
# ON BIG DATA MANAGEMENT IN INTERNET OF THINGS

A Thesis Presented to the Department of

Computer Science,

African University of Science and Technology, Abuja

In Partial Fulfilment of the Requirements for the Degree of

Master of Science

By

Mubarak Adetunji Ojewale

Abuja, Nigeria

June, 2016

ON BIG DATA MANAGEMENT IN INTERNET OF THINGS

By

Mubarak Adetunji Ojewale

A THESIS APPROVED BY THE COMPUTER SCIENCE DEPARTMENT

RECOMMENDED: _____

Supervisor, Dr. Ekpe Okorafor

_____

Head, Department of Computer Science

APPROVED: _____

Chief Academic Officer

_____

Date

## ABSTRACT

The Internet of Things (IoT) has generated a large amount of research interest across a wide variety of technical areas. These include the physical devices themselves, communications among them, and relationships between them. One of the effects of ubiquitous sensors networked together into large ecosystems has been an enormous flow of data supporting a wide variety of applications. In this work, we propose a new “IntelliFog-Cloud” approach to IoT Big Data Management by leveraging mined historical intelligence from a Big Data platform and combining it with real-time actionable events from IoT devices at the Fog layer to reduce action latency in IoT applications. This approach is demonstrated through an advertisement service simulation with VoltDB technology where advertisements are being served on mobile phones based on geo-location and highest bids, and displayed from user interests determined by data analytics of activities on the web. Results from the demonstration show very low latency overhead of processing large hundreds of thousands of transactions. This approach improves both action latency and accuracy of real-time decisions in IoT applications.

## ACKNOWLEDGEMENT

All praises are due to Allah alone, the Lord of the universe. May his peace and blessings continue to abide by the noble soul of his messenger Muhammad, his household, companions and all those who follow the guidance. Allah has fulfilled His promise and He has helped his slave. AlhamduliLah.

To my father, Alhaji Isa A. Ojewale, my mother, Hajiya Semiat Ojewale (nee Badmus), my brothers (AbdulWaasi, Ridwan and Ashrofs), sisters (Umu Zayd and RahmotaLlah) and all the Ojewales and Badmus, I appreciate your prayers, advices, financial and mental support before and during the course of my program. I pray Allah rewards you all abundantly.

I thank the Nelson Mandela Institute (NMI) considering me worthy of a scholarship for my Master of Science degree program out of many other qualified candidates who merited this award. Your faith in me has and will continue to yield the expected dividends.

I would like to express my deepest appreciation to my supervisor Dr. Ekpe Okorafor whom without his guidance, encouragement and persistence this thesis would not have been possible. You gave us me the needed freedom to explore a lot while giving your timely inputs at the appropriate moments. This has really helped my research skills. Thank you so much sir.

My profound gratitude goes to all the faculties in Computer Science and Engineering department especially Professor Mamadou Kaba Traore, Professor Lehel Csato, Professor Ben Abdallah, Professor Mohamed Hamada and Dr Ekpe Okorafor, my supervisor. I appreciate your constructive criticisms and the knowledge you impacted on me.

This acknowledgement would not be complete without the mention of the “5 of 5”, my Saka, Umu to be wherever she is, Imam Isa, Bro AbdulMajeed, AbdulHakeem, AbdulGaffar, Imam Usman, Ignace, Tabot, Salami, Gbade, Isun, all MSc and PhD students of the Computer Science Stream, and the entire AUST community.

## DEDICATION

To Allah; the Beginning without an end; the End without a beginning

**TABLE OF CONTENTS**

## LIST OF FIGURES

# CHAPTER ONE

# INTRODUCTION

## 1.0 Introduction

Advances in sensor technology, communication capabilities and data analytics have resulted in a new world of novel opportunities. With improved technology such as nanotechnology, manufacturers can now make sensors which are not only small enough to fit into anything and everything but also more intelligent. These sensors can now pass their sensing data effectively and in real time due to improvements in communication protocols among devices. There are now, also, emerging tools for processing these data. These phenomena combined have made the Internet of Things (IoT) a topic of interest among researchers in recent years. Simply put, the IoT is the ability of people's "things" to connect with anything, anywhere and at any time using any communication medium. "Things" here means connected devices of any form. It is estimated that by 2020 there will be 50 to 100 billion devices connected to the internet [2]. These devices will generate an incredible amount of massively heterogeneous data. These data, due to their size, rate at which they are generated and their heterogeneity are referred to as "Big Data". Big Data can be defined with the famous three characteristics known as the 3Vs: volume, variety, and velocity or sometimes 5Vs, including Value and Veracity [3], [12]. These data, if well managed, can give us invaluable insights into the behaviour of people and "things"; an insight that can have a wide range of applications.

The potentials of incorporating insights from IoT data into aspects of our daily lives are becoming a reality at a very fast rate. The acceptability and trust level is also growing as people

have expressed willingness to apply IoT data analytics results in situations even as delicate as stock market trading [1]. These developments inform the need for efficient approaches to manage and make use these huge and fast-moving data streams. Distributed processing frameworks such as Hadoop have been developed to manage large data but not data streams. One major limitation of distributed settings such as Hadoop is latency. They are still based on the traditional Store-Process-and-Forward approach which makes them unsuitable for real-time processing, a contrast with the real-time demands of the current and emerging application areas [4]. Store and forward also will not be able to satisfy the latency requirements of IoT data because of the velocity and the unstructured nature of the data. Stream processing frameworks like Apache Storm and Samza are then introduced to solve this problem. In stream processing, data from data sources are continuously processed as they arrive and do not need to be stored first. This improves latency, especially in stateless stream processing which processes data as it comes without reference to the current situation of the system.

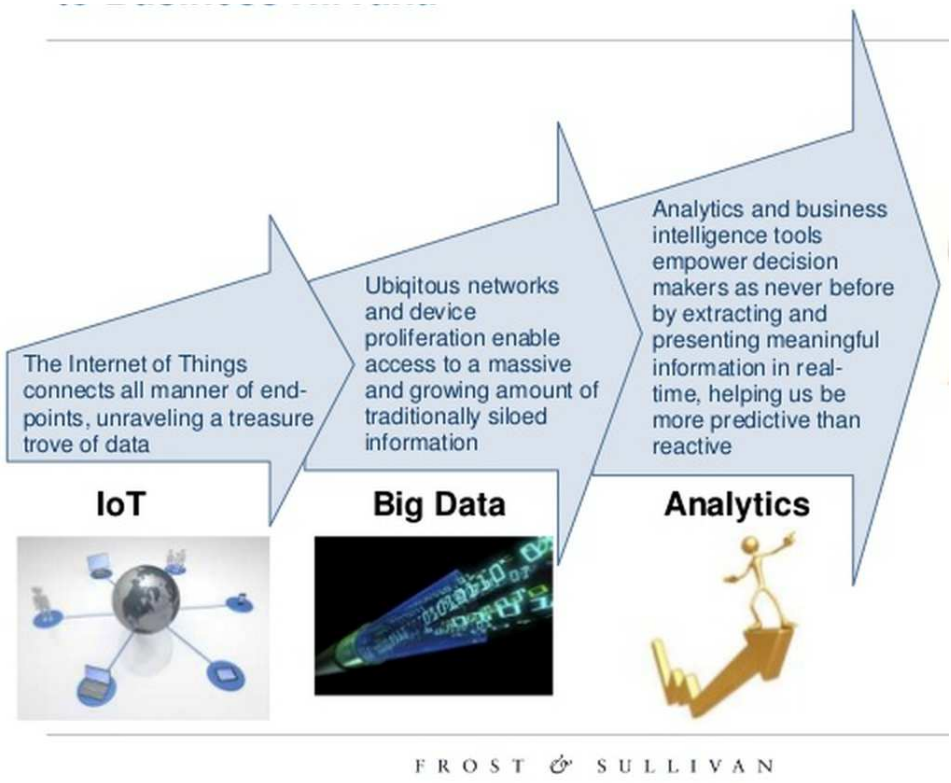


*Figure 1.0: IoT Big Data Management. (Source Sullivan and Frost)*

Stream processing frameworks, however, are more general for processing data streams and are not tailored for the specific needs of IoT data management systems. IoT applications typically have strict latency requirements. IoT applications also involve a great deal of Machine-to-Machine (M2M) communications. The latency requirements of emerging IoT applications, no doubt, requires a new approach to reduce latency to its barest minimum and make fast and efficient use of “things” data.

### 1.1 Research Question

Can we develop a generic Big Data management approach to reduce the latency of intelligent reaction to actionable events in IoT applications?

### 1.2 Objective of the Research

The aim of this work is to propose and demonstrate a generic, efficient, scalable and robust approach to Big Data management approach in IoT which extracts real-time value from data and demonstrate its operation in an application area. Using existing and emerging computing paradigms, we seek to develop an approach to significantly reduce latency in streaming data from a network of connected devices and thus capture events that trigger actions in real time.

### 1.3 Implication of Research

This work seeks to propose a general latency-reducing approach to IoT data management independent of data source, type or communication protocol. Finding this approach will improve significantly, the speed and responsiveness of current real-time applications and also broaden the applications of IoT to new latency critical domains. The approach will reduce response time of IoT applications and enable them to react fast enough to suit the requirements

of emerging applications. It will also serve as the underlying principle of both open-source and commercial IoT data management applications.

## 1.4 Scope of work

The scope of this work includes the following:

I. To provide a new approach to IoT data management with a view to reduce latency.

II. To implement this approach with software tools.

III. To apply this implementation to a challenging use case.

## 1.5 Organization

An extensive review of literature is contained in the second chapter of this write-up. This includes review of the main concepts, technologies used as well as related work in the research area. The third chapter presents and describes the proposed model, how it works and its latency-reducing advantages. The fourth chapter describes an implementation of the model with results and evaluations and the fifth chapter contains the conclusion and future works on the subject.

# CHAPTER TWO

# LITERATURE REVIEW

## 2.0 Internet of Things (IoT)

The Internet of Things (IoT) has become a popular topic in both industry and academia. The term was originally coined by the British technologist Kevin Ashton in 1999, to describe a system where the internet is connected to the physical world via ubiquitous sensors [37]. It can simply be described as an interconnection of a massive number of objects/sensors/devices (things) through a communication network to provide value-added services [1]. IoT is a concept and a paradigm that considers pervasive presence in the environment of a variety of things/objects that through a communication medium and unique addressing schemes are able to interact with each other and with other things/objects to create new applications/services and reach common goals [26].

*Figure 2.0: Internet of Things (IoT). (Source: ARM)*

"Things", in this context, are anything that can connect to any other thing over a communication medium or to the internet. They typically are recognisable and generate and communicate data to one another. IoT's growth has been rapid over the years as more and more devices are becoming connected and application areas are increasing by the day. With an

estimated figure of nearly 50 billion devices to be connected by 2020, and the advances in data handling capabilities, the possibilities in the IoT world are endless.

### 2.0.1 Why Internet of Things?

The main aim of the IoT concept is to have a smart and interconnected world and to have "things" capture data without help from humans, process the data and make intelligent decisions, all by themselves. This will engineer a new automated world where we can reduce cost and waste [10]. Analysing data from so many sources can give invaluable insights about human behaviour and decision-making patterns, which is very useful in modern marketing and business intelligence. Also, there are environments that are either unsafe or impossible for humans to go into (for example an oil well) and readings need to be taken and communicated to other nodes. IoT fits well into these environments as data can be obtained analysed and instructions can also be communicated to the devices in these environments.

### 2.0.2 Applications of IoT

Constant events monitoring is one main application area of IoT. An example is smart cement, cement equipped with sensors to monitor stresses, cracks and warpages on a bridge continuously for maintenance purposes [27]. Other applications areas include health, where patients' movements, heart rate, blood pressure and all other medical readings can be constantly monitored and actions taken whenever there is a need; transport, to monitor vehicular movements, fuel consumption and operational efficiency; and manufacturing, for safety monitoring and maintenance scheduling [9], [27], [28], [29].

The smart homes, smart grids, smart cars, smart farms and smart cities projects across the globe have been leading the way in IoT application areas [29], [30]. Smart home projects attempt to build applications to control all electronic devices in the home automatically and sometimes remotely. Smart home developers want to achieve a house that can switch off appliances upon detection that no one is using them; devices that can be controlled from work; or a fridge that can create a list of items that are in low supply and send it to a phone [5]. Smart vehicles can sense road conditions, distance from other cars, obstacles, possible collisions, fire, and even more to ensure a safe and automated ride [30]. They can also communicate with safety systems, traffic systems and vehicle maintenance systems.

IoT has also been deployed in supply chain management. Starting from when goods are leaving the factory, they are tracked until they get to the eventual retailer for efficiency, transparency and accountability. Smart grid projects are also concerned with efficient usage of scarce world energy. IoT is deployed to monitor the unsteady supply of renewable energy for electricity generation, balance demand and supply in energy allocation and improve the overall efficiency of the grid. Other applications of IoT include agriculture (smart farms), wearable devices, smart thermostats, oil and gas, and industrial IoT.

### 2.0.3 Challenges of IoT

Security and privacy have been the biggest challenges of IoT [1], [31]. Many connected devices translate to many entry points for malware. Anything connected to the internet stands a risk of being hacked; all the more so in a situation where all or almost all “things” are connected. Also, data is constantly in motion, which increases the risks of its being intercepted or hacked. Privacy is also a concern for individuals. While the acceptability of IoT is getting wider [1],

many people are still not comfortable with sensors that can track their location, health condition, online activities and transaction details. They think some of this information is too sensitive to entrust to the hands of IoT application vendors.

The lack of common standards among the very many disparate connected devices is also a major challenge for IoT applications. The fact that there are so many protocols, APIs and platforms that integrate with other platforms is becoming a major challenge; with more and more platforms coming on line [31]. Some customers also do not adopt IoT because of lack of concrete use case examples and success stories. While some organizations may want to invest based on the theoretical advantages presented, others insist on seeing a working case scenario before committing their resources – a big challenge in getting IoT applied to new application areas.

## 2.1 Big Data

There is no specific definition of Big Data that is agreed upon [34]. It is, however, described with some properties that are widely accepted. These properties are often referred to as the 5Vs of Big Data: volume, variety, velocity, veracity and value.

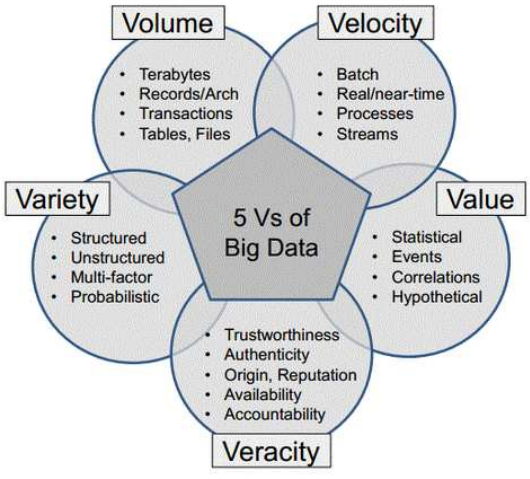


*Figure 2.1: The 5Vs of Big Data. (Source: Arqano)*

As at 2013 90% of the data in the world had been created in the previous two years [32]. Interestingly, also in 2015, 90% of the data in the world has been created in the previous two years [33]. This shows the geometric growth in the volume of data generated due to various digital activities. In 2013, 28,875 GB of data was generated per second. With the current growth rate, this figure is expected to rise by almost 100% (50,000 GB) by 2018. The above statistics just describe one characteristic of the Big Data – volume. The volume of data to be processed is undoubtedly big; much more serious is the fact that it will only keep growing.

Regarding velocity, the rate at which data is being generated is unprecedented. As at 2014, every minute, there were 2.5 million new Facebook posts, 300,000 tweets, 220,000 new photos on Instagram, 72 hours of new videos on YouTube and over 200 million emails [35]. Considering the growth rate in the internet penetration, one can only imagine the number at present. Combining this with data generated by other devices, websites, sensors and all other sources of data, the velocity of data is very high at present and, as the case for volume, it will only grow.

Traditionally, data is usually structured and data sources are not very disparate. Now, terms like “unstructured” or “semi-structured” data are commonplace and they form a very big part of Big Data. About 2.5 quintillion bytes of data are generated in the world per day from unstructured data sources like sensors, social media posts and digital photos [36]. Big Data contains not just the usual text files; it also contains videos, pictures, sensor data, web logs, tweets, emails, Facebook posts and CCTV footage. The disparate data sources and forms in which data are generated explain the variety of attributes of Big Data. The other two attributes

– veracity and value – have to do with processing of Big Data. Veracity describes quality and how understandable data is, while value describes the economic importance of the data.

### 2.1.1 Big Data Management

There is no reason to gather so much data if it will not be of any use. But dealing with large, massively heterogeneous and rapid new datasets is not “business as usual”. Collecting, organizing, storing, securing and extracting useful insights from large volumes of structured and unstructured data is a challenge to both academia and industry. Data management is a broad concept referring to the architectures, practices and procedures for proper management of the data lifecycle needs of a certain system. In the context of IoT, data management should act as a layer between the objects and devices generating the data and the applications accessing the data for analysis purposes and services [14].

The volume of data to be managed has outgrown the storage capacity of traditional data storage and analytics solutions. Also, data is generated in real time and insights from data are also demanded in real time. Traditional data management systems cannot scale up to the velocity of the Big Data. The variety of data sources also necessitates a shift from the traditional data management approaches to a new approach that will be tailored to suit the unique characteristics of Big Data. Traditional data management approaches cannot deal with the heterogeneity and variable nature of Big Data, which comes in formats as different as emails, social media data, videos, images, blogs and sensor data as well as “shadow data” such as access journals and web search histories [38]. Proper Big Data management can empower businesses to automate more business processes, operate near real time, and through analytics, learn valuable new facts about business operations, customers, partners, and so on [39].

According to a survey carried out in [39], the Apache Hadoop Framework tools will be the software products most aggressively adopted for Big Data Management in the next three years (referring to 2014-2017). Apache Hadoop is thought to be the best new approach to unstructured data analytics for now. Hadoop is an open-source framework that uses a simple programming model to enable distributed processing of Big Data on clusters of computers. The complete Hadoop technology stack includes common utilities, HDFS, analytics and data storage platforms and an application layer that manages distributed processing, parallel computation, work flow and configuration management [38].

Other emerging popular tools include complex event processing (like SQLstream and Apama) for streaming Big Data, NoSQL databases (like Cassandra) for schema-free Big Data, in-memory databases (for example VoltDb) for real-time analytic processing of Big Data, private clouds, in-database analytics, and grid computing [39]. Another fast-emerging practice in Big Data management is Fog computing; an emerging computing model that brings data management closer to the edge of the network.

### 2.1.2 Big Data and Internet of Things

A major application of the IoT is in sensing and monitoring. Continuous sensing and monitoring implies rapid and continuous generation of unstructured data. Fifty billion devices are expected to be connected by 2020 [2]. Even if each device generates an average of 1 byte per second, IoT will be generating about 50 GB of data per second by 2020. IoT is, no doubt, playing a significant role in the growth of both the volume, variety and velocity of data. With continuous data streams from “things”, data centres have to handle this additional load of

unstructured, heterogeneous data. This has had effects on Big Data infrastructures as most companies now outsource data storage on cloud services [40].

Automated sensor data is projected to soon overwhelm the data available from more traditional human-centred sources [41], i.e. they will be the main source of Big Data. Dealing with fast ingestion rate of IoT data is a major concern in Big Data collection. Also, the real time requirements of IoT applications have made scalability, distributed processing, and real-time analytics critical functionalities of any Big Data management environment. Usually, Big Data analytics involves batch processing of a large collection of unstructured datasets. But with IoT applications' real-time requirements, Big Data analytics solutions now have to process and act on data in real time or near real time. Data need to be processed in motion and not the store-process-forward approach any more. IoT has in fact pushed all fronts of Big Data management; from data collection, storage and processing of data.

## 2.2 Data Streams

Traditional database usually contains records arranged in columns and rows and are typically not concerned with the time of arrival of these data except when the time stamp is used.

The Database Management Systems (DBMS) for these databases are basically concerned with insertions, deletions, updates and queries. Emerging applications (IoT especially), however, have pushed the limits of the traditional databases. Information naturally occurs in the form of a data streams; examples include sensor data, internet traffic, online auctions, and transaction logs (such as web usage logs and telephone call records). A data stream is a real-time, continuous, ordered (implicitly by arrival time or explicitly by time stamp) sequence of items [18]. A data stream consists of an unbounded sequence of tuples with fixed schemas.

