2016

Adaptive Latency-Aware Query Processing in IoT Networks

Reshma Kotamsetty
Iowa State University

Adaptive latency-aware query processing in IoT networks

by

Reshma Kotamsetty

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Engineering

Program of Study Committee:
Manimaran Govindarasu, Major Professor
Doug Jacobson
Ahmed Kamal

Iowa State University

Ames, Iowa

2016

DEDICATION

I would like to dedicate this thesis to my family – my father K.V. Satyanarayana who has been my greatest strength and a constant source of knowledge and inspiration, my mother Jyothi, who has always showered immense love on me, my sisters Gowthami and Himaja, without whose care and support it is almost impossible for me to come this far.

TABLE OF CONTENTS

Page

LIST OF FIGURES iv

LIST OF TABLES v

ACKNOWLEDGMENTS vi

ABSTRACT vii

CHAPTER 1. INTRODUCTION 1
The Internet of Things 1
Overview 1
Background 3
Integration with Cloud Computing 4
Need for Security 6

CHAPTER 2. LITERATURE REVIEW 9
Outline of Existing Literature in the Field of IoT Security 9
Providing End-to-End Security 9
Privacy-Preserving Cryptographic Schemes 9
Cloud Security 10

CHAPTER 3. PROPOSED WORK 14
Adaptive Latency-Aware Query Processing 14
Background 14
Underlying Architecture 16
Adaptive Data Size Determination 18

CHAPTER 4. PERFORMANCE EVALUATION 25
Experimental Setup 25
Performance Evaluation 26

CHAPTER 5. CONCLUSIONS AND FUTURE WORK 31
Summary 31
Future work 32

REFERENCES 34

LIST OF FIGURES

Page

Figure 1. The Inter-Network of Things 1

Figure 2. Objective of the Internet of Things 2

Figure 3. Applications of the Internet of Things 4

Figure 4. Design of the Internet of Things Cloud 5

Figure 5. Conventional Encrypted Query Processing architectures 14

Figure 6. Illustration of the proposed scheme 15

Figure 7. Underlying Architecture of the proposed Algorithm 17

Figure 8. Pseudo code of the Adaptive query processing Algorithm 23

Figure 9. Timing Diagram of the Adaptive query processing protocol 24

Figure 10. Comparison of the latency performance of No-Step versus Fixed-Step versus the proposed Adaptive Scheme 28

Figure 11. Comparison of the energy performance of No-Step versus Fixed-Step versus the proposed Adaptive Scheme 30

Figure 12. Comparison of energy performance of Adaptive scheme by varying the initial step factors 30

LIST OF TABLES

Page

Table 1. Experimental Setup 26

Table 2. Initial Step-factor for better latency performance of Adaptive latency aware algorithm 30

ACKNOWLEDGMENTS

This work is supported in part by the U.S. National Science Foundation, Award # CNS 1528731 and CNS 1446831.

I am very grateful to my major professor, Manimaran Govindarasu, whose expertise, understanding, generous support and guidance made it possible for me to work on a topic of my interest. I sincerely thank him for his guidance and support throughout the course of this research, without which this thesis would not have been possible.

I would like to express my gratitude to Dr. Ahmed Kamal and Dr. Doug Jacobson for finding out time to respond to my emails and for accepting my request to be a part of my POS committee.

In addition, I would also like to thank the department faculty and staff for making my time at Iowa State University a wonderful experience. I want to also thank my best friends, who gave me great support and motivation to meet my goals.

ABSTRACT

The massive adoption of The Internet of Things (IoT) and the creation of a smart-world around us leads to several privacy and security concerns. There has been significant work in the past to address the privacy and confidentiality of IoT data such as: providing secure end-to-end channels for the transmission of IoT data, encrypting IoT data using optimized cryptographic schemes such as order-preserving and homomorphic encryption that impose a reasonable energy overhead while improving security. However, for data intensive IoT applications, decrypting large data sets using cryptographic schemes is significantly expensive in terms of latency as seen by the end user of the application.

