159-170, 2007.

[3] Gulisano, Vincenzo, Ricardo Jimenez-Peris, Marta Patino-Martinez, and Patrick Valduriez. "Streamcloud: A large scale data streaming system." *IEEE 30th International Conference on Distributed Computing Systems (ICDCS),*pp. 126-137, 2010.

[4] Anderson, Catherine J., Arthur A. Daniel, Thomas J. Gelm, Raymond F. Kiter, John P. Meeham, Robert E. Moore, John G. Stevenson, and Lawrence E. Troan. "Method and apparatus for communication network alert message construction." *U.S. Patent 4,965,772*, issued October 23, 1990.

[5] Puthal, Deepak, B. P. S. Sahoo, Sambit Mishra, and Satyabrata Swain. "Cloud Computing Features, Issues and Challenges: A Big Picture." *in International Conference on Computational Intelligence & Networks (CINE)*, pp. 116-123, 2015.

[6] TCG Trusted Platform Module (TPM) Specification, https://www.trustedcomputinggroup.org/specs/tpm/

[7] Nepal, Surya, John Zic, Dongxi Liu, and Julian Jang. "A mobile and portable trusted computing platform." *EURASIP Journal on Wireless Communications and Networking*, *2011*(1), pp. 1-19, 2011.

[8] Ranjan, Rajiv. "Streaming big data processing in datacenter clouds." IEEE Cloud Computing 1, no. 1 (2014): 78-83.

[9] Ji, Changqing, Yu Li, Wenming Qiu, Uchechukwu Awada, and Keqiu Li. "Big data processing in cloud computing environments." In 2012 12th International Symposium on Pervasive Systems, Algorithms and Networks, pp. 17-23. IEEE, 2012.

[10] Yu, Liyang, Neng Wang, and Xiaoqiao Meng. "Real-time forest fire detection with wireless sensor networks." In *Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005.*, vol. 2, pp. 1214-1217. IEEE, 2005.

[11] Puthal, Deepak, Surya Nepal, Rajiv Ranjan, and Jinjun Chen. "A dynamic prime number based efficient security mechanism for big sensing data streams." *Journal of Computer and System Sciences* (2016).

[12] Park, Taejoon, and Kang G. Shin. "LiSP: A lightweight security protocol for wireless sensor networks." *ACM Transactions on Embedded Computing Systems (TECS)* 3, no. 3 (2004): 634-660

[13] Deepak Puthal, Surya Nepal, Rajiv Ranjan, and Jinjun Chen. "DLSeF: A Dynamic Key Length based Efficient Real-Time Security Verification Model for Big Data Stream." *ACM Transactions on Embedded Computing Systems (TECS)*, 2016.

[14] Hu, Han, Y. O. N. G. G. A. N. G. Wen, T. Chua, and X. U. E. L. O. N. G. Li. "Towards Scalable Systems for Big Data Analytics: A Technology Tutorial." IEEE Access, Vol. 2, pp. 652 – 687, 2014.

[15] Kaddoura, Issam, and Samih Abdul-Nabi. "On formula to compute primes and the nth prime." *Applied Mathematical science, 6*(76), pp.3751-3757, 2012.

[16] Stamp, Jason, Phil Campbell, Jennifer DePoy, John Dillinger, and William Young. "Sustainable security for infrastructure SCADA." *Sandia National Laboratories, Albuquerque, New Mexico (www. sandia. gov/scada/documents/SustainableSec urity. pdf)* (2003).

[17] Davidson, Euan M., Stephen DJ McArthur, James R. McDonald, Tom Cumming, and Ian Watt. "Applying multi-agent system technology in practice: automated management and analysis of SCADA and digital fault recorder data." *Power Systems, IEEE Transactions on* 21, no. 2 (2006): 559-567.

[18] Ramesh, Maneesha V. "Real-time wireless sensor network for landslide detection." In *Sensor Technologies and Applications, 2009. SENSORCOMM'09. Third International Conference on*, pp. 405-409. IEEE, 2009.

[19] Tseng, Chun-Pin, and Cheng-Wu Chen. "Natural disaster management mechanisms for probabilistic earthquake loss." *Natural Hazards* 60, no. 3 (2012): 1055-1063.

[20] Castillo-Effer, M., Daniel H. Quintela, W. Moreno, R. Jordan, and W. Westhoff. "Wireless sensor networks for flash-flood alerting." In *Devices, Circuits and Systems, 2004. Proceedings of the Fifth IEEE International Caracas Conference on*, pp. 142-146. IEEE, 2004.

[21] Nehme, Rimma V., Hyo-Sang Lim, Elisa Bertino, and Elke A. Rundensteiner. "StreamShield: a stream-centric approach towards security and privacy in data stream environments." In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pp. 1027-1030. ACM, 2009.

[22] Sandhu, Ravi S. "Lattice-based enforcement of chinese walls." *Computers & Security* 11, no. 8 (1992): 753-763.

[23] Arasu, Arvind, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. "STREAM: the stanford stream data manager (demonstration description)." In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 665-665. ACM, 2003.

[24] Roesch, Martin. "Snort: Lightweight Intrusion Detection for Networks." In*LISA*, vol. 99, no. 1, pp. 229-238. 1999.

[25] Tsikoudis, Nikos, Antonis Papadogiannakis, and Evangelos P. Markatos. "LEoNIDS: a Low-latency and Energy-efficient Network-level Intrusion Detection System." *IEEE Transactions on Emerging Topics in Computing* 4, no. 1 (2016): 142-155.