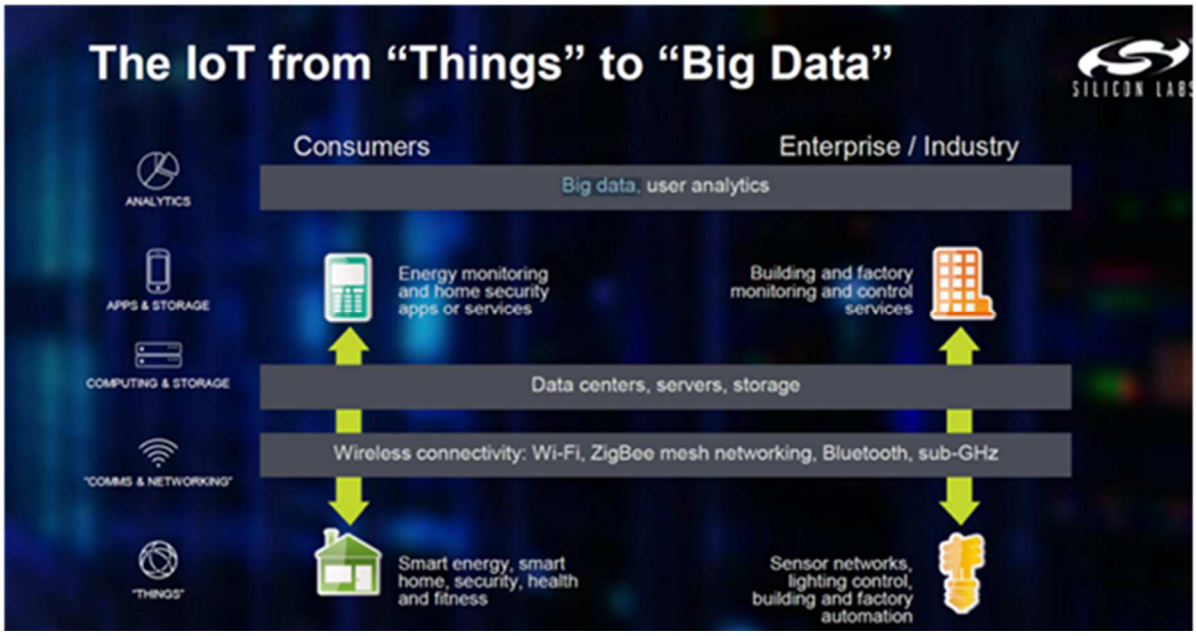


*Figure 2.2: Big Data and IoT (Source: Silicon Labs)*

### 2.2.1 Data Stream Processing

Data streams cannot be handled by the traditional DBMS. This is mainly because of the fact that it is practically impossible to store data streams locally in their entirety and that queries on data streams typically run continuously (stateless) or over a window (stateful). The unique characteristics of data streams and continuous queries necessitates data stream processing solutions to have requirements such as order-based and time-based operations; use of approximate summary structures (synopses), since we cannot store data streams; shared execution for scalability; adaptation to different conditions; and real-time reaction to outlier values [18, 19]. Data stream processing is the in-memory, record-by-record analysis of data streams in motion with the objective of extracting actionable intelligence as streaming analytics, and to react to operational exceptions through real-time alerts and automated actions in order to correct or avert the problem [22]. Modern stream processing engines have high requirements in terms of scalability, fault tolerance and latency. They differ from traditional

stream processing engines mainly in their architecture, (master-slave), data pattern (publish and subscribe) and highly parallel execution [25].

### 2.2.2 Stream Processing Models

Stream processing models process data while still in motion (before and/or without storing). A stream processing system is usually a network of processing units called processing elements (PEs). These PEs receive tuples and perform some computations on these tuples and then produce results in the form of decisions or other tuples. A stream processing engine creates a network of PEs connected in a directed acyclic graph (DAG). The vertices of the graph represent the PEs and the edges represents the tuple flows. Each PE is independent and only communicates with other PEs via messaging. This communication can either be pull-based or push-based messaging [42]. The distribution, load balancing and management of PEs may not be the same in different implementations.

## 2.3 Real-Time Data Stream Processing

Applications based on the traditional store-first, process-second data management architectures are unable to scale for data streams applications largely due to the velocity and volume of data streams. Even Hadoop-based systems are unable to offer the combination of latency and throughput requirements for real-time data stream applications in industries such as e-commerce, telecoms, IoT and financial institutions [22]. Data stream processing therefore needs to meet the real-time requirements of these emerging applications.

Fast-moving data from emerging applications are most valuable at the time they arrive at the data stream processing system [21]. It is not so useful detecting a potential buyer after the user leaves an e-commerce site, or detecting theft after the burglar has absconded [20]. Another

perspective may be a social network that wants to detect trending conversation topics in minutes; a search site may wish to model which users visit a new page; and a service operator may wish to monitor program logs to detect failures in seconds. Hence the need to process and get value from data streams in real time is paramount. The definition of real time, however, depends on the latency constraints of the application.

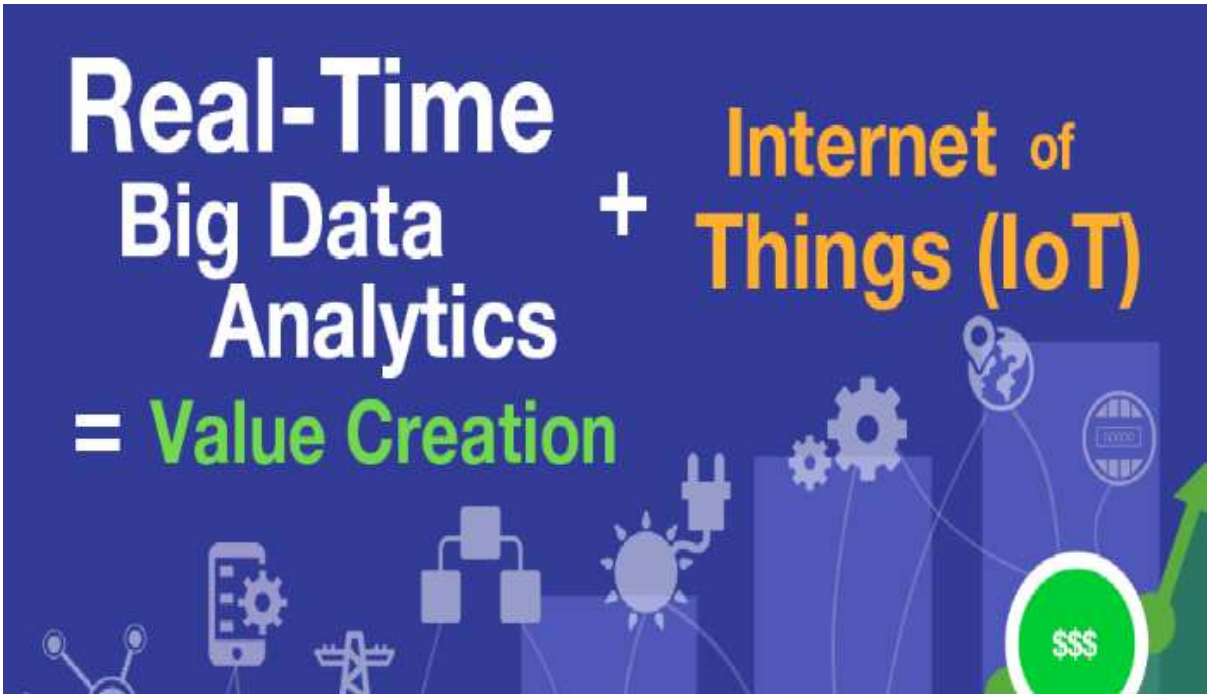


*Figure 2.3: Real time Stream Analytics. (Source: LinkedIn)*

### 2.3.1 Requirements of Real-time Data Stream Processing

Apart from the low latency, high throughput requirements, Stonebraker [23] provided one of the most comprehensive requirement analysis for Real-Time Data Stream Processing. Stonebraker listed eight requirements which encompass most others found in other works. He argued that data must be kept moving (processed on the go without storing first) to achieve low latency. This, he said is the first requirement of real-time stream processing. The second requirement is that the stream processing framework should provide an SQL or SQL-like query language for processing streaming data.

Handling data losses, delays and out-of-order delivery is the third requirement, according to Stonebraker. The stream processing engine should also be predictable and repeated processing over the same data set should give the same results. There should also be an integration between stored data processing and streaming data processing – a mechanism to process both information from the "past" to the "present" at the same time. Data security and availability is also a vital requirement and integrity of data should be guaranteed at all times, even in time of failure. Scalability and distributed processing are also key requirements for stream processing frameworks to handle large and growing data streams and to increase the speed of processing. Lastly, he mentioned real-time response as the last requirement of real-time stream processing. Stream processing should be optimized and done with such low latency that response will be in real time.

## 2.4 Stream Processing Applications

A review of some already existing and popularly used stream processing applications is presented below.

### 2.4.1 Aurora

Started as a project in Brown University, Brandeis University and MIT, Aurora is one of the early stream processing applications. It was designed to handle three broad application types in a single framework; real-time applications, archival applications (interested in the past), and spanning applications (interested in both past and present) [43].

**Architecture**

Aurora's architecture is very close to the generic stream processing model. It also uses a DAG with boxes and vertexes. The boxes represent the PEs and the vertexes represent the tuple flow. The communication model in Aurora is pull based. Tuples arrive at the queues of a box processor and a scheduler selects which boxes to run. Each box is a stream operator specified in the Aurora query model. Data flow between boxes is in the direction of the edges of the DAGs. Aurora, however, was designed as a single-site stream processing engine and not distributed. Features like fault tolerance were also absent. But by combining Aurora with Medusa, another MIT project stressing distributed processing infrastructure for stream processing, distributed processing and even fault tolerance can be achieved.

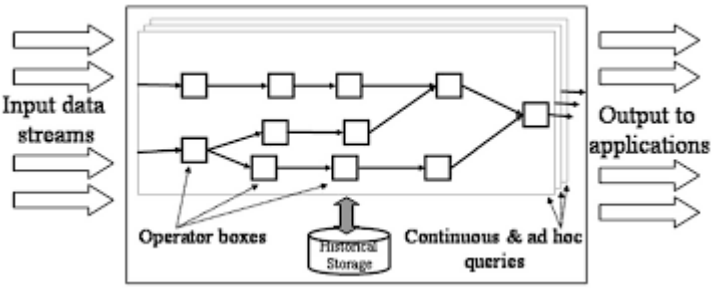


*Figure 2.3: Aurora Architecture. (Source: Abadi et. al.)*

### 2.4.2 Borealis

Sometimes described as an improved version of Aurora, Borealis [44] is a distributed stream processing engine developed by the same collaborating universities that developed Aurora. It sets out to solve some identified problems in Aurora and other existing stream processing applications at the time. It also uses the DAGs network and adds a few advanced functionalities to take care of the identified shortcomings of existing applications at the time [42]. Dynamic load balancing is one of the main issues addressed in Borealis due to the nature of data streams. They sometimes experience bursts and variations over time scales [44]. Other identified and

addressed problems include dynamic query results revision – if there is an update in processed data – and dynamic query modification to edit queries at runtime.

**Architecture**

Borealis architecture is fully distributed. There is no central controller. It has several interconnected servers running at different locations. These servers, since they are connected, take coordinated actions. Each server has a query processor and an admin module that determines where a query should be run locally or remotely. Each query processor has a box processor, like that of Aurora, that runs individual PEs, a load shedder for load balancing operations, a storage manager for storing streams and a priority scheduler that determines execution order [42]. The data communication system in Borealis is also pull based. The Borealis project is no longer active as the last version was released in 2008 [44].

### 2.4.3 Apache Storm

Storm, originally built by Twitter, is an Apache open-source project. It is a distributed real-time computation system for processing large volumes of high-velocity data in parallel, at scale and in real time.

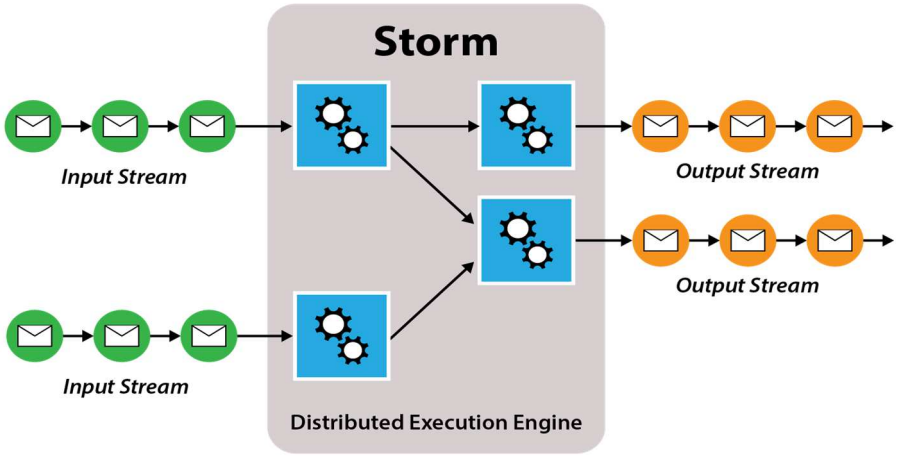


*Figure 2.4: Apache Storm. (Source: Sain Technologies)*

Storm has a robust fault tolerance system and guarantees that data is processed at least once. It is scalable and very fast also; a benchmark clocked it at over a million tuples processed per second per node. Storm is very popular in stream processing and sometimes envisaged as doing to stream processing what Hadoop did to Big Data [46].

**Architecture**

A storm cluster has three sets of nodes: Nimbus nodes, Zookeeper nodes and the Supervisor nodes [42]. Nimbus is the main server where a user program is submitted for execution. Nimbus then distributes this code among worker nodes for execution. A Worker node is composed of workers. Each Worker contains a variable number of Slots. A Slot is an available computation unit which has dedicated CPU resources [46]. Fault tolerance is achieved by keeping track of the progress of Worker nodes so that failed computations can be restarted or relocated to other nodes; this is also done by Nimbus [42].

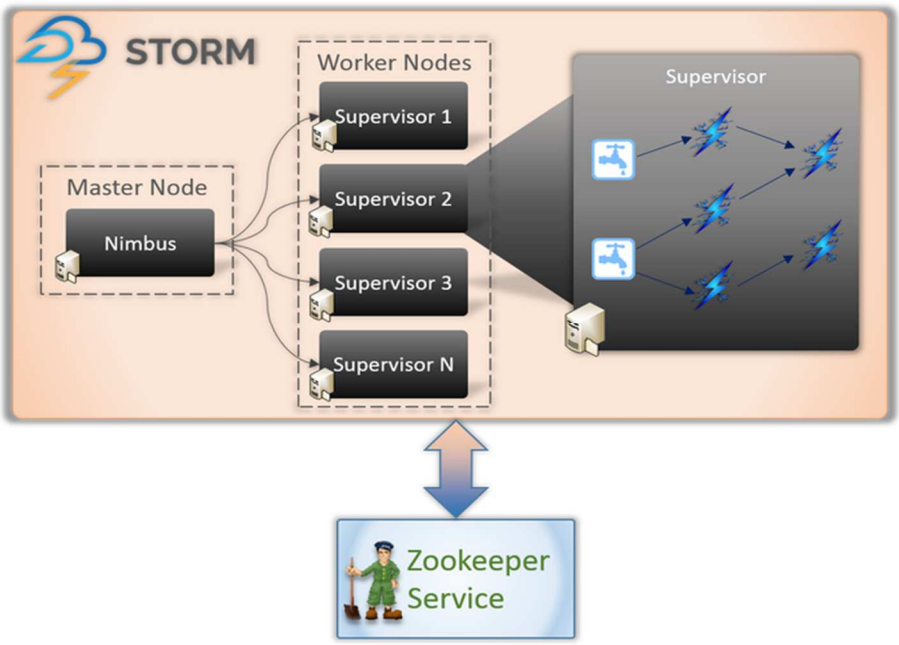


*Figure 2.5: Apache Storm Architecture. (Source: Business-software)*

A Supervisor is a daemon run by the set of worker nodes in a Storm cluster. The coordination between the Supervisor nodes and Nimbus happens through the Zookeeper node. Storm jobs is being represented as user-defined work flows, denoted as topologies. These topologies are also work flows but vertices belong to two main categories: Spouts and Bolts [46]. The Spouts are simply the data entry points. They receive data from one or more data sources and they transmit data to Bolts. Bolts are execution units for user-defined operations considered to be atomic. Storm does not support stateful operators and thus does not support window incremental processing, but they can be added through an API extension [46].

### 2.4.4 Apache S4

Started by Yahoo! and later donated to Apache, S4 is a general-purpose, distributed, scalable, partially fault-tolerant, pluggable platform that allows programmers to easily develop

applications for processing continuous unbounded streams of data [47]. S4 stands for Simple Scalable Streaming System. S4 is fully distributed and performs processing in real time. It employs the Actor model for computations [42]. This processing model is based on the MapReduce concept.

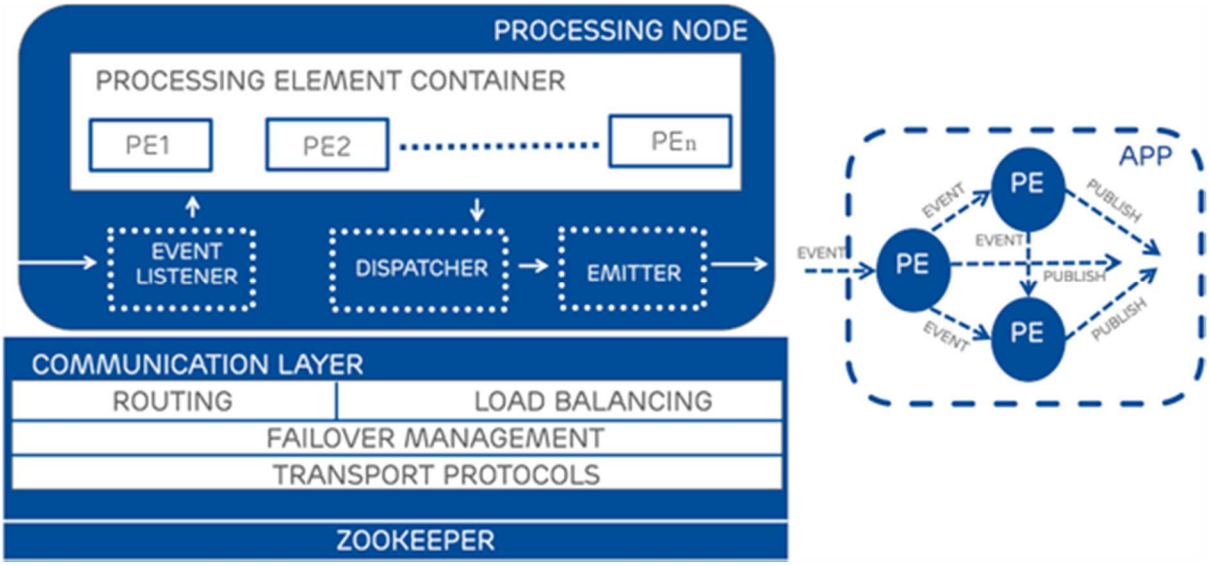


*Figure 2.6: Apache S4 architecture. (Source: S. Kamburugamuv)*

**Architecture**

The basic processing units of S4 are called PEs. These PEs are arranged in a dynamic network with a DAG layout at runtime. A runtime instance of a PE is identified using the PE's code and configuration, key attributes for the events, the events consumed by the PE, and value of the keyed attributes [42].

Each PE instance has its own unique key value attributes except that the PEs take input into the S4 framework, like the Spouts in Storm. Dealing with key attributes of very large value domains can be problematic, however, as it creates an unnecessarily large number of PEs. PEs are hosted by Processing Nodes which also perform the hashing key value attributes.

Coordination between nodes is achieved through the Zookeeper. The “partial” fault tolerance of S4 is also achieved through the Zookeeper. If the Zookeeper detects failure, it distributes the tasks of that PE to other Pes, but there are no message delivery guarantees.