In this thesis, an Adaptive Latency-Aware Query Processing over encrypted IoT data is proposed that aims to: (i) minimize query latency for data intensive applications as seen by the end user and, (ii) at the same time maintain low energy consumption overhead, comparable to the current schemes as much as possible. This work presents two main contributions: (i) a novel Adaptive latency-aware algorithm which chops down the results of a single large query into several iterations of small sized results by adaptively computing the suitable size (t) of data to be retrieved in each iteration, and (ii) a novel IoT architecture with server cache that implements a latency-hiding technique by establishing concurrency between computation and communication, while leaving the Cloud database unmodified. Both contributions together allow minimizing query latency while maintaining low energy overhead. The effectiveness of the proposed adaptive algorithm is evaluated for latency and energy performance. The results show that the proposed adaptive solution delivers significantly a better latency performance while being comparable to the existing solutions in terms of energy efficiency.

CHAPTER 1. INTRODUCTION

The Internet of Things

Overview

The Internet of Things (IoT) can be considered as a cutting edge vision of pervasive computing where tiny networked computers turn out to be a part of everyday objects entwining the virtual world and the physical world. It is an integral part of Future Web and could be characterized as a dynamic worldwide network infrastructure with self configuring capabilities relying on standard and interoperable communication protocols where things have physical traits, personalities, and virtual identities and use insightful interfaces, and are consistently integrated into the information network.

To make it less complex, IoT alludes a world of physical and virtual things which are uniquely recognized and capable of associating with each other, with individuals, and with the environment. It permits individuals and things to be associated at any time and anywhere, with anything and anyone. Communication among the things is accomplished by exchanging the information generated and sensed amid their interactions. All the more particularly,

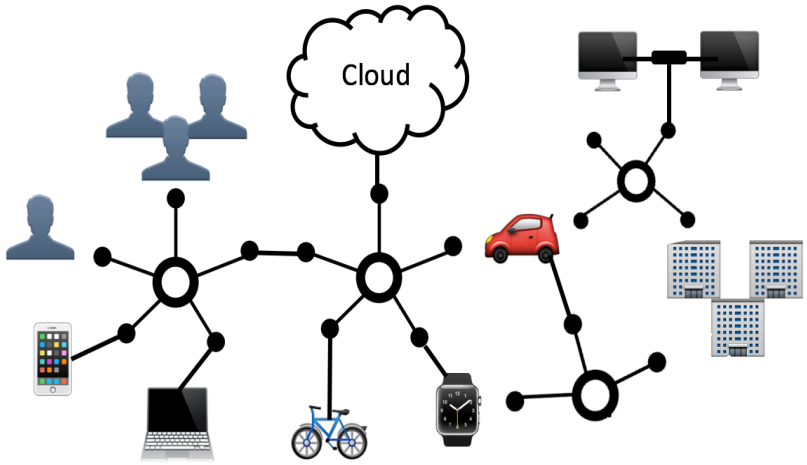


Figure 1. *The Inter-network of Things*

the IoT comprises of a variety of advancements which make it practical to access and control uniquely identifiable objects in our surroundings, for example, sensors and actuators.