### 2.4.5 Apache Samza

Apache Samza, now also an Apache incubator project, was developed and open sourced by LinkedIn to meet the stream processing needs of the company. Samza’s goal is to provide a lightweight framework that helps in building applications to process continuous streams of unbounded tuples [48]. A Samza stream is an immutable unbounded collection of messages of the same type. Messages can be added to or deleted from streams. A logical collection of processing units that act on a stream and produce output is called a Job in Samza [42]. A Job is the user program that consumes and processes a set of input streams. A Job creates a network topology that processes messages like the runtime DAG in S4. Streams are partitioned into sub-streams and distributed across the parallel processing tasks. A task is the processing element in a Samza network like the PEs of S4. Samza has a good fault tolerance system, providing at least one message delivery guarantee. It also has inbuilt support for stateful streaming.

#### Architecture

Samza’s architecture consists of three components:

1. A streaming layer: responsible for providing partitioned streams that are replicated and durable
2. An Execution layer: responsible for scheduling and coordinating tasks across the machines
3. A processing layer: responsible for processing the input stream and applying transformations [48].

It typically uses Apache Kafka, a distributed messaging application, for the streaming layer. It also uses the Apache YARN for the execution layer while the Samza API sits at the processing layer.

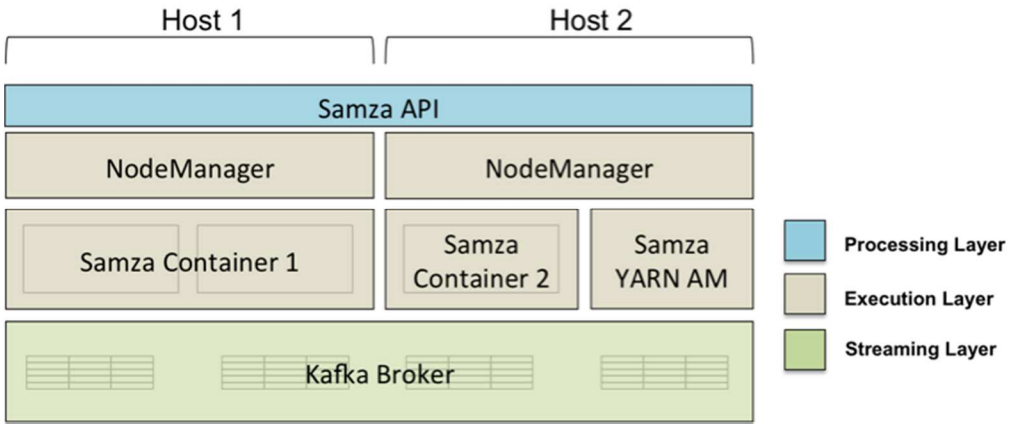


*Figure 2.7: Sample Samza Implementation Architecture. (Source: LinkedIn)*

### 2.4.6 VoltDb

VoltDB is the in-memory NewSQL database built to solve the problem of managing Fast Data (Streams) generated from emerging data sources. Unlike other streaming solutions and data collectors, VoltDB focuses mainly on the requirements of Fast Data with unlimited scalability, the ability to perform real-time data collection, exploration, analysis and taking action all with the transactional consistency (ACID) of traditional relational databases [49]. It also has a good fault tolerance system, recovery system in time of failures and SQL query support.

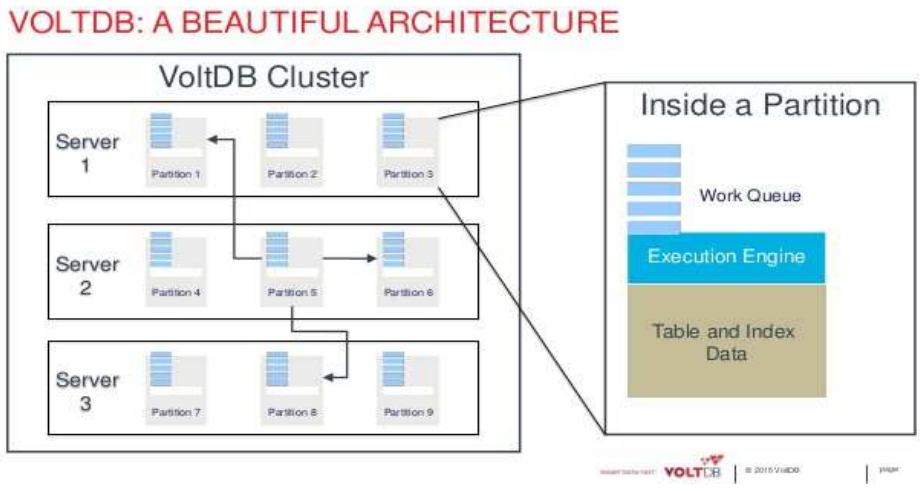


*Figure 2.8: VoltDb Architecture*

## 2.5 Related Work

Much work has been done with respect to IoT data management but, in this section, we focus on the ones related to data, analytics and performance. We briefly summarize some of the related works.

### 2.5.1 Towards Cloud-Based Big Data Analytics for Smart Future Cities

Khan et al. [13] proposed a cloud-based data management and analytics service for a cloud cities application. They proposed an architecture that provides basic components to build necessary functionality for a cloud-based Big Data analytical service for smart cities data. This architecture was implemented in both Hadoop and Apache Spark on the Bristol open data. The results show that Hadoop incurs more overhead, especially in job submission, than Spark and that Spark is more appropriate for the Bristol open data. The part of the open data analysed is the data about quality of life measured by indicators such as crime rate, security, etc. Their

work is an important contribution to the smart cities projects, a very popular application of IoT. Their work, however, does not look into real-time analytics.

### 2.5.2 A Data-Centric Framework for Development and Deployment of Internet of Things Applications in Clouds

Khodadadi et al. [8] proposed another cloud-based data-centric framework for development and deployment of IoT applications. The framework architecture manages data collection, filtering and load balancing from different sources, producing both structured and unstructured data. It was demonstrated with an application to compute tweet sentiments of the six biggest capital cities in Australia. This application was built on top of the Aneka Cloud Application Platform. This framework focuses on data collection and its abstraction from developers and it is an important contribution towards finding a generic approach to IoT data collection management. It, however, does not deal with data analytics.

### 2.5.3 A CIM-Based Framework for Utility Big Data Analytics

Zhu et al. [11] proposed a framework to make disparate and incompatible datasets usable, interoperable and valuable across the enterprise. This framework, though not specifically for IoT, enables data from disparate sources to be more “homogeneous” by proposing a Common Information Model (CIM) standard for information interchange between data sources. This approach will require data sources to use the CIM adapters; a condition many IoT devices may not be willing or able to meet. This approach is not also concerned with performance as it is mainly concerned with making data more homogeneous from the source for easy analytics. It is also specifically designed for the utility industry and not a general framework for Big Data analytics.

### 2.5.4 Data Management for the Internet of Things: Design Primitives and Solution

Abu-Elkheir et al. [14], having investigated the issues of IoT data management, also introduced a model framework for IoT data management. This model provided layered stages of data management. The work proposed six layers of IoT data management including a “things layer”, communication layer, source/data layer, federation layer, query layer and application layer. Their work also does not involve any specific handling of real-time processing and optimization issues. It focused mainly on how data is collected, stored and processed. The model is also focused on batch processing.

### 2.5.5 An Architecture to Support the Collection of Big Data in the Internet of Things

Cecchinel et al. [15] also developed an architecture support for a data collection framework in IoT. This framework handles IoT Big Data issues such as sensor heterogeneity, scalability data and even reconfiguration capability. The use was also used demonstrated on sample projects. The work, however, focuses on the data collection part and does not look into the data processing. Also the framework handles data aggregation at the bridge to make data arrive at the middle-ware in a unified format. Their approach will work fine if the IoT data management application is location specific and data always arrive through the bridge. It may not work when data sources are widely dispersed and data come in directly through the internet without passing the bridge.

### 2.5.6 Lambda Architecture

The Lambda architecture [17] was proposed by Nathan Marz. The architecture is generic for large stream processing, giving the options of real-time processing and batch processing concurrently. In the Lambda architecture, incoming data is dispatched to both the batch layer

and the speed layer for processing. The batch layer manages the master dataset (an immutable, append-only set of raw data), and also pre-computes the batch views. The service layer indexes batch views for low latency, ad-hoc queries and the speed layer deals with recent data and real-time analytics only. Incoming queries can be typically answered by merging the results from batch views and real-time views.

The Lambda architecture received some criticisms, however, especially in terms of its complexity [59]. Some others criticized the redundancy in implementing almost identical processes in both layers. Lambda architecture is also criticized for the one-way data flow and immutability of data. Lambda's shortcomings also include its inability to build responsive, event-oriented applications.

## 2.6 Chapter Summary

The chapter describes the concept of the IoT, Big Data and the intersection between the two. It also discusses Big Data management in IoT. Data streams, and real-time data streams processing were also discussed in detail. The chapter further discusses the requirement for stream processing frameworks and reviews some of the existing frameworks. It finally looked at the works of other authors related to this work.

# CHAPTER THREE

# ANALYSIS

## 3.0 Latency

The question of how real is “real time” is often answered by how long getting a response can take before data become obsolete, start depreciating in value or even become useless. Latency refers to the time to get a computation/communication request satisfied. Examples are the time to confirm that a debit card transaction is not fraudulent or the time to recommend another article to a user on a website. Detecting a fraud after the transaction has been completed is of no use, and also taking 50 seconds to confirm that a transaction is not fraudulent will put off most customers. Also, detecting what a visitor to a website might like to read after he/she might have left the website is useless and taking too much time to load a page because of that can also discourage customers from visiting. There are more critical systems like stock trading where milliseconds latency improvements can mean gains in millions of Dollars for companies. Latency requirements of IoT systems are typically strict. Constant monitoring systems especially need to react to actionable events at near zero latency.

### 3.0.1 Types of Latency

Hackathorn [59] identified three types of latency:

- Data latency: the time taken to collect and store the data
- Analysis latency: the time taken to analyse the data and turn it into actionable information
- Action latency: the time taken to react to the information and take action.

Streaming in itself has reduced data latency to a great extent since data are always in motion and not stored before processing. Data are no longer stored before processing; rather they are processed while data are still in motion. Improvements in communication technologies such as the LTE have also contributed to the reduction of data latency. Distributed stream processing engines can now process millions of tuples per second.

There is, of course, still the need for improvement in analysis and data latency and they are active areas in research. Of the three identified latency types, action latency is most critical to IoT as “things” most of the time do not only send data but also do need to receive feedback to act on the information taken. Reducing action latency, however, requires developing a new approach to compute and communicate feedbacks to “things” at near zero latency.

## 3.1 Fog Computing

One can think of cloud computing as a manufacturing company (an application) where all customers have to buy goods (access services) from the factory (cloud) alone. Fog computing would be the company deciding to open retail outlets at locations close to the customers and the retail stores will give a level of access to the customer; if not exactly, it will be close to what they can obtain from the factory. The Fog computing paradigm extends cloud computing and services to the edge of the network; close to the devices. It is a hierarchical distributed platform for service delivery providing computation, communication and storage at a layer that is much closer to the devices than the cloud.

Fog computing is described as a highly virtual platform that provides compute, storage and networking services between end devices and traditional cloud computing data centres, typically, but not exclusively located at the edge of network [62]. Fog computing is not meant

to replace or cannibalize cloud computing; rather, it is to complement it in applications where the traditional cloud computing may not be suitable. Examples of such applications include geo-distributed applications, large-scale distributed control systems, and applications with very strict latency requirements [60]. It is also not a trivial extension of cloud computing. It has some distinctive characteristics that make it more than just an extension.

These characteristics according to Bonomi [62] include:

- **Edge location, location awareness and low latency:** The idea of supporting endpoints with rich services at the edge of the network is said to be one of the origins of Fog computing. This also has to do with applications with strict latency requirement.
- **Geographical distribution:** Unlike the more centralized cloud, the services and applications targeted by the Fog demand widely distributed deployments. This is a key feature in Fog computing.
- **Very large number of nodes:** this is as a consequence of the wide geo-distribution. Applications such as smart connected vehicles and traffic system and smart grids would typically require a very large number of nodes.
- **Support for mobility:** It is essential for many Fog applications to communicate directly with mobile devices (phones or wearable devices), and therefore support mobility techniques.
- **Real-time interactions:** Fog applications involve real-time interactions rather than batch processing. In fact, one of the underlying reasons of the Fog approach is real time interactions.
- **Predominance of wireless access:** Access is predominantly wireless over a network and may or may not need internet communication.

- **Heterogeneity:** Fog nodes come in different forms, and will be deployed in a wide variety of environments.
- **Interoperability and federation:** Fog components must be able to interoperate, and services must be federated across domains.
- **Support for online analytic and interplay with the Cloud:** The Fog is positioned to play a significant role in the ingestion and processing of the data close to the source.

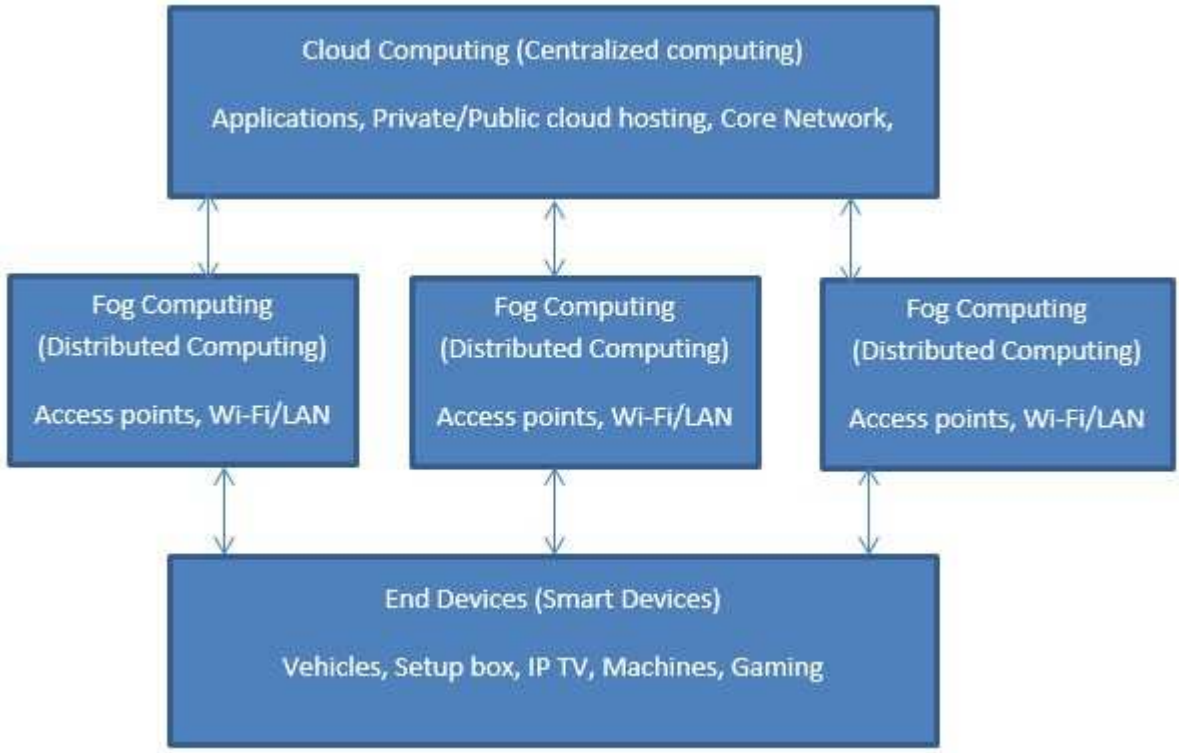


*Figure 3.0: Fog Computing*

### 3.2 Multi-Tier Fog-Cloud Architecture

Multi-tier Fog-Cloud Architecture was proposed by John et al. in [61]. It is a product of superimposing Fog computing on cloud computing resulting in a three-tier distributed computing architecture with the "Fog Servers" serving as the near-end computing proxies between the front-end devices and the far-end servers. The first tier of the architecture, called the front-end basically consist of "things"; devices generating data and transmitting data. The

second tier, called the near-end servers, consist of the Fog Servers These Fog Servers have some storage and processing capabilities which front-end devices can access while incurring minimal amount of communication latency. The third tier consists of the far-end servers. These are cloud infrastructures for batch and near real time analytics and Big Data management. The main objective of the architecture is to optimize communication and computation efficiency; which makes it suitable in meeting the IoT latency requirements.

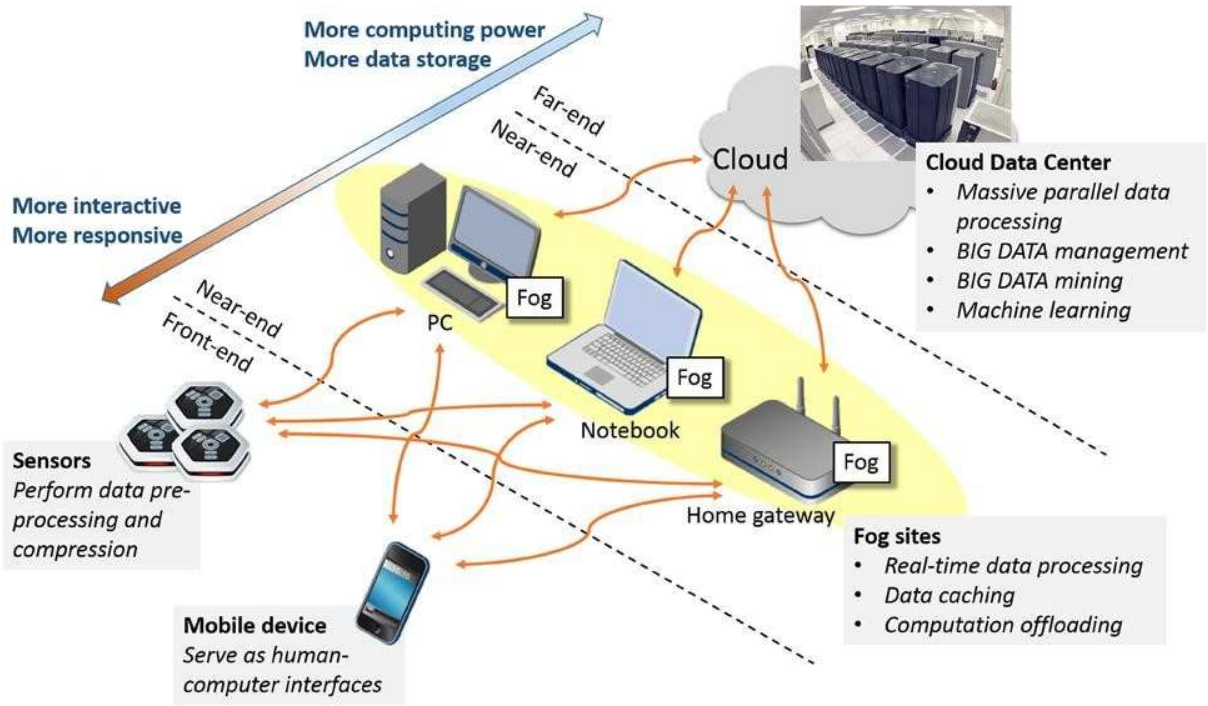


*Figure 3.1: Multi-Tier Cloud-Fog Architecture*

## 3.3 Analysis of the Existing/Traditional Approach

### 3.3.1 From Devices to Cloud

The early approach to managing IoT data is for the device to send data directly to the cloud via a communication medium, in real time. Big Data analytics are also done at the cloud layer and decisions, where applicable, are communicated to the devices from the cloud to the devices via the communication medium also. IoT applications are typically highly distributed and the

velocity of data is high. This can cause congestion on the network and communicating actionable events and decisions can be a challenge. An actionable event can be caught in a busy channel and can be delayed with other messages. This is a major contributing factor to action latency. Also, even when the channel is free, the distance between the cloud layer and the devices is usually too wide. Processing actionable events at a layer closer to the devices was suggested and that introduced the Fog concept to IoT data management.

The Fog layer enables distributed processing at the edge of the network; very close to the devices.

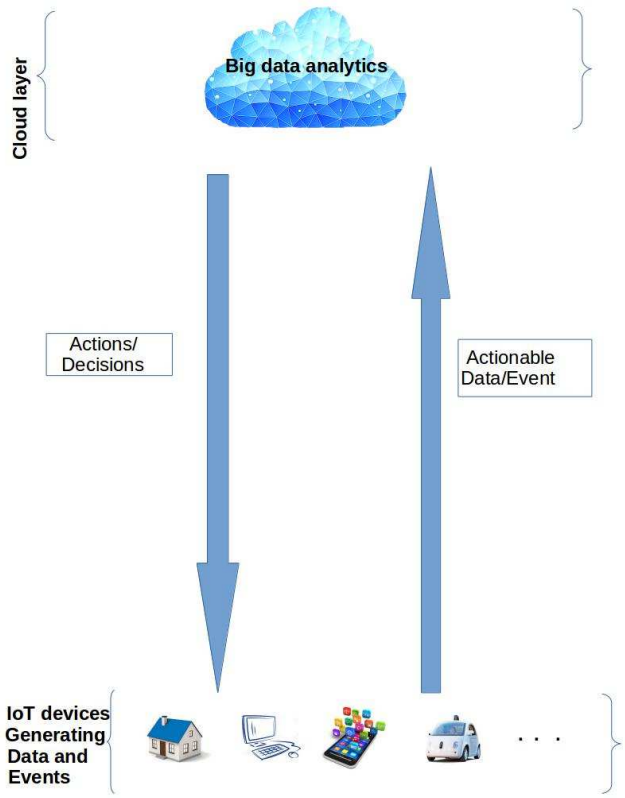


*Figure 3.2: Existing Approach from Cloud to Device*

### 3.3.2 Existing Fog Approach

The Fog approach reduced latency significantly and brought processing close to the edge of the network. It is also sometimes used for dynamic real-time load balancing. Processing of geo-sensitive data is also easier and data is aggregated before transmission to the cloud layer for further processing. However, the existing Fog approach does not provide a mechanism for making intelligent decisions in real time while taking mined intelligence from Big Data analytics into consideration. Decisions are typically based on the actionable events alone and/or the real-time state of the system.

## 3.4 The Proposed Latency-Reducing Intelli-Fog Approach

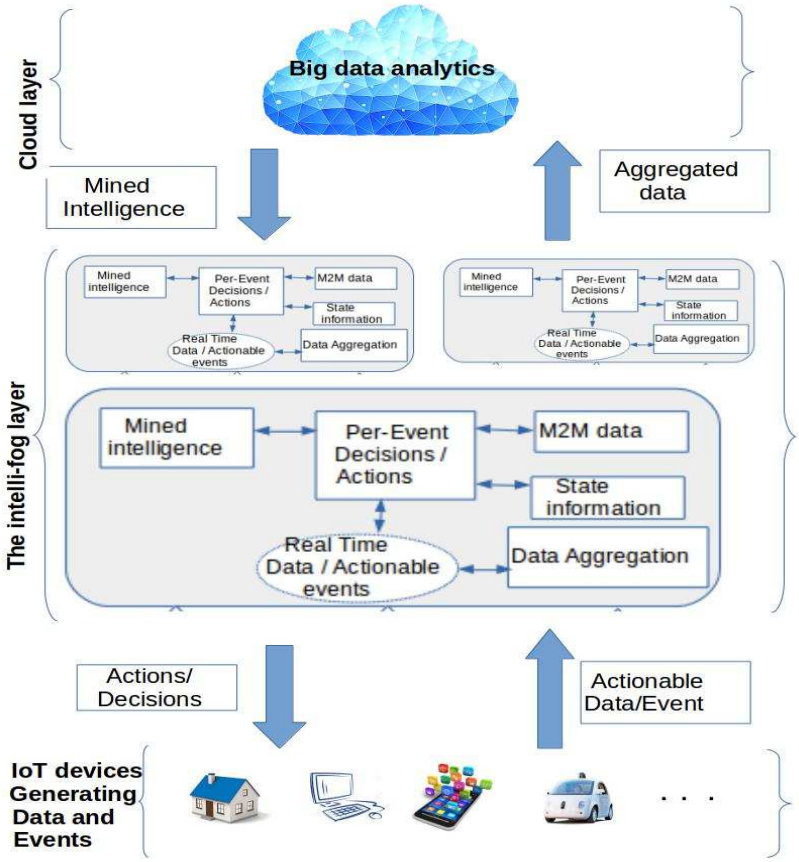


*Figure 3.3: The Proposed Latency Reducing Approach*

### 3.4.1 The intelli-Fog Layer

The intelli-Fog Layer of the proposed approach differs from the existing Fog approaches in that it is capable of making more intelligent decisions; without any significant latency increase. This layer communicates both with the IoT devices as well as the cloud layer where data analytics is performed. Communication with devices involves taking in raw data/actionable events that are observed or reported by IoT devices. It also involves communicating decisions/actions taken at on it (the cloud layer) back to the IoT devices. This is done in parallel with communication with the cloud layer. The communication with the cloud layer is also a two-way communication. Aggregated data are sent to the cloud for storage and analytics and mined intelligence from these historical data are sent back to the intelli-Fog layer. Other information necessary for decision making can also be sent to the intelli-Fog layer.

### 3.4.2 The Cloud Layer

The cloud layer is not too different from the traditional cloud layer. Data storage, and analysis still take place in the cloud layer. The only difference is that decisions making has been moved from the cloud layer to the intelli-Fog layer and that data are sent to the cloud layer after aggregation. Aggregation reduces the number of data to be transmitted and stored and provides a sort of filter to ensure that only useful data is transmitted and stored.

## 3.5 Chapter Summary

This chapter contains review of latency and the different IoT data management architecture approaches. The existing system and its benefits as well as the problems associated with it were

discussed. It contains a description and analysis of the existing approaches as well as a full detailed analysis of the proposed approach in this work.

# CHAPTER FOUR

# USE CASES AND IMPLEMENTATION

## 4.0 Introduction

In this chapter, some use case scenarios of the proposed approach are presented. One of the use cases is also implemented. The use case, tools and programming languages used are discussed. The implementation and screen-shots from the latency benchmark readings are also presented.

## 4.1 Use Case Scenarios

In this section, we present some use case scenarios of the new approach proposed in the previous chapter.

### 4.1.1 Intelligent Patient Monitoring System.

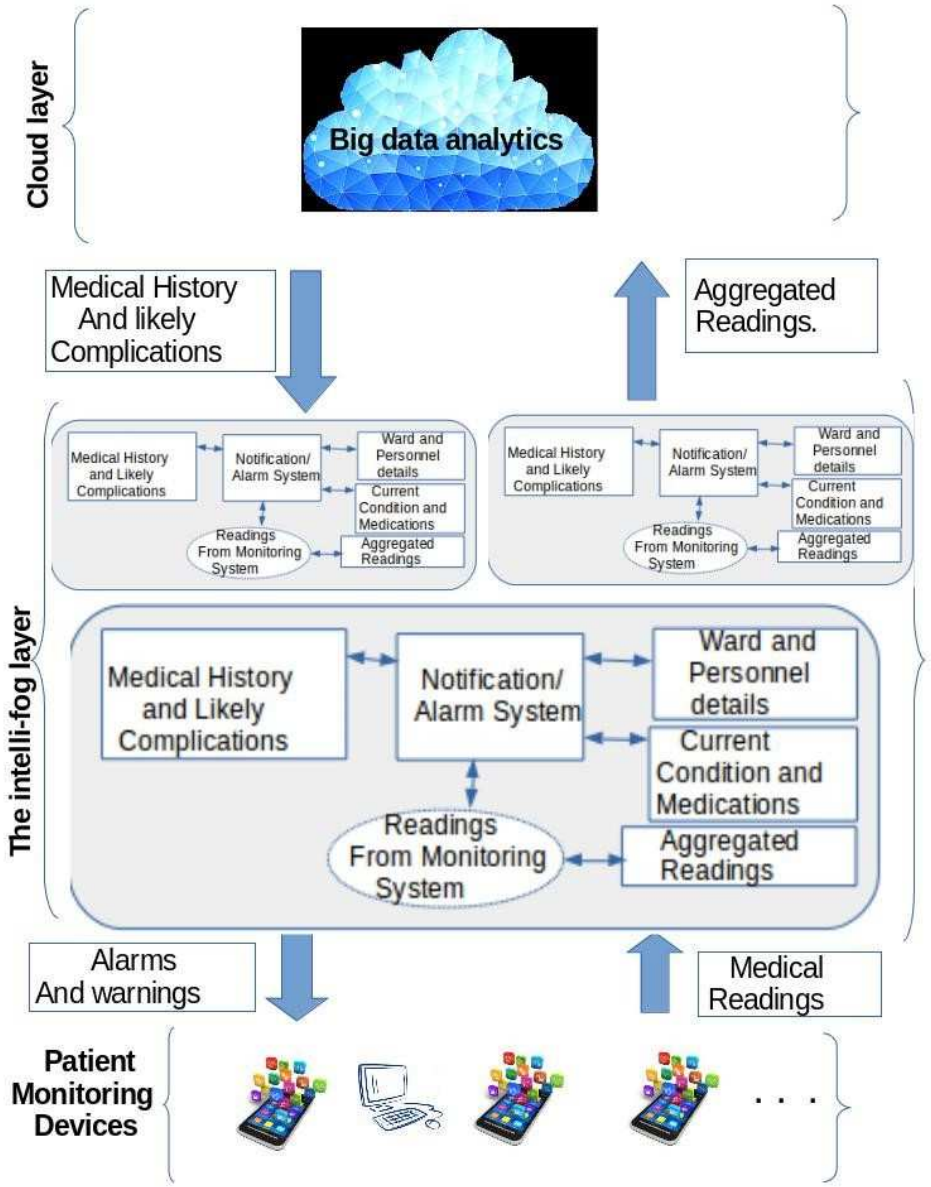


*Figure 4.0: Intelligent Patient Monitoring System*

A patient monitoring machine will sound an alarm by sensing an abnormal reading on patient data, but the alarm cannot state the likely problem that the health care officer should look out for immediately. With the proposed system, a patient monitoring system can be made more intelligent to make these decisions based on some Big Data analysis on a patient’s medical history as well as other available relevant medical records to predict likely complications to watch out for. An intelli-Fog layer can also go further to communicate this reading to the patient's doctor or a doctor on duty; and communicate the patient's necessary medical record too for speedy, accurate and efficient health care service.

### 4.1.2 Smart Cities (Intelligent Traffic Light)

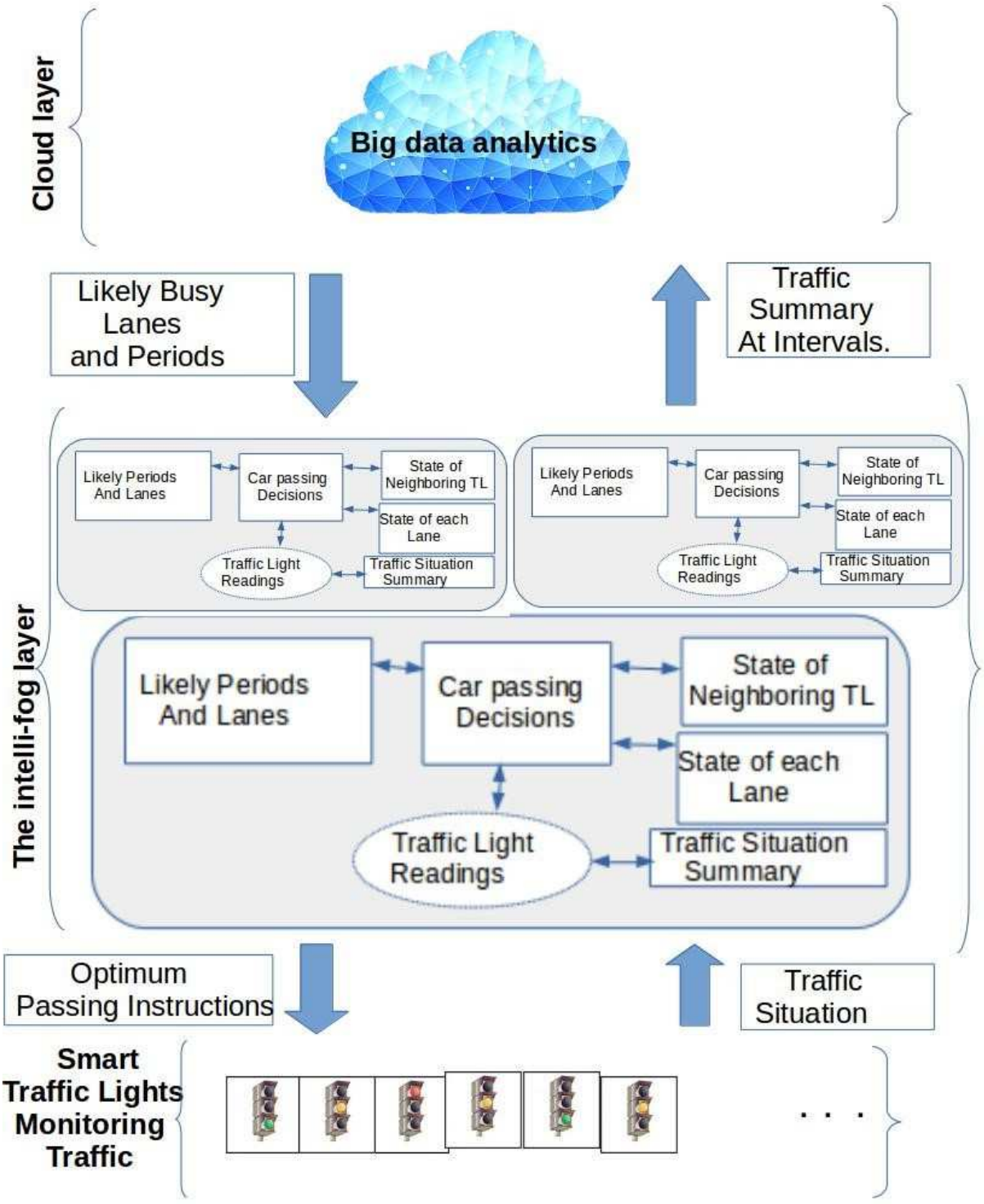


*Figure 4.1: Intelligent Traffic Light System*

In this use case, smart traffic lights serve as the intelli-Fog layer of a traffic management system. The smart traffic light senses both vehicles and pedestrians on the road. It monitors the state of each lane and uses the M2M communication protocol with other smart traffic lights to get the state of neighbouring devices. It aggregates these data and then sends them to a central cloud layer for data analytics. The cloud layer sends back information on likely busy lanes and periods and suggestions on how to pass cars in each lane. This information, together with the state of other lanes is used by a smart traffic light to pass cars on its lane.

### 4.1.3 Geo-Location and User Interest Based Mobile Advertisement display

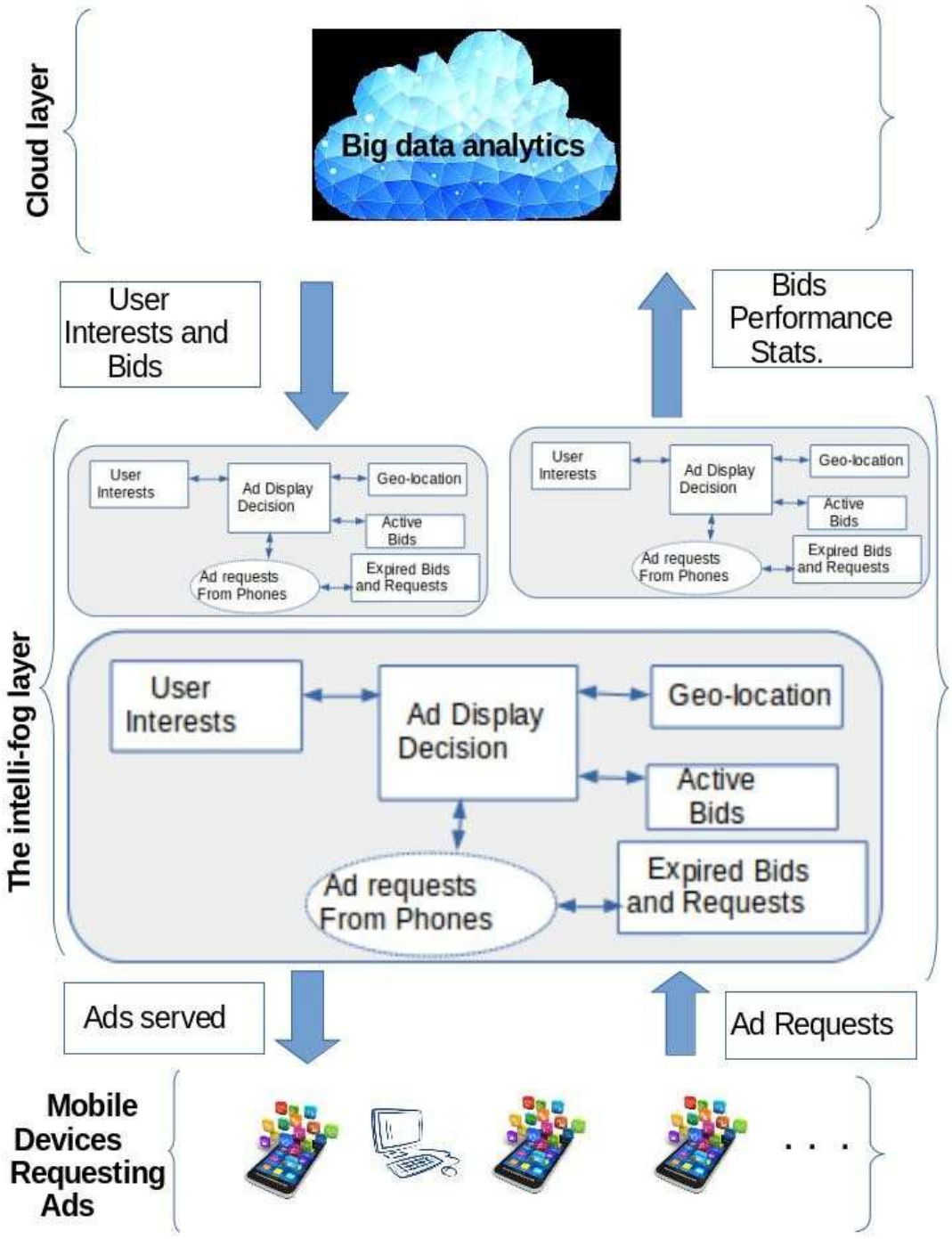


*Figure 4.2: Geo-Location and User Interest Based Mobile Advertisement display*

A geo-targeted advertisement display system on a web browser or mobile phone can determine which advertisement to serve based on the highest bidder and location in real time. Digital marketing these days seek better efficiency in terms of serving advertisements that are relevant to the user. So, adding user interests, based on users' activities on their devices and/or the web pages they visit, will enable a more efficient digital advertising. The intelli-Fog approach can help achieve this with the web browser/mobile phone serving as the intelli-Fog layer. Here, data of user activities are aggregated and sent to the cloud Big Data analytics system. The cloud layer, in turn, returns user interest categories. This information, together with user location information and price of advertisement bids, are then used in serving advertisements.

## 4.2 Use Case Implementation (Intelli-Fog Advertisement Display Based on Location and User Interest)

The intelli-Fog layer of the third use case scenario in the previous section was implemented to demonstrate the proposed approach. The implementation simulates how advertisements are served on mobile phones in real time based on customer location. There are several threads simulating advertisement bids, advertisement requests, and the benchmark threads monitoring and reporting performance of the system in terms of latency. In this program, advertisements are displayed based on location, the highest bidder and also user interests. The user interest, although hard-coded in this program, is assumed to be obtained from data analytics on user activities on the web. Also, expired bids are deleted from the VoltDB memory in the geospatial app. In this implementation expired advertisement bids are exported to a file where they will be imported into Hortonworks Hadoop for analytics so advertisers can have an insight into how their bids are faring.

### 4. 2.1 Technology and Tools

**VoltDb**

VoltDB is the in-memory NewSQL database built to solve the problem of managing fast data (streams) generated from emerging data sources. VoltDB focuses mainly on the requirements of fast data with unlimited scalability, the ability to perform real-time data collection, exploration, analysis and taking action all with the transactional consistency (ACID) of traditional relational databases [49]. The choice of VoltDb for demonstration of this approach is mainly because of its superior latency claims in terms of performance; even above popular streaming frameworks like Storm. While most of these claims can be traced to authors from the VoltDB community themselves, the choice was made due to the absence of counter-claims in literature about VoltDB's performance. Also VoltDB allows per-event decision making and its in-memory feature removes completely disk read/write latency.