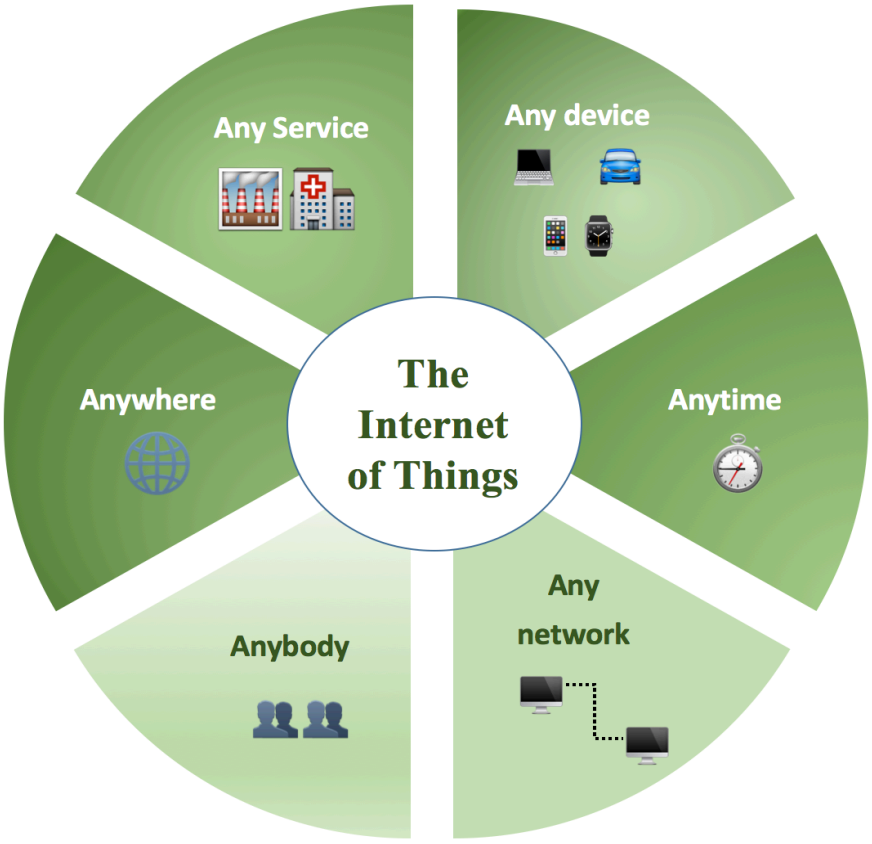


Figure 2. *Objective of the Internet of Things*

Henceforth, environments have ended up being "smart" by increasing physical objects with sensing or actuating abilities and networking them with advanced services. Continuous advancements of Internet Protocols for such IoT gadgets empowers the seamless integration of smart objects into the Internet. This pattern is expected to result in many billions of connected devices that need to be programmed, maintained, and managed. It has been shown that Web technologies can essentially facilitate this procedure by giving well-known patterns and tools for engineers, developers and clients.

Background

The idea of the IoT began two decades ago, from connecting real-world artifacts to virtual counterparts through radio-frequency identification (RFID) tags. Back then, the term fundamentally implied equipping everyday objects with RFID tags. Later on, advancements like Near Field Communications (NFC) [34], Universally Unique IDentifier (UUID) [33], and Wireless Sensor and Actuator Networks [35] in conjunction with RFID have turned into the center segments that make up the Internet of Things. Used as a part of everyday objects, these innovations alone may have noteworthy effect on society. However, the idea of the Internet of Things has developed from that point forward to include even more powerful network empowered embedded devices (sensors, switches, and so forth.) surrounding us/our homes that are interconnected to Wireless Sensor and Actuator Networks (WSANs), and autonomously gather information and act on changes in their surroundings.

Most IoT devices are resource-constrained, to minimize power utilization, dimensions, and unit costs. The idea of things has been extended to be of any kind: from human to electronic gadgets, for example, computers, phones, sensors, actuators. Any regular object can be made smart and turned into a thing in the network. For instance, TVs, vehicles, books, garments, pharmaceuticals, or food items can be equipped with embedded sensor devices that make them uniquely identifiable, have the capacity to gather data, interface with the Internet, and fabricate a network of networks of IoT objects.

With its efficiency, the scope of the Internet of Things is broad. It can give applicability and benefits to clients and organizations in an assortment of fields, including ecological monitoring, stock and product administration, client profiling, statistical surveying, medicinal services, smart homes, or security and surveillance [36]. Examples of such IoT applications are

innumerable. The distinctive research fields are unending, yet a portion of the more noticeable regions which the IoT can possibly impact significantly are brilliant smart grid, medicinal services, environmental monitoring, home automation, and so on.