**Java**

Java is a powerful computer programming language that is fun for novices and appropriate for experienced programmers to use in building substantial information systems. It emphasizes achieving program clarity through the proven techniques of object-oriented programming. This application is based on Oracle's Java 2 Platform, Standard Edition (J2SE). Oracle provides an implementation of this platform, called the J2SE Development Kit (JDK) that includes the minimum set of tools needed to write software in Java.

**Hortonworks Data Platform**

The Hortomworks data platform is an open-source data management package that contains most of Apache's open-source Big Data and stream analytics software. It runs on a virtual system (Virtual Box or VMWare) using the Ambari server.

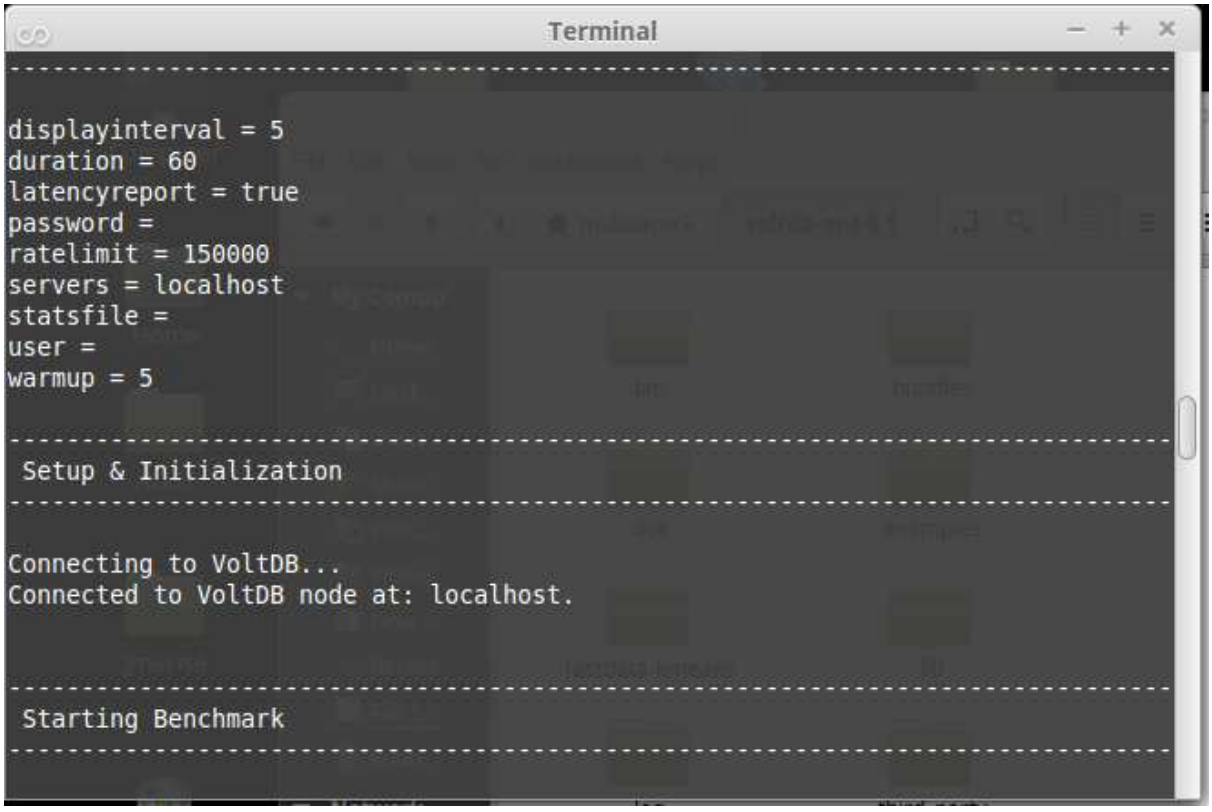


*Figure 4.3: VoltDb Server initialization and setup*

### 4.2.2 The Developed Advertisement Display Simulation

The advertisement display simulation is a modification of an inbuilt VoltDb "geospatial" app. This app was adapted to the intelli-Fog layer for the Advertisement Display model. The major changes included were the database structure of the advertisers table. Each advertiser row was modified to include a category from a range of categories representing user interests. Accordingly, advertisement bids and advertisement requests are generated with categories. These categories are used, together with geo-location data, to serve advertisements.

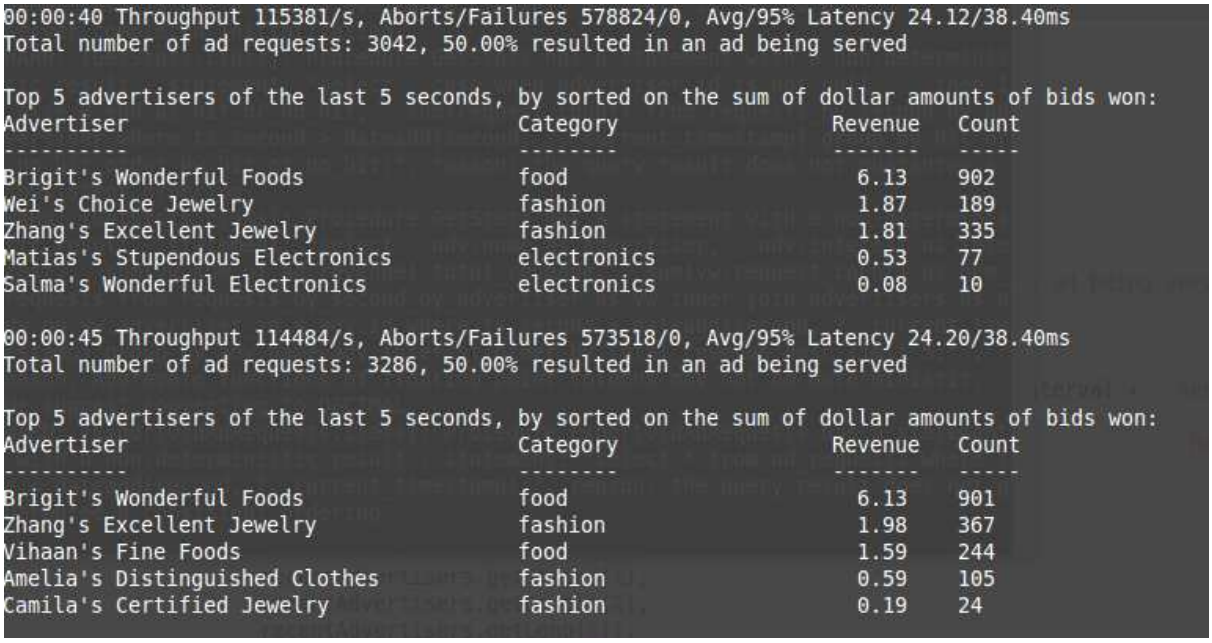


*Figure 4.4: Top five Advertisers in the last five seconds*

The application initializes by loading configuration settings in the AdBrokerBenchmark.java file. These include information like duration of running, display interval, server name, the time when a bid will expire etc. The application then connects to VoltDB in-memory database at this point. Once the initialization is successful, A Java class generates advertisement requests and another Java class generates advertisement bids. Advertisement bids and requests are processed at a throughput of about an average of 110,000 transactions per second.

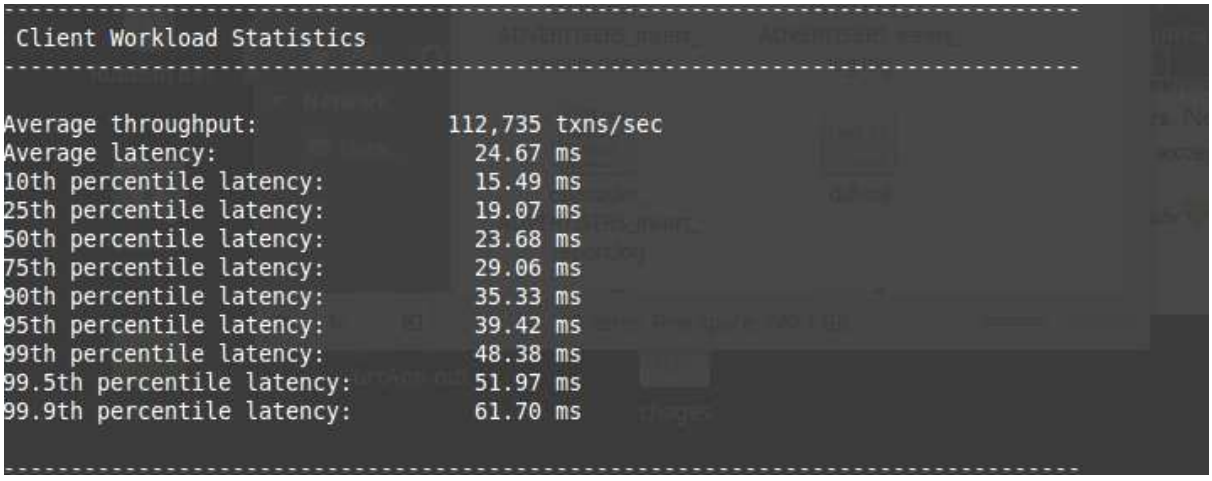


*Figure 4.5: Client Workload Statistics showing very low latency and high throughput*

At intervals of five seconds, the app displays the top five advertisers, their category as well as revenue for the time interval. In the example above, “Bright’s Wonderful Foods” in the food category is the top advertiser for a consecutive five seconds interval gaining $6.03 and $2.58 consecutively.

After the advertisement bids and advertisement requests have stopped, the latency benchmark is evaluated. It gives the client’s workload statistics in terms of throughput and latency


*Figure 4.6: Server also reports low latency*

In this session, the average throughput was at about 112,735 transactions per second and the average latency was 24.67ms. The benchmark also gives the latency breakdown of some stages of execution. Percentile latency is reported at $10^{th}$, $20^{th}$, up to $99.9^{th}$ percentile. The system server also reports an average latency of 6.94ms.

From the implementation, the average latency of running about 112,000 transactions per second is 24.67 ms. This implies that more relevant geo-location based advertisements can be served to users in real time with a latency of 24.67 ms which cannot be noticed by the user. The additional latency of analysing the user’s interest and communicating advertisement requests to the cloud before advertisements are served have both been eliminated. These communication and computation latency can add significant latency which will disrupt the real-time experience of the user. Since the intelli-Fog layer is distributed, it is scalable and latency will not increase with more devices joining the advertisement display system.

## 4.4 Chapter Summary

The chapter contains discussions on use case scenarios of the proposed model in this work. It also contains discussion on the implementation of one of the use case scenarios. The use case implemented was the advertisement display scenario where advertisements are served to users based on location and the user interests. All expired advertisements are also exported back to the cloud layer for (Big) data analytics and intelligence mining. The implementation and screen-shots of latency benchmarks were also included in the chapter.

# CHAPTER FIVE

# SUMMARY, CONCLUSION AND RECOMMENDATION

## 5.0 Summary

This work focuses on the reducing the time it takes to react to actionable events in IoT applications. It investigates how real-time value can be obtained from the Big Data typically emitted by IoT application. The work presents and extensive literature review of IoT and Big Data, as well as the relationship between the two concepts. It also discussed data streams and existing stream processing frameworks. The work also looked into related work by other authors as well as the existing approaches in reacting to actionable events from IoT applications. This work finally presents a new latency-reducing approach to processing Big Data and actionable events in IoT applications; illustrates use case scenarios and also implements a particular use case.

## 5.1 Conclusion

A new latency-reducing approach of processing Big Data in a way to react to actionable events under strict latency requirements has been proposed in this work. Fog computing paradigm, which has been widely advocated for IoT applications, is leveraged to reduce action latency in IoT applications. The work proposed an “intelli-Fog” layer which caches mined intelligence from Big Data and makes it available for decision making at the Fog layer. This makes intelligent response to actionable events faster and closer to the devices. Immediate decision making and Big Data analytics can also be taking place concurrently in this new approach with just a single data entry point unlike the Lambda architecture.

## 5.2 Recommendations

The proposed approach is still very new and had only been tested on just one of the use cases presented in this work. Future work should look into testing the approach by implementing in use cases and comparing the benchmark with existing approaches. While testing the approach against existing approaches, however, careful attention should be paid to the environments to make user the programs run on the same hardware and using the same communication facilities.

Also this work investigates the architectural part of IoT Big Data management. A second phase of the work should focus on the Big Data processing part. Data ingestion, message queues and stream processing methods there are most suitable for IoT or can be optimized for IoT applications should be investigated.

## References


[1]. Perera C., Ranjan R., Wang L., Khan S., and Zomaya y., “Big Data Privacy in the Internet of Things Era” Issue No.03 - May-June (2015 vol.17) pp: 32-39

[2] C. Perera, A. Zaslavsky, P. Christen and D. Georgakopoulos, “Context Aware Computing for The Internet of Things: A Survey,” Communications Surveys Tutorials, IEEE, vol. 16, no. 1, pp. 414- 454, 2013

[3] C. Eaton, D. Deroos, T. Deutsch, G. Lapis and P. Zikopoulos, “Understanding Big Data” , McGraw- Hill Companies, 2012.

[4] Mohak Shah “Big Data and the Internet of Things” Research and Technology Center - North America, Palo Alto, 2015.

[5] Cramer I., “Big Data” Business Technology magazine, 2.2012 Issue 9.

[6] B. Baesens, “Analytics in a Big Data World: The Essential Guide to Data Science and Applications.” John Wiley & Sons, 2014.

[7] Ortiz, A.M., Hussein, D. ; Soochang Park ; Han, S.N. ; Crespi, N., “The Cluster Between Internet of Things and Social Networks: Review and Research Challenges”. IEEE IoT Journal, Vol 1, Issue 3, 2015.

[8] Khodadadi, F. ; Calheiros, R.N. ; Buyya, R. “A data-centric framework for development and deployment of Internet of Things applications in clouds ” (ISSNIP) 2015 IEEE Tenth International Conference, 2015

[9] Mike Barlow “Real-Time Big Data Analytics: Emerging Architecture” O’Reilly Media, ISBN: 978-1-449-36421-2 2013

[10] Kevin Ashton, That 'Internet of Things' Thing, RFID Journal, July 22, 2009.

[11]. Zhu J., Baranowski J., Shen J., “A CIM-Based Framework for Utility Big Data Analytics” (2014) available at: http://www.powerinfo.us/publications/CIM_Based_Framework_for_Big_Data_Analytics.pdf

[12] Yuri Demchenko “Defining the Big Data Architecture Framework (BDAF)”, University of Amsterdam, Amsterdam, 2013.

[13] Khan Z., Anjum A., Soomro K., and Tahir M., “Towards cloud based Big Data analytics for smart future cities” Journal of Cloud Computing: Advances, Systems and Applications, 2015.

[14] Mervat Abu-Elkheir 1, Mohammad Hayajneh and Najah Abu Ali, “Data Management for the Internet of Things: Design Primitives and Solution” sensors ISSN 1424-8220, 2013.

[15] Cecchinel C., Jimenez M., Mosser S., Riveill M., “An Architecture to Support the Collection of Big Data in the Internet of Things” IEEE International Workshop on Ubiquitous Mobile Cloud, 2014.

[16]. Rhodes P., “Build an IoT analytics solution with Big Data tools” InfoWorld, 2015 available at:

http://www.infoworld.com/article/2876247/application-development/building-an-iot-analytics-solution-with-big-data-tools.html

[17] http://lambda-architecture.net/

[18] Lukasz Golab and M. Tamer Ozsu, “Issues with Data Stream Management” , ACM, Volume 32 Issue 2, June 2003.

[19] Arvind Arasu , Brian Babcock , Shivnath Babu , John Cieslewicz , Keith Ito , Rajeev Motwani , Utkarsh Srivastava , Jennifer Widom Stream: The stanford data stream management system (2004),Available at: http://ilpubs.stanford.edu:8090/583/1/2003-21.pdf.

[20] Patricio Córdova, “Analysis of Real Time Stream Processing Systems Considering Latency”, University of Toronto, 2015.

[21]Matei Zaharia et al., "Discretized Streams: Fault-Tolerant Streaming Computation at Scale," SOSP '13 Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, pp. 423-438, 2013.

[22] sqlstream “Stream Processing Eplained” 2015, Available at: http://www.sqlstream.com/stream-processing/

[23] Micheal Stonebraker et al., “The 8 Requirements of Real-Time Stream Processing”, ACM SIGMOD, Volume 34 Issue 4, 2005 .

[24] Kai Wahner, “Real-Time Stream Processing as Game Changer in a Big Data World with Hadoop and Data Warehouse” 2014. Available at: http://www.infoq.com/articles/stream-processing-hadoop

[25] Bjorn Lohrmann et al., “Elastic Stream Processing with Latency Guarantees”, Technische Universitat, Berlin, 2015.

[26] Ovidiu Vermesan, Peter Friess, “Internet of Things – From Research and Innovation to Market Deployment”, River Publishers, 2014.

[27] Daniel Burrus, “The Internet of Things Is Far Bigger Than Anyone Realizes”, 2014, Available at: http://www.wired.com/insights/2014/11/the-internet-of-things-bigger/ .

[28] Tibco Software, “ Streaming Analytics and the Internet of Things: Transportation and Logistics”, 2015, Available at: http://www.tibco.com/assets/blt2251ea123c9b2780/wp-streaming-analytics-and-iot-transportation-logistics.pdf .

[29] Knud Lasse Leuth, “The 10 most popular Internet of Things applications right now ” 2015, Available at: http://iot-analytics.com/10-internet-of-things-applications/ .

[30] Jose’ San “The Smart and Connected Vehicle and the Internet of Things” WSWC, 2013. Available at: http://tf.nist.gov/seminars/WSTS/PDFs/1-0_Cisco_FBonomi_ConnectedVehicles.pdf

[31] Chris Kocher, “The Internet of Things: Challenges and Opportunities” 2014, Available at:

http://sandhill.com/article/the-internet-of-things-challenges-and-opportunities/

[32] Bart Baesens, “Analytics in a Big Data World”, Online ISBN: 9781119204183, 2014.

[33] Ben Walker, “Every Day Big Data Statistics – 2.5 Quintillion Bytes of Data Created Daily ”, 2015. Available at: http://www.vcloudnews.com/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/

[34] A. Zaslavsky, C. Perera and D. Georgakopoulos, “Sensing as a Service and Big Data,” in

International Conference on Advances in Cloud Computing (ACC-2012), Bangalore, India, 2012

[35] Susan Gunelius, “The Data Explosion in 2014 Minute by Minute – Infographic”, 2014. Available at: http://aci.info/2014/07/12/the-data-explosion-in-2014-minute-by-minute-infographic/

[36] Digital Reasoning, “Unstructured data: A big deal in Big Data” 2015. Available at: http://www.digitalreasoning.com/resources/Holistic-Analytics.pdf

[37] Jim Kaskade, “IoT is the new Big Data?” , 2014. Available at: http://jameskaskade.com/?p=3094

[38] T.K.Das, P.Mohan Kumar, “Big Data Analytics: A Framework for Unstructured Data Analysis” International Journal of Engineering and Technology (IJET), ISSN : 0975-4024

Vol 5 No 1 Feb-Mar 2013.

[39], Philip Russom, “Managing Big Data”, TDWI Best Practices Report, 2013. Available at: https://www.sas.com/content/dam/SAS/en_us/doc/whitepaper2/tdwi-managing-big-data-106702.pdf

[40] Kaushik Pal, “The Impact of Internet of Things on Big Data”, 2015. Available at: **h**ttp://data-informed.com/the-impact-of-internet-of-things-on-big-data/

[41] Charu C. Aggarwal, et. al., “THE INTERNET OF THINGS: A SURVEY FROM THE DATA-CENTRIC PERSPECTIVE ”, Managing and Mining Sensor Data, ISBN 978-1-4614-6308-5, 2013.

[42] Kamburugamuv S., “Survey of Distributed Stream Processing for Large Stream Sources”, 2013. Available at: http://grids.ucs.indiana.edu/ptliupages/publications/survey_stream_processing.pdf

[43] “The Aurora Project” , 2004. Available at: http://cs.brown.edu/research/aurora/ .

[44] “Borealis Distributed Stream Processing Engine” 2006. Available at: http://cs.brown.edu/research/borealis/public/

[45] “Apache Storm”, 2015. Available at: http://storm.apache.org/ .

[46] Roland Kotto-Kombi, Nicolas Lumineau, Philippe Lamarre, Yves Caniou, “Parallel and Distributed Stream Processing: Systems Classification and Specific Issues”. 2015. <hal-01215287>

[47] Leonardo, Bruce Robbins, Anish Nair, and Anand Kesari Neumeyer, "S4: Distributed stream computing platform", in (ICDMW), 2010 IEEE International Conference, 2010.

[48] Ramesh Navina, “Apache Samza, LinkedIn’s Framework for Stream Processing”, 2015. Available at: http://thenewstack.io/apache-samza-linkedins-framework-for-stream-processing/

[49] “About VoltDb”, 2016. Available at: https://voltdb.com/about

[50] Sain Technology Solutions “Introduction to Apache Storm”, 2015. Available at:

http://www.allprogrammingtutorials.com/tutorials/introduction-to-apache-storm.php

[51] Erricson Research Blog, “Real Time Processing Frameworks”, 2014. Available at: http://www.ericsson.com/research-blog/data-knowledge/real-time-processing-frameworks/

[52] Kamburugamuv S., “Survey of Apache Big Data Stacks”, 2013. Available at: http://grids.ucs.indiana.edu/ptliupages/publications/survey_apache_big_data_stack.pdf

[53] http://www.business-software.com/wp-content/uploads/2014/09/Storm.png

[54] Abadi D. J., et.al. “Aurora: a new model and architecture for data stream management”, The VLDB Journal (2003).

[55] Jain Nitin, “Key to Unlock the Power of Internet of Things (IoT) - Big Data & Analytics”, 2015. Available at: https://www.linkedin.com/pulse/key-unlock-power-internet-things-iot-big-data-analytics-nitin-jain

[56] iiht Official Blog, “5Vs of Hadoop Big Data”, 2014. Available at: http://iihtofficialblog.blogspot.com.ng/2014/07/5-vs-of-hadoop-big-data.html

[57] Silicon Labs, “Internet of Things Keynote #12: The IoT from Things to Big Data” Available at: http://community.silabs.com/t5/Official-Blog-of-Silicon-Labs/Internet-of-Things-Keynote-12-The-IoT-from-Things-to-Big-Data/ba-p/131778

[58] ARM, “Internet of Things”, 2015. Available at: https://www.arm.com/ja/markets/internet-of-things-iot.php

[59] Richard Hackathorn, “The BI Watch: Real-Time to Real Value,” DM Review , 2004.

[60] More Pranali “REVIEW OF IMPLEMENTING FOG COMPUTING” , International Journal of Research in Engineering and Technology (IJRET), Volume: 04 Issue: 06, 2015.

[61] John K. Z., et. al. “Pervasive brain monitoring and data sharing based on multi-tier distributed computing and linked data technology” , Frontiers in Human Neuroscience, 2014. Available at: http://journal.frontiersin.org/article/10.3389/fnhum.2014.00370/full

[62] Bonomi F. et. al., “Fog Computing and Its Role in the Internet of Things”, in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (New York, NY: ACM), 2012.