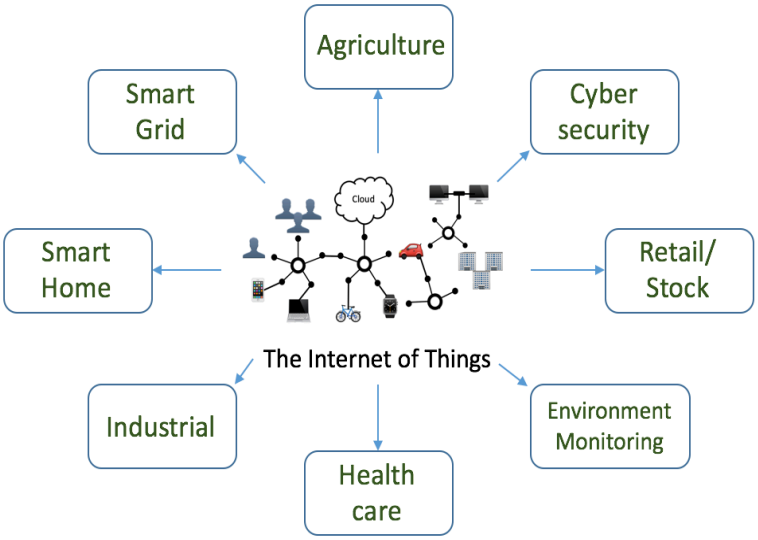


Figure 3. *Applications of the Internet of Things*

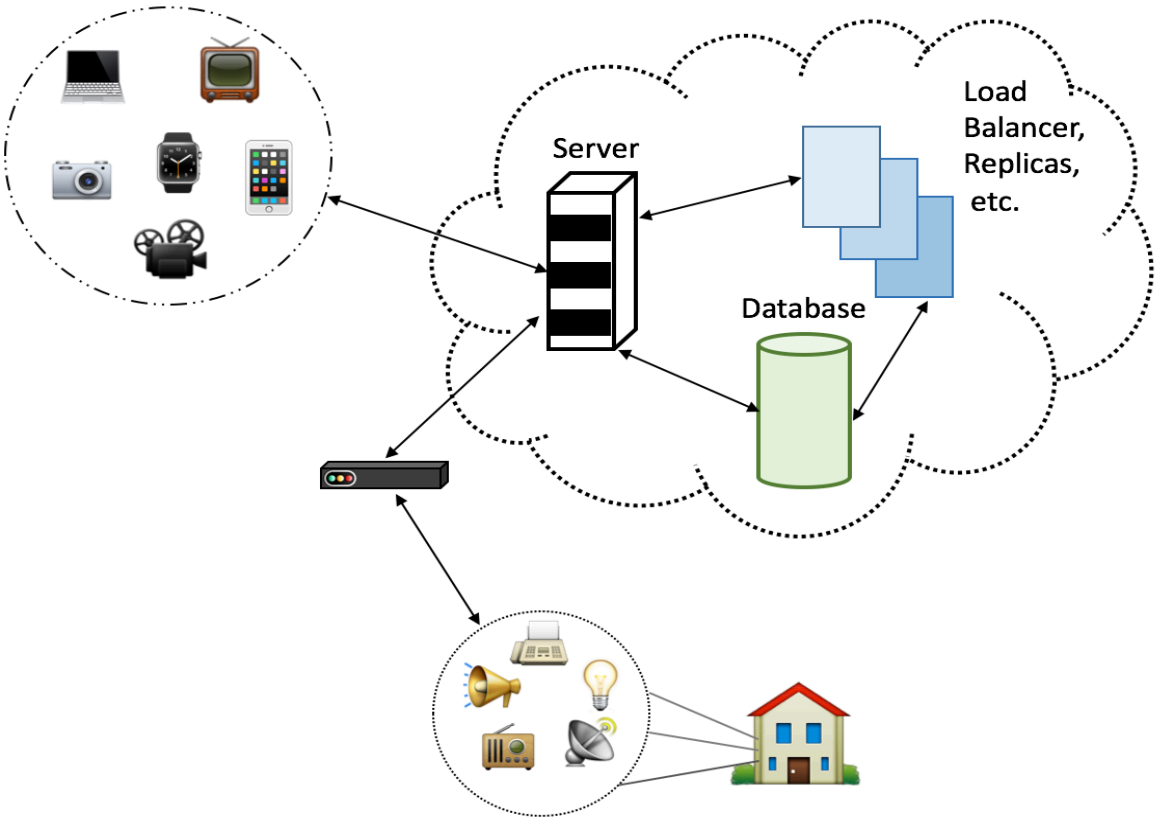
Integration with Cloud Computing

As of late, the term cloud computing has commanded the information industry as it got increasingly popular to move information into the so-called cloud, a straightforward network of computers intended to hide specific data from the clients and permit them to access only their resources from anywhere throughout the world. The primary quality of cloud computing is that it can adjust progressively to various circumstances, say, the number of requests, locations of clients around the world etc.

We have seen so far that The Internet of Things includes infrastructure to gather information from all the embedded sensors. The infrastructure then performs a real-time aggregation of this information in some fashion. The aggregated data is stored on a cloud and

processed using appropriate cloud-based applications and translated into useful intelligence. Here, the could-based applications are a crucial key to leveraging this data coming from sensors. The Internet of Things wouldn't function without efficient cloud-based applications to interpret and properly transmit the useful sensor data to the rest of the world.