# Appendix

## 1. Regions Class: A class defining the geographic regions in polygons

```
package geospatial;

import java.util.Random;

import org.voltdb.types.GeographyValue;

/**
 * Defines a set of regions as polygons roughly around the South Bay area
in California.
 */
public class Regions {
    private static final Random RAND = new Random(888);
    private static final GeographyValue[] POLYGONS = {
        GeographyValue.fromWKT("POLYGON ((-121.877094307528
37.393549788232, -121.857799131047 37.402928988778, -121.875569345227
37.414949501508, -121.877094307528 37.393549788232))"),
        GeographyValue.fromWKT("POLYGON ((-121.937723896967
37.406109873323, -121.948270936942 37.406822679761, -121.956232850697
37.399868826539, -121.956945657135 37.389321786564, -121.949991803913
37.381359872809, -121.939444763937 37.380647066371, -121.931482850183
37.387600919593, -121.930770043745 37.398147959569, -121.937723896967
37.406109873323))"),
        GeographyValue.fromWKT("POLYGON ((-121.836238386245
37.367375816559, -121.830292925514 37.381400226271, -121.836005605124
37.39552105708, -121.850030014836 37.401466517811, -121.864150845646
37.395753838202, -121.870096306377 37.38172942849, -121.864383626767
37.36760859768, -121.850359217055 37.361663136949, -121.836238386245
37.367375816559))"),
        GeographyValue.fromWKT("POLYGON ((-121.990803320492
37.323566511634, -121.995730632055 37.307370834623, -121.979534955045
37.30244352306, -121.974607643482 37.318639200071, -121.990803320492
37.323566511634))"),
        GeographyValue.fromWKT("POLYGON ((-121.971357167486
37.260306030788, -121.961361381973 37.262584890995, -121.96364024218
37.272580676509, -121.973636027693 37.270301816302, -121.971357167486
37.260306030788))"),
        GeographyValue.fromWKT("POLYGON ((-122.160420417102
37.335312891185, -122.165957489605 37.3263905148, -122.162434003087
37.316498446512, -122.152503214761 37.313085615503, -122.143643210789
37.318721953133, -122.142525754873 37.329163182204, -122.149992314108
37.336546844265, -122.160420417102 37.335312891185))"),
        GeographyValue.fromWKT("POLYGON ((-122.098136225109
37.431336300026, -122.115050794684 37.419060533836, -122.102775028493
37.402145964261, -122.085860458918 37.414421730451, -122.098136225109
37.431336300026))"),
        GeographyValue.fromWKT("POLYGON ((-121.790114981429
37.383134841282, -121.79636344619 37.365923698256, -121.779152303164
37.359675233495, -121.772903838403 37.376886376521, -121.790114981429
37.383134841282))"),
        GeographyValue.fromWKT("POLYGON ((-122.059255156907
37.376163585933, -122.050676544009 37.371594337553, -122.043679993389
37.378341107849, -122.0479345002 37.387080089587, -122.057560480634
37.385734307031, -122.059255156907 37.376163585933))"),
```

```
        GeographyValue.fromWKT("POLYGON ((-122.070415124747
37.342078602196, -122.059857857089 37.343887821811, -122.053672052996
37.352632248821, -122.055481272611 37.363189516478, -122.064225699621
37.369375320571, -122.074782967278 37.367566100956, -122.080968771371
37.358821673947, -122.079159551756 37.348264406289, -122.070415124747
37.342078602196))"),
        GeographyValue.fromWKT("POLYGON ((-121.811676472986
37.345149569966, -121.812993937728 37.354681283653, -121.820130041728
37.361136149944, -121.829745722618 37.361493865165, -121.837341696441
37.355587050387, -121.839363722627 37.346179569893, -121.834865672651
37.337673288357, -121.825952234086 37.334048389449, -121.816794103899
37.337001003652, -121.811676472986 37.345149569966))"),
        GeographyValue.fromWKT("POLYGON ((-121.7615029261 37.377354840825,
-121.75649997807 37.365337375692, -121.743524693582 37.366381861061, -
121.740508474785 37.379044853653, -121.751619633539 37.385826528105, -
121.7615029261 37.377354840825))"),
        GeographyValue.fromWKT("POLYGON ((-121.805222306558
37.406771549585, -121.812650629806 37.398019316393, -121.811714484108
37.386577935211, -121.802962250917 37.379149611962, -121.791520869735
37.380085757661, -121.784092546486 37.388837990852, -121.785028692184
37.400279372034, -121.793780925376 37.407707695283, -121.805222306558
37.406771549585))"),
        GeographyValue.fromWKT("POLYGON ((-122.006116100962
37.289459224174, -122.011760459403 37.267895147512, -121.992996006457
37.25586337748, -121.975754578314 37.269991411317, -121.983863242654
37.290754786455, -122.006116100962 37.289459224174))"),
        GeographyValue.fromWKT("POLYGON ((-121.997568540152
37.409414603312, -122.016000688743 37.40906028362, -122.015646369052
37.390628135029, -121.997214220461 37.39098245472, -121.997568540152
37.409414603312))"),
        GeographyValue.fromWKT("POLYGON ((-122.182683956548
37.259689895897, -122.170422679603 37.256123063745, -122.159989236801
37.263485432605, -122.159240223376 37.27623298856, -122.168739661714
37.284766561973, -122.181334280993 37.282660198011, -122.187540076013
37.271500031699, -122.182683956548 37.259689895897))"),
        GeographyValue.fromWKT("POLYGON ((-122.096482049875
37.278482685217, -122.08730516972 37.281934132174, -122.087751878916
37.291728419611, -122.097204840538 37.294330175184, -122.102600382918
37.286143861123, -122.096482049875 37.278482685217))"),
        GeographyValue.fromWKT("POLYGON ((-121.807312199097 37.43008269536,
-121.811221164734 37.424397313364, -121.809965048681 37.417613085091, -
121.804279666685 37.413704119453, -121.797495438412 37.414960235507, -
121.793586472774 37.420645617503, -121.794842588828 37.427429845776, -
121.800527970824 37.431338811413, -121.807312199097 37.43008269536))"),
        GeographyValue.fromWKT("POLYGON ((-122.118570249798
37.352928556994, -122.109770432626 37.368813110194, -122.122158247403
37.382090830544, -122.138614155152 37.374412359813, -122.13639665068
37.35638908357, -122.118570249798 37.352928556994))"),
        GeographyValue.fromWKT("POLYGON ((-121.880073129207
37.433576269856, -121.898690522765 37.436280014902, -121.90701502811
37.41940932461, -121.893542461795 37.40627891955, -121.876891452551
37.415034573229, -121.880073129207 37.433576269856))"),
        GeographyValue.fromWKT("POLYGON ((-122.038349410981
37.405397592909, -122.037212465856 37.410014753925, -122.039309369255
37.414282518704, -122.043658956772 37.416203952673, -122.048226008068
37.414879995525, -122.050873549084 37.410930141523, -122.050362765954
37.406202571253, -122.046932659783 37.402909367385, -122.042188215368
37.402591456608, -122.038349410981 37.405397592909))"),
```

```
        GeographyValue.fromWKT("POLYGON ((-122.190059774575
37.423284506276, -122.193969268814 37.401203880402, -122.174177445851
37.390662441789, -122.158035932322 37.40622810031, -122.167851751293
37.426389644947, -122.190059774575 37.423284506276))"),
        GeographyValue.fromWKT("POLYGON ((-121.984000813632
37.429223163029, -121.967459480503 37.424715858575, -121.955285373778
37.43678742105, -121.959652600183 37.453366287979, -121.976193933312
37.457873592433, -121.988368040036 37.445802029958, -121.984000813632
37.429223163029))"),
        GeographyValue.fromWKT("POLYGON ((-121.839127353673
37.348525345092, -121.837817089039 37.329607487828, -121.82208887638
37.340201138918, -121.839127353673 37.348525345092))"),
        GeographyValue.fromWKT("POLYGON ((-121.832869785007
37.255831300194, -121.828728434999 37.271162488243, -121.839934958315
37.28241459658, -121.85528283164 37.278335516869, -121.859424181648
37.26300432882, -121.848217658332 37.251752220482, -121.832869785007
37.255831300194))"),
        GeographyValue.fromWKT("POLYGON ((-121.849385712276
37.373265101684, -121.863534283384 37.380572891766, -121.876937300794
37.371973764801, -121.876191747098 37.356066847754, -121.86204317599
37.348759057671, -121.84864015858 37.357358184636, -121.849385712276
37.373265101684))"),
        GeographyValue.fromWKT("POLYGON ((-121.864059086889
37.322883303986, -121.859542567026 37.317684918657, -121.852662301643
37.317974935838, -121.848599270906 37.323534966677, -121.850413019829
37.330178194548, -121.856737758481 37.332902133367, -121.862810829655
37.329655601645, -121.864059086889 37.322883303986))"),
        GeographyValue.fromWKT("POLYGON ((-122.168516044664
37.336154934467, -122.184470726715 37.355158966178, -122.192951359925
37.331839790358, -122.168516044664 37.336154934467))"),
        GeographyValue.fromWKT("POLYGON ((-122.03546240617 37.422831282902,
-122.040740885647 37.416240215666, -122.039812745943 37.407847178695, -
122.033221678707 37.402568699217, -122.024828641736 37.403496838922, -
122.019550162258 37.410087906158, -122.020478301963 37.418480943128, -
122.027069369199 37.423759422606, -122.03546240617 37.422831282902))"),
        GeographyValue.fromWKT("POLYGON ((-121.786788629691 37.37490928998,
-121.815432564024 37.375594868001, -121.816118142046 37.346950933667, -
121.787474207712 37.346265355646, -121.786788629691 37.37490928998))"),
        GeographyValue.fromWKT("POLYGON ((-122.028319670084
37.354143624005, -122.040841152204 37.354182384754, -122.048678472983
37.344416862756, -122.045929970022 37.332200695258, -122.034665322111
37.326732905549, -122.023367038882 37.332130850803, -122.020542958051
37.344329768145, -122.028319670084 37.354143624005))"),
        GeographyValue.fromWKT("POLYGON ((-122.114000967447
37.427059806787, -122.124141031109 37.434241718781, -122.136525214573
37.433225475271, -122.145358820762 37.424486587891, -122.146508507165
37.412114079167, -122.139436322737 37.401897183437, -122.127451421171
37.39861649976, -122.116161671106 37.403807096491, -122.110849672069
37.41504023573, -122.114000967447 37.427059806787))"),
        GeographyValue.fromWKT("POLYGON ((-121.846581175915
37.453317815882, -121.84191979489 37.450152970322, -121.836314646346
37.450724835935, -121.832388441581 37.454765830444, -121.83197829544
37.460385127609, -121.83527611986 37.464953395834, -121.840738826144
37.466333091646, -121.84581035331 37.463878640041, -121.848117677434
37.458738506204, -121.846581175915 37.453317815882))"),
        GeographyValue.fromWKT("POLYGON ((-122.120295017623
37.366571458845, -122.124985344282 37.388146406208, -122.141324633454
37.373296990488, -122.120295017623 37.366571458845))"),
```

```
        GeographyValue.fromWKT("POLYGON ((-121.973356963195
37.414681812112, -121.966395291112 37.414498481895, -121.961911426229
37.41982703178, -121.963281810257 37.42665495502, -121.969474516073
37.429840686152, -121.975826309888 37.426985304656, -121.977554161408
37.420238971038, -121.973356963195 37.414681812112))"),
        GeographyValue.fromWKT("POLYGON ((-122.083927536924
37.256602168475, -122.071096112779 37.251083968933, -122.05990149972
37.259437208438, -122.061538310806 37.273308647485, -122.074369734951
37.278826847028, -122.08556434801 37.270473607522, -122.083927536924
37.256602168475))"),
        GeographyValue.fromWKT("POLYGON ((-121.748837497616
37.378174970027, -121.762469718169 37.381124248208, -121.774194608366
37.373570267214, -121.777143886547 37.359938046661, -121.769589905553
37.348213156463, -121.755957685 37.345263878282, -121.744232794803
37.352817859277, -121.741283516622 37.36645007983, -121.748837497616
37.378174970027))"),
        GeographyValue.fromWKT("POLYGON ((-122.028984357862
37.299496106068, -122.020699673882 37.30862858004, -122.024466286351
37.320369563815, -122.0365175828 37.322978073619, -122.044802266781
37.313845599646, -122.041035654312 37.302104615871, -122.028984357862
37.299496106068))"),
        GeographyValue.fromWKT("POLYGON ((-122.166580560196
37.393843491324, -122.161306991809 37.376134082471, -122.143597582956
37.381407650858, -122.148871151343 37.399117059711, -122.166580560196
37.393843491324))"),
        GeographyValue.fromWKT("POLYGON ((-121.811221531006
37.291755882448, -121.79851000009 37.279709932077, -121.783125541418
37.288076913009, -121.786328953975 37.305293941979, -121.803693230488
37.307567670135, -121.811221531006 37.291755882448))"),
        GeographyValue.fromWKT("POLYGON ((-122.000717973994
37.389215831956, -122.00481661131 37.380315338571, -122.001421186321
37.371123565104, -121.992520692937 37.367024927789, -121.98332891947
37.370420352777, -121.979230282155 37.379320846161, -121.982625707143
37.388512619628, -121.991526200527 37.392611256944, -122.000717973994
37.389215831956))"),
        GeographyValue.fromWKT("POLYGON ((-122.180903554404
37.334440373392, -122.187370963164 37.324085939595, -122.183307904245
37.31257363194, -122.171773943885 37.3085724529, -122.161454389484
37.3150953719, -122.160120075987 37.32723049789, -122.168775768672
37.335839833489, -122.180903554404 37.334440373392))"),
        GeographyValue.fromWKT("POLYGON ((-121.78473113096 37.312454731808,
-121.800400264589 37.316892929725, -121.804838462507 37.301223796096, -
121.789169328878 37.296785598179, -121.78473113096 37.312454731808))"),
        GeographyValue.fromWKT("POLYGON ((-122.129761859396
37.318073196053, -122.138116556823 37.320962981211, -122.144796532895
37.315172493576, -122.143121811539 37.306492220784, -122.134767114111
37.303602435626, -122.12808713804 37.309392923261, -122.129761859396
37.318073196053))"),
        GeographyValue.fromWKT("POLYGON ((-122.168027119293
37.348350086926, -122.169217082587 37.359721990027, -122.17966042121
37.364377403136, -122.188913796538 37.357660913142, -122.187723833243
37.346289010041, -122.177280494621 37.341633596933, -122.168027119293
37.348350086926))"),
        GeographyValue.fromWKT("POLYGON ((-121.992583074066
37.385208599158, -121.98667933669 37.383608075458, -121.981747076827
37.387225892999, -121.981500386753 37.393337761381, -121.986125029126
37.397341319054, -121.992138553913 37.396221805432, -121.995012654296
37.390822237107, -121.992583074066 37.385208599158))"),
```

```
        GeographyValue.fromWKT("POLYGON ((-121.738872368397
37.405654189445, -121.746470229676 37.409610055097, -121.752580349739
37.403606489331, -121.748758750334 37.395940215982, -121.740286751947
37.39720576425, -121.738872368397 37.405654189445))"),
        GeographyValue.fromWKT("POLYGON ((-122.129558412644
37.451408694491, -122.13178088514 37.460238274061, -122.140610464709
37.458015801566, -122.138387992214 37.449186221996, -122.129558412644
37.451408694491))"),
        GeographyValue.fromWKT("POLYGON ((-122.15617020982 37.383021933373,
-122.164386187044 37.376699142284, -122.164565407924 37.366333436867, -
122.156572915482 37.359730404725, -122.146427219545 37.361862263737, -
122.141768236087 37.371123680585, -122.146104274678 37.380540619485, -
122.15617020982 37.383021933373))"),
        GeographyValue.fromWKT("POLYGON ((-122.152524605875
37.448866002858, -122.163441761542 37.448521446495, -122.168601944812
37.438894634169, -122.162844972415 37.429612378206, -122.151927816749
37.429956934569, -122.146767633479 37.439583746895, -122.152524605875
37.448866002858))"),
        GeographyValue.fromWKT("POLYGON ((-122.03290369774 37.268840066725,
-122.027893427866 37.259872555615, -122.019749014119 37.253612556858, -
122.014488056627 37.262435333404, -122.013138945549 37.272618601986, -
122.023410172916 37.27276333655, -122.03290369774 37.268840066725))"),
        GeographyValue.fromWKT("POLYGON ((-121.875888611149
37.361945245099, -121.872483058801 37.362678925585, -121.869166434332
37.363744794235, -121.871504596445 37.366327248839, -121.874085978007
37.368666595559, -121.875153368242 37.365350460469, -121.875888611149
37.361945245099))"),
        GeographyValue.fromWKT("POLYGON ((-121.909212761308
37.369076148347, -121.913847991686 37.36753267141, -121.918625847784
37.366512978894, -121.917082370847 37.361877748515, -121.91606267833
37.357099892418, -121.911427447952 37.358643369355, -121.906649591855
37.359663061872, -121.908193068792 37.36429829225, -121.909212761308
37.369076148347))"),
        GeographyValue.fromWKT("POLYGON ((-121.927631316308
37.376628197745, -121.932090611936 37.379624441796, -121.928289562892
37.383421127465, -121.933631530079 37.382850003593, -121.933160220087
37.388201700781, -121.936885293018 37.384330444194, -121.939964253261
37.388733034309, -121.940329428911 37.383373048988, -121.945517979673
37.384766511186, -121.942352388298 37.380425793832, -121.947222749012
37.378158111664, -121.942007605997 37.376867732169, -121.944280880758
37.371999979323, -121.939456409481 37.374363720594, -121.938068907763
37.369173572727, -121.935892531951 37.374085413953, -121.931493481228
37.371001398932, -121.932983551312 37.376163035015, -121.927631316308
37.376628197745))"),
        GeographyValue.fromWKT("POLYGON ((-121.760173354465 37.35312275237,
-121.756996095628 37.341024391259, -121.755443315509 37.32861253446, -
121.743344954399 37.331789793297, -121.7309330976 37.333342573415, -
121.734110356436 37.345440934526, -121.735663136555 37.357852791325, -
121.747761497666 37.354675532488, -121.760173354465 37.35312275237))"),
        GeographyValue.fromWKT("POLYGON ((-121.740304951166
37.252511776004, -121.740742605781 37.257698702017, -121.738136821769
37.262204879189, -121.743205124267 37.263391493203, -121.746685521887
37.267262226393, -121.749380250481 37.262808668172, -121.754137038517
37.260694735673, -121.750734169881 37.256755671307, -121.750193628944
37.251578455983, -121.745395811873 37.253597538542, -121.740304951166
37.252511776004))"),
        GeographyValue.fromWKT("POLYGON ((-121.978536475521 37.40228232499,
-121.978722013584 37.404689866647, -121.976669987755 37.405962608253, -
121.978847749022 37.407005698406, -121.978923962671 37.409419175706, -
```

```
121.980916185874 37.408054724202, -121.983044425353 37.409195459896, -
121.98285888729 37.40678791824, -121.984910913119 37.405515176633, -
121.982733151852 37.404472086481, -121.982656938203 37.40205860918, -
121.980664715 37.403423060685, -121.978536475521 37.40228232499))"),
        GeographyValue.fromWKT("POLYGON ((-121.916733594663
37.371941920919, -121.917772987004 37.374225813401, -121.91648315581
37.376378215966, -121.91891682538 37.376989569983, -121.919795444872
37.379340001666, -121.921790788847 37.377818455173, -121.924176240628
37.378596992977, -121.924230724297 37.376088301517, -121.926326714521
37.374708690597, -121.92439931057 37.373101950008, -121.924627515848
37.370603065524, -121.922169598764 37.371108184241, -121.920358175867
37.369371737179, -121.919220607347 37.371608350505, -121.916733594663
37.371941920919))"),
        GeographyValue.fromWKT("POLYGON ((-121.83642184556 37.458501238609,
-121.84192260987 37.456958950208, -121.845745563628 37.461204192964, -
121.847160284847 37.455669247132, -121.852748249799 37.454481093438, -
121.848662206709 37.450488436006, -121.850427217903 37.445055039555, -
121.844926453593 37.446597327957, -121.841103499835 37.4423520852, -
121.839688778615 37.447887031033, -121.834100813663 37.449075184726, -
121.838186856754 37.453067842158, -121.83642184556 37.458501238609))"),
        GeographyValue.fromWKT("POLYGON ((-121.846185963602
37.431831053759, -121.848767985118 37.430927088968, -121.851464006689
37.431391260425, -121.851402173613 37.428656271554, -121.852676743384
37.426235639539, -121.850056506925 37.425449288249, -121.848148212794
37.423489083933, -121.84659065068 37.425738080979, -121.844136690275
37.426947240101, -121.845794300408 37.429123548007, -121.846185963602
37.431831053759))"),
        GeographyValue.fromWKT("POLYGON ((-122.145411669944
37.386017256956, -122.150427547853 37.392478574111, -122.146048394825
37.399387312731, -122.154227404718 37.399494306822, -122.156898516822
37.407225594599, -122.162081697428 37.400897696894, -122.169791672769
37.403629716442, -122.168075983375 37.395631962978, -122.175019053266
37.391307447855, -122.167696442978 37.387662310115, -122.168644334179
37.379537708412, -122.161228877898 37.382990049462, -122.155467809001
37.377183351973, -122.153543496555 37.385133488586, -122.145411669944
37.386017256956))"),
        GeographyValue.fromWKT("POLYGON ((-122.175072556813
37.339152169094, -122.175761078975 37.335618743906, -122.178823889544
37.333727091785, -122.17610811465 37.331364101508, -122.176001301142
37.327765803687, -122.172597004086 37.328936238598, -122.169427380009
37.327229592899, -122.168738857847 37.330763018087, -122.165676047278
37.332654670208, -122.168391822172 37.335017660486, -122.16849863568
37.338615958306, -122.171902932736 37.337445523395, -122.175072556813
37.339152169094))"),
        GeographyValue.fromWKT("POLYGON ((-121.974152377653
37.376573315628, -121.962509882735 37.374595470509, -121.950724046064
37.375339522393, -121.954832429406 37.386411141314, -121.961369715574
37.396245949334, -121.968903827151 37.387152175531, -121.974152377653
37.376573315628))"),
        GeographyValue.fromWKT("POLYGON ((-122.081986065779
37.309048103438, -122.088307063409 37.304874825308, -122.091815371915
37.311587725785, -122.093975005551 37.304327750307, -122.100977495039
37.307215033175, -122.097965248101 37.30026538356, -122.105185375914
37.297976057077, -122.098410712222 37.294588551617, -122.102470100314
37.288193817088, -122.095102960301 37.289953507236, -122.094102175869
37.282445532016, -122.089589726224 37.288529039194, -122.083997047426
37.28342088833, -122.084450713489 37.29098167192, -122.07688301689
37.290663505973, -122.082090523267 37.296163791303, -122.076088818211
```

```
37.300784483655, -122.083613514794 37.301650626092, -122.081986065779
37.309048103438))"),
        GeographyValue.fromWKT("POLYGON ((-121.851760371304
37.409949894428, -121.859015163753 37.405956351411, -121.862477409672
37.413479200106, -121.865467903143 37.405756677973, -121.872955731322
37.409294025634, -121.870282640685 37.401455978317, -121.878292413099
37.39935266066, -121.871206507171 37.395066597608, -121.875990362291
37.38830678034, -121.867807215206 37.389578198091, -121.867126734054
37.381324874839, -121.861675331279 37.387558862896, -121.855848918549
37.381673855335, -121.855680032028 37.389953461403, -121.847433930989
37.389190429976, -121.852626584602 37.395641534358, -121.845819237577
37.400357509949, -121.853943730988 37.40196156921, -121.851760371304
37.409949894428))"),
        GeographyValue.fromWKT("POLYGON ((-122.112254639222 37.28176410241,
-122.10954279343 37.277881394876, -122.112668918984 37.274323746157, -
122.108005865609 37.273495821879, -122.107700732653 37.268769679766, -
122.103818025119 37.271481525558, -122.100260376399 37.268355400003, -
122.099432452122 37.273018453379, -122.094706310009 37.273323586335, -
122.097418155801 37.277206293869, -122.094292030246 37.280763942588, -
122.098955083622 37.281591866866, -122.099260216578 37.286318008979, -
122.103142924112 37.283606163187, -122.106700572831 37.286732288741, -
122.107528497109 37.282069235366, -122.112254639222 37.28176410241))"),
        GeographyValue.fromWKT("POLYGON ((-121.854797624137
37.445206190763, -121.858674752582 37.451684569826, -121.854390616484
37.457901297113, -121.861939621551 37.457782794918, -121.865181397261
37.464601329255, -121.868853273882 37.458004447996, -121.87637918569
37.458606255047, -121.872502057245 37.452127875984, -121.876786193343
37.445911148697, -121.869237188276 37.446029650892, -121.865995412566
37.439211116555, -121.862323535945 37.445807997813, -121.854797624137
37.445206190763))"),
        GeographyValue.fromWKT("POLYGON ((-121.918806030972
37.275549627606, -121.906732831009 37.278628282365, -121.894282728862
37.278143278218, -121.897361383621 37.290216478181, -121.896876379474
37.302666580328, -121.908949579437 37.299587925569, -121.921399681584
37.300072929716, -121.918321026825 37.287999729753, -121.918806030972
37.275549627606))"),
        GeographyValue.fromWKT("POLYGON ((-121.980098959495
37.274283510618, -121.971739667153 37.27479801789, -121.963730595531
37.27724694785, -121.968355818069 37.284229053741, -121.974481189438
37.289940648246, -121.978215259242 37.282444035084, -121.980098959495
37.274283510618))"),
        GeographyValue.fromWKT("POLYGON ((-121.796298869554
37.371477112795, -121.792831144424 37.364643782369, -121.796031715938
37.35768132129, -121.788461246511 37.358867708642, -121.782828583522
37.353672265451, -121.781617501236 37.361238823585, -121.774935752547
37.364990324186, -121.781757731958 37.368480326938, -121.783260847153
37.375994325009, -121.788688144586 37.370584707197, -121.796298869554
37.371477112795))"),
        GeographyValue.fromWKT("POLYGON ((-121.834920335017
37.319416496241, -121.837978100304 37.318603425509, -121.841142007905
37.318629906672, -121.840328937174 37.315572141385, -121.840355418336
37.312408233783, -121.837297653049 37.313221304515, -121.834133745447
37.313194823352, -121.834946816179 37.316252588639, -121.834920335017
37.319416496241))"),
        GeographyValue.fromWKT("POLYGON ((-122.040265610731
37.279689033833, -122.044182240481 37.280899158525, -122.044634066209
37.284973498211, -122.04825922904 37.283059710131, -122.051459711296
37.285621214316, -122.052669835989 37.281704584566, -122.056744175674
37.281252758839, -122.054830387595 37.277627596008, -122.05739189178
```

```
37.274427113751, -122.05347526203 37.273216989059, -122.053023436302
37.269142649373, -122.049398273471 37.271056437452, -122.046197791214
37.268494933268, -122.044987666522 37.272411563017, -122.040913326836
37.272863388745, -122.042827114916 37.276488551576, -122.040265610731
37.279689033833))"),
        GeographyValue.fromWKT("POLYGON ((-121.898844019793
37.319696052495, -121.900365554478 37.309791700644, -121.907923907133
37.30321275251, -121.89897448884 37.298705073992, -121.895053196766
37.289483646669, -121.888000617396 37.296602099986, -121.878018772961
37.29748189261, -121.882609457494 37.306389017225, -121.880361630436
37.316154186292, -121.89025140888 37.31454063873, -121.898844019793
37.319696052495))"),
        GeographyValue.fromWKT("POLYGON ((-122.165703283001
37.306955828621, -122.160109319002 37.308459398827, -122.157116578666
37.303499896648, -122.153797829929 37.308247428996, -122.148317351273
37.306371926319, -122.148826697215 37.312141997661, -122.143422857113
37.314228063032, -122.147521969107 37.31832079286, -122.144723284398
37.323392333107, -122.150494142386 37.323892687649, -122.15161014875
37.329576672727, -122.156352504143 37.326250530531, -122.160861009799
37.329887360653, -122.162355861804 37.324291060619, -122.168147286849
37.32417902255, -122.165695177599 37.31893113566, -122.17005964989
37.315122653259, -122.164807948555 37.312678724113, -122.165703283001
37.306955828621))"),
        GeographyValue.fromWKT("POLYGON ((-122.092319488831
37.395902680725, -122.079946384541 37.400943208165, -122.070607504458
37.410497576037, -122.081159281414 37.418692734956, -122.09410304675
37.422003258415, -122.095924374084 37.408767572056, -122.092319488831
37.395902680725))"),
        GeographyValue.fromWKT("POLYGON ((-122.122464714335
37.430421366138, -122.127670104581 37.426256082278, -122.133679450692
37.429142851504, -122.13441491697 37.4225167843, -122.140705420451
37.420308789009, -122.136540136591 37.415103398764, -122.139426905817
37.409094052653, -122.132800838613 37.408358586374, -122.130592843322
37.402068082894, -122.125387453077 37.406233366754, -122.119378106966
37.403346597528, -122.118642640687 37.409972664732, -122.112352137207
37.412180660022, -122.116517421066 37.417386050268, -122.113630651841
37.423395396379, -122.120256719045 37.424130862657, -122.122464714335
37.430421366138))"),
        GeographyValue.fromWKT("POLYGON ((-121.989703939722
37.306107949194, -121.997731076128 37.303714632571, -121.999823306516
37.311825455917, -122.004434055484 37.304832325194, -122.011250333697
37.309700716577, -122.010287274541 37.30137993534, -122.018638188249
37.300727920326, -122.012551947052 37.29497287459, -122.01853001092
37.28910553825, -122.010168407579 37.288609077875, -122.010976419086
37.280271812091, -122.004251940733 37.285266236405, -121.999511822313
37.278360140496, -121.997570927104 37.286508502854, -121.989500632847
37.284265015848, -121.993251487221 37.291754606946, -121.985627197501
37.295223481346, -121.99331473501 37.298549838271, -121.989703939722
37.306107949194))"),
        GeographyValue.fromWKT("POLYGON ((-122.119427156036
37.372521670089, -122.10655564891 37.376130533559, -122.094011451916
37.380750350981, -122.103572572923 37.390092971402, -122.113845550668
37.398646655958, -122.117155936788 37.385695172067, -122.119427156036
37.372521670089))"),
        GeographyValue.fromWKT("POLYGON ((-121.776959720674
37.338289104272, -121.781428754804 37.344872379736, -121.77557364036
37.350260275126, -121.78350704958 37.350870848661, -121.784068871676
37.358807859296, -121.78949263703 37.352985956577, -121.796048332169
37.357495351563, -121.794878247721 37.349624984083, -121.802491243751
```

```
37.347311097021, -121.795608406956 37.343318812018, -121.79854596259
37.33592404706, -121.791133289939 37.33881611657, -121.787183365869
37.331908882555, -121.784822751059 37.339507519249, -121.776959720674
37.338289104272))"),
        GeographyValue.fromWKT("POLYGON ((-122.122025575174
37.380872905699, -122.117484931031 37.3820908361, -122.114148980498
37.378778399272, -122.112270151512 37.383087784999, -122.107600452979
37.383630665568, -122.109798235698 37.387786451272, -122.107311167405
37.391775849097, -122.111930586615 37.392648643381, -122.113498961714
37.397080460531, -122.11706150053 37.394013031497, -122.121504300582
37.395550019266, -122.121327294615 37.390852203541, -122.125299000564
37.388336978789, -122.121515738915 37.385546327431, -122.122025575174
37.380872905699))"),
        GeographyValue.fromWKT("POLYGON ((-122.033545778443
37.355658173452, -122.026008603158 37.360495623065, -122.017054130426
37.360660995833, -122.019325703151 37.369324129643, -122.01671589775
37.37789144228, -122.025656982187 37.37840810381, -122.032998506476
37.383537621444, -122.036252827831 37.375193802021, -122.043399944772
37.369796705627, -122.036470141542 37.364123280097, -122.033545778443
37.355658173452))"),
        GeographyValue.fromWKT("POLYGON ((-121.965149316532
37.433379736177, -121.959009110641 37.434816012567, -121.957328358775
37.428738176487, -121.954002178328 37.434095558486, -121.948516028179
37.430986350421, -121.949952304569 37.437126556311, -121.943874468489
37.438807308177, -121.949231850488 37.442133488625, -121.946122642423
37.447619638774, -121.952262848313 37.446183362383, -121.953943600179
37.452261198464, -121.957269780627 37.446903816465, -121.962755930776
37.45001302453, -121.961319654385 37.443872818639, -121.967397490466
37.442192066773, -121.962040108467 37.438865886326, -121.965149316532
37.433379736177))"),
        GeographyValue.fromWKT("POLYGON ((-121.946263907282
37.277507667261, -121.939573757903 37.27981694547, -121.933929254646
37.275547232653, -121.93258407355 37.282495711074, -121.926064142155
37.285249137878, -121.931409110437 37.289888338091, -121.9305336823
37.296911477711, -121.937223831678 37.294602199503, -121.942868334935
37.29887191232, -121.944213516032 37.291923433899, -121.950733447426
37.289170007095, -121.945388479144 37.284530806882, -121.946263907282
37.277507667261))"),
        GeographyValue.fromWKT("POLYGON ((-122.033142128963 37.26324647456,
-122.026501154929 37.261668883016, -122.023082998518 37.255760622871, -
122.018396277148 37.260723079319, -122.011570495565 37.260729159532, -
122.013524748229 37.267269207523, -122.010117123056 37.273183547881, -
122.01675809709 37.274761139424, -122.020176253501 37.280669399569, -
122.024862974871 37.275706943122, -122.031688756454 37.275700862909, -
122.029734503791 37.269160814918, -122.033142128963 37.26324647456))"),
        GeographyValue.fromWKT("POLYGON ((-121.967135528286
37.389115811061, -121.965208189234 37.380052909308, -121.974432289846
37.379177242766, -121.96713034152 37.373473526902, -121.973633544933
37.366873589829, -121.964373650108 37.367197891898, -121.96511303517
37.357961868201, -121.958228001547 37.364162443658, -121.952857601769
37.356611971473, -121.951569013098 37.365787502151, -121.942601698224
37.363455531327, -121.947512499465 37.371312684447, -121.939144175783
37.375290370049, -121.947956548459 37.378152696344, -121.944102847625
37.386578835072, -121.952693384621 37.383107020257, -121.955157496083
37.392038928154, -121.959506589666 37.383857472966, -121.967135528286
37.389115811061))"),
        GeographyValue.fromWKT("POLYGON ((-121.772101003698
37.447599563723, -121.765044204931 37.450411610237, -121.763897802066
37.442902164525, -121.760299529241 37.449592339941, -121.754594355037
```

```
37.444576664341, -121.756138279999 37.45201456123, -121.748543848873
37.451839546097, -121.754507551974 37.456544889842, -121.748577382652
37.461292426702, -121.756170380931 37.46106353467, -121.754679265546
37.468512198264, -121.760348710721 37.46345617158, -121.763994358733
37.470120649429, -121.765087454399 37.462603259171, -121.772164026586
37.465365168022, -121.768169301133 37.458903884638, -121.775365590721
37.456470896645, -121.768152220584 37.454089026439, -121.772101003698
37.447599563723))"),
        GeographyValue.fromWKT("POLYGON ((-121.894693070128 37.45757067615,
-121.895850813094 37.451507275179, -121.902023338099 37.451578935165, -
121.899012741649 37.446189917712, -121.903787232281 37.442277189852, -
121.898016987952 37.440084137074, -121.899159406981 37.434017830219, -
121.893329476224 37.436046895884, -121.89030527309 37.430665502431, -
121.887143545299 37.435967264164, -121.881367778258 37.433788797917, -
121.882353661005 37.439882502481, -121.876528875647 37.441926292008, -
121.881201063437 37.445960627312, -121.878052741566 37.45127036078, -
121.884225065807 37.4513576165, -121.885226342216 37.457448810607, -
121.890010703798 37.453548158821, -121.894693070128 37.45757067615))"),
        GeographyValue.fromWKT("POLYGON ((-122.007211471274 37.41114760587,
-122.004605204112 37.406994915619, -122.003518960632 37.402213959655, -
121.999366270381 37.404820226816, -121.994585314416 37.405906470297, -
121.997191581578 37.410059160548, -121.998277825059 37.414840116512, -
122.002430515309 37.412233849351, -122.007211471274 37.41114760587))"),
        GeographyValue.fromWKT("POLYGON ((-121.952615713615
37.376788434609, -121.948602494226 37.38039325221, -121.943299913919
37.379401646493, -121.945488146195 37.384332394845, -121.942906485688
37.389069025381, -121.948272106999 37.388511577851, -121.951979133365
37.392430582231, -121.95310703743 37.387155312359, -121.957979766228
37.384840759733, -121.953311227965 37.382137911181, -121.952615713615
37.376788434609))"),
        GeographyValue.fromWKT("POLYGON ((-122.176680423543
37.298801542038, -122.17190904069 37.294091404825, -122.173654487622
37.287618005262, -122.167189697713 37.289395075418, -122.162456292709
37.284646774252, -122.160762885654 37.291133981621, -122.154284033717
37.292859080018, -122.159055416571 37.297569217231, -122.157309969639
37.304042616794, -122.163774759547 37.302265546638, -122.168508164551
37.307013847804, -122.170201571606 37.300526640435, -122.176680423543
37.298801542038))"),
        GeographyValue.fromWKT("POLYGON ((-122.089923450559
37.344455636075, -122.081209786977 37.346515430221, -122.077200759363
37.338509283673, -122.073378290893 37.346606160833, -122.064619245636
37.344748794111, -122.068566369001 37.352785640638, -122.061653045831
37.358475688766, -122.07039749663 37.360400595303, -122.0705357689
37.369353335986, -122.077492797328 37.36371680865, -122.084578543198
37.36919064555, -122.084509364951 37.360237104389, -122.093206873258
37.358110126675, -122.086163580967 37.352581770802, -122.089923450559
37.344455636075))"),
        GeographyValue.fromWKT("POLYGON ((-122.16951148068 37.458273389623,
-122.176089277289 37.453374151223, -122.184201423898 37.454583705898, -
122.183247457289 37.446437547734, -122.18835103567 37.440017000028, -
122.180819272452 37.436770080264, -122.177810704223 37.429139977883, -
122.171232907614 37.434039216283, -122.163120761004 37.432829661608, -
122.164074727614 37.440975819772, -122.158971149233 37.447396367478, -
122.166502912451 37.450643287242, -122.16951148068 37.458273389623))"),
        GeographyValue.fromWKT("POLYGON ((-121.937458212503 37.32199725111,
-121.932778397751 37.323984325055, -121.92799756038 37.32225437621, -
121.928441237715 37.327319184143, -121.925318598599 37.331331447086, -
121.930272621024 37.33247459659, -121.933123561286 37.336684260305, -
```

```
121.93574163819 37.332325957619, -121.940626255289 37.330915409934, -
121.937290293376 37.327078681237, -121.937458212503 37.32199725111))"),
        GeographyValue.fromWKT("POLYGON ((-122.150337685249
37.338120926545, -122.148143922704 37.34649712376, -122.14308322539
37.353522977201, -122.150371552425 37.358197756653, -122.15548969465
37.365181873923, -122.162187891023 37.359694849299, -122.170411774191
37.356985417712, -122.167263160179 37.348919470546, -122.167227657271
37.340260832465, -122.15858351042 37.340762827589, -122.150337685249
37.338120926545))"),
        GeographyValue.fromWKT("POLYGON ((-121.951187794308
37.438189779126, -121.954047975598 37.437319213079, -121.956920081512
37.438149592436, -121.956975968626 37.435160378679, -121.958653235862
37.432685444967, -121.955827594707 37.431708575312, -121.953992096953
37.429348602801, -121.952189867566 37.43173407791, -121.949378200332
37.432750468397, -121.951090002469 37.435201642748, -121.951187794308
37.438189779126))"),
        GeographyValue.fromWKT("POLYGON ((-122.118676232585 37.45791211585,
-122.12167368055 37.45702183085, -122.123949457046 37.459166158957, -
122.125122282442 37.456267576153, -122.128217709101 37.455825269148, -
122.126682750484 37.453101080511, -122.128266907898 37.450405204591, -
122.125180020413 37.449906779728, -122.124060005737 37.446987384446, -
122.121745678622 37.449090047446, -122.118764885752 37.448145496994, -
122.118965854528 37.45126589973, -122.116368881292 37.453007459864, -
122.118933812372 37.454795875433, -122.118676232585 37.45791211585))"),
        GeographyValue.fromWKT("POLYGON ((-122.132909428555
37.263902217184, -122.134450223456 37.256385415304, -122.13595501605
37.24886132342, -122.128674877216 37.251285356834, -122.121406426207
37.253744214162, -122.12714577014 37.258836982628, -122.132909428555
37.263902217184))"),
        GeographyValue.fromWKT("POLYGON ((-121.956697528197
37.405649860226, -121.960024996629 37.404972211189, -121.962520708991
37.407274964801, -121.963597581926 37.404054468089, -121.966839681233
37.403044494589, -121.964589085736 37.400501646915, -121.965335472681
37.397188919803, -121.962008004249 37.397866568841, -121.959512291887
37.395563815229, -121.958435418952 37.398784311941, -121.955193319645
37.399794285441, -121.957443915142 37.402337133115, -121.956697528197
37.405649860226))"),
        GeographyValue.fromWKT("POLYGON ((-121.946568191214
37.378477783604, -121.960848621058 37.376385750639, -121.974468527307
37.371610247288, -121.965516558691 37.360289048748, -121.954570898349
37.350881615615, -121.94924243712 37.36429484712, -121.946568191214
37.378477783604))"),
        GeographyValue.fromWKT("POLYGON ((-121.749726033534
37.287351634624, -121.744308888962 37.290773233835, -121.738111628438
37.289146347702, -121.738366247991 37.295548532124, -121.733858693009
37.300102074105, -121.739530457134 37.303082659314, -121.741220162676
37.309263087428, -121.746637307248 37.305841488216, -121.752834567771
37.307468374349, -121.752579948219 37.301066189928, -121.7570875032
37.296512647947, -121.751415739075 37.293532062737, -121.749726033534
37.287351634624))")
    };

    public static double MIN_LNG = -122.183915;
    public static double MAX_LNG = -122.742392;
    public static double MIN_LAT = 37.254242;
    public static double MAX_LAT = 37.459942;

    public static GeographyValue pickRandomRegion() {
        int whichRegion = RAND.nextInt(POLYGONS.length);
```

```
        return POLYGONS[whichRegion];
    }
}
```

**The Ad Broker Class:**

```
package geospatial;

import java.io.IOException;
import java.util.Random;
import java.util.Timer;
import java.util.TimerTask;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import org.voltdb.CLIConfig;
import org.voltdb.VoltTable;
import org.voltdb.client.Client;
import org.voltdb.client.ClientConfig;
import org.voltdb.client.ClientFactory;
import org.voltdb.client.ClientStats;
import org.voltdb.client.ClientStatsContext;
import org.voltdb.client.ClientStatusListenerExt;
import org.voltdb.client.NullCallback;
import org.voltdb.client.ProcCallException;
import org.voltdb.types.GeographyPointValue;

/**
 * This class simulates an ad broker application, with two streams of
 * data entering a VoltDB cluster:
 *   - The logins of devices from a particular place
 *   - The bids generated by businesses wishing to advertise on those
devices
 *
 *  This class initiates several tasks in parallel:
 *   - The simulation of businesses making bids to show ads
 *   - The simulation of devices requesting ads
 *   - Periodic display of stats (e.g., counts of ads displayed for the top
5 businesses)
 *   - Periodic deletion of expired bids and historical data.  (In a real
app,
 *     this data would probably be exported.)
 */
public class AdBrokerBenchmark {

    // handy, rather than typing this out several times
    static final String HORIZONTAL_RULE =
            "----------" + "----------" + "----------" + "----------" +
            "----------" + "----------" + "----------" + "----------" +
"\n";

    // validated command line configuration
    final AdBrokerConfig m_config;
```

```
    // Reference to the database connection we will use
    final Client m_client;

    // Instance of Random for generating randomized data.
    Random m_rand;

    // Benchmark start time
    long m_benchmarkStartTS;

    // Statistics manager objects from the client
    final ClientStatsContext m_periodicStatsContext;
    final ClientStatsContext m_fullStatsContext;

    final ScheduledExecutorService m_scheduler =
Executors.newScheduledThreadPool(4);

    // Parameters of generated bids:

    // The bid generator will generate one bid (where the advertiser
    // is chosen randomly) at a rate of BID_FREQUENCY_PER_SECOND.
    // Each generated bid will begin at the current timestamp and last
    // for BID_DURATION_SECONDS.  Thus, at any given time we'll have
    // (BID_DURATION_SECONDS * BID_FREQUENCY_PER_SECOND) active bids
    // in the system.
    static final int BID_DURATION_SECONDS = 20;
    static final int BID_FREQUENCY_PER_SECOND = 5;

    // Before the benchmark begins, the bid generator will randomly
    // generate NUM_BID_REGIONS polygons.  Each generated bid will
    // randomly choose one of these polygons as its bid region.
    //
    // The number chosen here is based in the number of active bids
    // that exist at any given time.
    static final int NUM_BID_REGIONS = BID_DURATION_SECONDS *
BID_FREQUENCY_PER_SECOND;

    // Devices will log in with a device id between 0 (inclusive) and
    // NUM_DEVICES (exclusive).
    static final int NUM_DEVICES = 1000000;

    /**
     * Uses included {@link CLIConfig} class to
     * declaratively state command line options with defaults
     * and validation.
     */
    static class AdBrokerConfig extends CLIConfig {
        @Option(desc = "Interval for performance feedback, in seconds.")
        long displayinterval = 5;

        @Option(desc = "Benchmark duration, in seconds.")
        int duration = 60;

        @Option(desc = "Warmup duration, in seconds.")
        int warmup = 5;

        @Option(desc = "Comma separated list of the form server[:port] to
connect to.")
        String servers = "localhost";
```

```
        @Option(desc = "Maximum TPS rate for benchmark.")
        int ratelimit = 150000;

        @Option(desc = "Report latency for async benchmark run.")
        boolean latencyreport = true;

        @Option(desc = "Filename to write raw summary statistics to.")
        String statsfile = "report.txt";

        @Option(desc = "User name for connection.")
        String user = "";

        @Option(desc = "Password for connection.")
        String password = "";

        @Override
        public void validate() {
            if (duration <= 0) exitWithMessageAndUsage("duration must be >
0");
            if (warmup < 0) exitWithMessageAndUsage("warmup must be >= 0");
            if (displayinterval <= 0)
exitWithMessageAndUsage("displayinterval must be > 0");
            if (ratelimit <= 0) exitWithMessageAndUsage("ratelimit must be
> 0");
        }
    }
    class StatusListener extends ClientStatusListenerExt {
        @Override
        public void connectionLost(String hostname, int port, int
connectionsLeft, DisconnectCause cause) {
            // if the benchmark is still active
            if ((System.currentTimeMillis() - m_benchmarkStartTS) <
(m_config.duration * 1000)) {
                System.err.printf("Connection to %s:%d was lost.\n",
hostname, port);
            }
        }
    }

    public AdBrokerBenchmark(AdBrokerConfig config) {
        m_config = config;

        ClientConfig clientConfig = new ClientConfig(config.user,
config.password, new StatusListener());
        clientConfig.setMaxTransactionsPerSecond(config.ratelimit);

        m_client = ClientFactory.createClient(clientConfig);

        m_rand = new Random(777);

        m_periodicStatsContext = m_client.createStatsContext();
        m_fullStatsContext = m_client.createStatsContext();

        System.out.print(HORIZONTAL_RULE);
        System.out.println(" Command Line Configuration");
        System.out.println(HORIZONTAL_RULE);
        System.out.println(config.getConfigDumpString());
        if (config.latencyreport && config.ratelimit == Integer.MAX_VALUE)
{
```

```
            System.out.println("WARNING: Option latencyreport is ON, and no
rate limit is set.  "
                    + "Expect high latencies to be reported.\n");
        }
    }
    static void connectToOneServerWithRetry(Client client, String server) {
        int sleep = 1000;
        while (true) {
            try {
                client.createConnection(server);
                break;
            }
            catch (Exception e) {
                System.err.printf("Connection failed - retrying in %d
second(s).\n", sleep / 1000);
                try { Thread.sleep(sleep); } catch (Exception interruted)
{}
                if (sleep < 8000) sleep += sleep;
            }
        }
        System.out.printf("Connected to VoltDB node at: %s.\n", server);
    }

    static void connect(final Client client, String servers) throws
InterruptedException {
        System.out.println("Connecting to VoltDB...");

        String[] serverArray = servers.split(",");
        final CountDownLatch connections = new
CountDownLatch(serverArray.length);

        // use a new thread to connect to each server
        for (final String server : serverArray) {
            new Thread(new Runnable() {
                @Override
                public void run() {
                    connectToOneServerWithRetry(client, server);
                    connections.countDown();
                }
            }).start();
        }
        // block until all have connected
        connections.await();
    }

    public void schedulePeriodicStats() {
        Runnable statsPrinter = new Runnable() {
            @Override
            public void run() { printStatistics(); }
        };
        m_scheduler.scheduleWithFixedDelay(statsPrinter,
                m_config.displayinterval,
                m_config.displayinterval,
                TimeUnit.SECONDS);
    }

    /**
     * Print stats for the last displayinterval seconds to the console.
     */
```

```
    public synchronized void printStatistics() {
        ClientStats stats =
m_periodicStatsContext.fetchAndResetBaseline().getStats();
        long time = Math.round((stats.getEndTimestamp() -
m_benchmarkStartTS) / 1000.0);

        System.out.printf("%02d:%02d:%02d ", time / 3600, (time / 60) % 60,
time % 60);
        System.out.printf("Throughput %d/s, ", stats.getTxnThroughput());
        System.out.printf("Aborts/Failures %d/%d",
                stats.getInvocationAborts(), stats.getInvocationErrors());
        if(m_config.latencyreport) {
            System.out.printf(", Avg/95%% Latency %.2f/%.2fms",
stats.getAverageLatency(),
                stats.kPercentileLatencyAsDouble(0.95));
        }
        System.out.printf("\n");

        VoltTable[] tables = null;
        try {
            tables = m_client.callProcedure("GetStats",
m_config.displayinterval).getResults();
        } catch (IOException | ProcCallException e) {
            e.printStackTrace();
            System.exit(1);
        }
    /*//the Expired Ads Request Export
          VoltTable[] tables2 = null;
        try {
            tables2 = m_client.callProcedure("ExportOldAdRequests", 1,
m_config.displayinterval).getResults();
        } catch (IOException | ProcCallException e) {
            e.printStackTrace();
            System.exit(1);
        }
    VoltTable expired = tables2 [0];
    //System.out.printf("Expired Ad requests exported to the Cloud layer
for further analytics\n");
     // m_client.writeSummaryCSV(expired, "MyExport.csv");
    while (expired.advanceRow()) {
            System.out.printf("%-40s  %9.2f    %d\n",
                    expired.getString(0),
                    expired.getDouble(1),
                    expired.getLong(2));
        //}
        System.out.println();
    }
*/


    //---------------------

        VoltTable hits = tables[0];
        hits.advanceRow();
        assert(hits.getLong(0) == 0);
        long unmetRequests = hits.getLong(1);
        hits.advanceRow();
        assert(hits.getLong(0) == 1);
        long metRequests = hits.getLong(1);
```

```
        long totalRequests = unmetRequests + metRequests;
        double percentMet = (((double)metRequests) / totalRequests) *
100.0;
        System.out.printf("Total number of ad requests: %d, %3.2f%%
resulted in an ad being served\n",
                totalRequests, percentMet);

        VoltTable recentAdvertisers = tables[1];
        System.out.println("\nTop 5 advertisers of the last " +
m_config.displayinterval + " seconds, "
                + "by sorted on the sum of dollar amounts of bids won:");
        System.out.println("Advertiser
Category                 Revenue   Count");
        System.out.println("----------                                 ------
--                 -------   -----");
        while (recentAdvertisers.advanceRow()) {
            System.out.printf("%-40s %-20s  %9.2f    %d\n",
                    recentAdvertisers.getString(0),
                recentAdvertisers.getString(1),
                    recentAdvertisers.getDouble(2),
                    recentAdvertisers.getLong(3));
        }
        System.out.println();
    }

    public synchronized void printResults() throws Exception {
        ClientStats stats = m_fullStatsContext.fetch().getStats();

        System.out.print(HORIZONTAL_RULE);
        System.out.println(" Client Workload Statistics");
        System.out.println(HORIZONTAL_RULE);

        System.out.printf("Average throughput:            %,9d txns/sec\n",
stats.getTxnThroughput());
        if(m_config.latencyreport) {
            System.out.printf("Average latency:               %,9.2f ms\n",
stats.getAverageLatency());
            System.out.printf("10th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.1));
            System.out.printf("25th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.25));
            System.out.printf("50th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.5));
            System.out.printf("75th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.75));
            System.out.printf("90th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.9));
            System.out.printf("95th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.95));
            System.out.printf("99th percentile latency:       %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.99));
            System.out.printf("99.5th percentile latency:     %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.995));
            System.out.printf("99.9th percentile latency:     %,9.2f ms\n",
stats.kPercentileLatencyAsDouble(.999));

            System.out.print("\n" + HORIZONTAL_RULE);
            System.out.println(" System Server Statistics");
            System.out.println(HORIZONTAL_RULE);
```

```
            System.out.printf("Reported Internal Avg Latency: %,9.2f ms\n",
stats.getAverageInternalLatency());

            System.out.print("\n" + HORIZONTAL_RULE);
            System.out.println(" Latency Histogram");
            System.out.println(HORIZONTAL_RULE);
            System.out.println(stats.latencyHistoReport());
        }

        // 4. Write stats to file if requested
        m_client.writeSummaryCSV(stats, m_config.statsfile);
    }


    private void shutdown() {

        // Stop the stats printer, the bid generator and the nibble
deleter.
        m_scheduler.shutdown();

        try {
            m_scheduler.awaitTermination(60, TimeUnit.SECONDS);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        try {
            // block until all outstanding txns return
            m_client.drain();
            // close down the client connections
            m_client.close();
        }
        catch (IOException | InterruptedException e) {
            e.printStackTrace();
        }
    }
    private GeographyPointValue getRandomPoint() {
        double lngRange = Regions.MAX_LNG - Regions.MIN_LNG;
        double lng = Regions.MIN_LNG + lngRange * m_rand.nextDouble();

        double latRange = Regions.MAX_LAT - Regions.MIN_LAT;
        double lat = Regions.MIN_LAT + latRange * m_rand.nextDouble();

        return new GeographyPointValue(lng, lat);
    }

    private void requestAd() {
        long deviceId = Math.abs(m_rand.nextLong()) %
AdBrokerBenchmark.NUM_DEVICES;
        GeographyPointValue point = getRandomPoint();

        try {
            m_client.callProcedure(new NullCallback(),
"GetHighestBidForLocation", deviceId, point);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

```
    public void runBenchmark() throws Exception {
        System.out.print(HORIZONTAL_RULE);
        System.out.println(" Setup & Initialization");
        System.out.println(HORIZONTAL_RULE);

        // connect to one or more servers, loop until success
        connect(m_client, m_config.servers);

        System.out.print("\n\n" + HORIZONTAL_RULE);
        System.out.println(" Starting Benchmark");
        System.out.println(HORIZONTAL_RULE);

        System.out.println("\nStarting bid generator\n");
        BidGenerator bidGenerator = new BidGenerator(m_client);
        long usDelay = (long) ((1.0 / BID_FREQUENCY_PER_SECOND) *
1000000.0);
        m_scheduler.scheduleWithFixedDelay(bidGenerator, 0, usDelay,
TimeUnit.MICROSECONDS);

        //System.out.println("\nStarting nibble deleter to delete expired
data\n");
        NibbleDeleter deleter = new NibbleDeleter(m_client,
m_config.displayinterval + 1);
        // Run once a second
        m_scheduler.scheduleWithFixedDelay(deleter, 1, 1,
TimeUnit.SECONDS);

        System.out.println("\nWarming up...");
        final long warmupEndTime = System.currentTimeMillis() + (1000l *
m_config.warmup);
        while (warmupEndTime > System.currentTimeMillis()) {
            requestAd();
        }

        // reset the stats
        m_fullStatsContext.fetchAndResetBaseline();
        m_periodicStatsContext.fetchAndResetBaseline();

        // print periodic statistics to the console
        m_benchmarkStartTS = System.currentTimeMillis();
        schedulePeriodicStats();

        // Run the benchmark loop for the requested duration
        // The throughput may be throttled depending on client
configuration
        System.out.println("\nRunning benchmark...");
        final long benchmarkEndTime = System.currentTimeMillis() + (1000l *
m_config.duration);
        while (benchmarkEndTime > System.currentTimeMillis()) {
            requestAd();
        }

        printResults();
        shutdown();
    }
    public static void main(String[] args) throws Exception {
        // create a configuration from the arguments
        AdBrokerConfig config = new AdBrokerConfig();
        config.parse(AdBrokerBenchmark.class.getName(), args);
```

```
        AdBrokerBenchmark benchmark = new AdBrokerBenchmark(config);
        benchmark.runBenchmark();
    }
}
```