Figure 4. *Design of the Internet of Things Cloud*



Cloud computing is very robust in the sense that, in the event that the available resources are insufficient to handle upcoming tasks, more resources are allocated to address the issues of the application, and if a lot of resources are being used compared to the really required amounts, some of those resources are de-allocated. This lessens costs for the client and thus provides resources to those applications that need it more, which is beneficial for the service provider.

There are fundamentally two deploy models for a cloud base, private and public cloud [32]. A private cloud is utilized solely by a solitary organization, whereas, in the public deploy model, the cloud access is conceivable by means of public network like the Internet. Today, there are three distinct service models being used by cloud suppliers to offer their diverse services to clients. Those service models are Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Starting from the earliest stage, IaaS gives (virtual) infrastructure on which a client's framework can run, PaaS gives infrastructure for Web applications to keep running on and SaaS gives programming functionality to the end-client to work with. Also, database administration is provided along with features like, load balancing, backup and recovery, on-the-go scaling and failover.

Need for Security

While the IoT systems provide huge opportunity, likewise there is huge risk. If there's one thing can keep the IoT from changing the way we live, it will be a breakdown in security. The typical IoT devices are physically integrated into our environment and are connected to Cloud services for data aggregation and processing. Hence these devices are a source of private and sensitive information which could be embezzled to infer security violating information. As a consequence, this arises many privacy and security concerns in the IoT world. Today, hackers are able to steal credit card credentials, steal your sensitive medical data, able to control your home, and so forth.

To address the security concerns in the field of Internet of Things, conventional methods provide secure end-to-end channels for the transmission of IoT data securely onto a Cloud database. While it is practically impossible to eavesdrop/perform man in the middle attacks on

such communication channels, it leaves the IoT data unprotected from hackers and curious administrators on the Cloud. The data is vulnerable to breaches caused my malicious Cloud service providers, leading to the disclosure of sensitive information [2] or unauthorized sharing of data [4].

End-To-End Security

Normally, most IoT applications follow a typical design, comprising of four sections: embedded devices/sensors, middleware such as a gateway device, servers (in the cloud or inside the network), and client-side applications. Checking and building up security properties over this hodgepodge of existing frameworks is obstinate. The objective of IoT security is to construct effective Internet of Things applications as simple as a modern web application.

The vital key to IoT applications is end-to-end security that backs an application information processing. With end-to-end security, information is encrypted before it leaves a sensor and is not decrypted until seen by an end-user: private information stays confidential as it goes through gateways, middleware devices, and cloud frameworks. With end-to-end security, your information is protected regardless of the possibility that any point along the whole information pipeline is compromised.

Lately, advances in cryptography have demonstrated that it is conceivable to build cryptosystems that can compute on encrypted data. Put another way, it's conceivable to send encrypted data to a server, request it to perform any computations as required (e.g., respond to queries, compute on the data), and yet not give it a chance to learn anything about the data. Moreover, recent results have demonstrated that such cryptosystems can be beneficial as long as they are intended for a narrow set of computations. Cryptosystems that can support any random computation can have a

million-fold slowdown, however, those cryptosystems intended for a particular application or calculation can impose just a little overhead.

The rest of this thesis is organized as follows: In Chapter 2, the background and the related work are discussed in detail. In Chapter 3, the proposed adaptive latency aware data size determination techniques and the underlying IoT architecture are described, along with the use case scenario. Later in Chapter 4, the experimental results obtained from the performance evaluation are detailed, by comparing the performance of the proposed technique with two variants: (i) conventional encrypted query processing technique without any latency optimization and (ii) implementing a heuristic-based fixed data size instead of adaptively determining the suitable size. Finally, the conclusions and future work are outlined in Chapter 5.

CHAPTER 2. LITERATURE REVIEW

As discussed in Chapter 1, there has been significant former work to address the security concerns in the field of Internet of Things, that is related to this work. It is grouped and outlined in the following subsections.

Outline of Existing Literature in the Field of IoT Security

Providing End-to-End Security

Early efforts to provide IoT security investigated low-power cryptographic methodologies [17,18], which put forth research on secure communication protocols for IoT. Hummen et al. [2] present a handshake methodology which permits very constrained IoT devices without the ability of computing public key-based computations to still take advantage from the security and scalability of public key-based handshake methods. Hu et al. [37] examined the possibility and the benefits of hardware accelerators for IoT devices.

Secure end-to-end channels provide transmission of confidential IoT data safely onto a Cloud database. While it is practically impossible to eavesdrop or perform man in the middle attacks on such communication channels, it leaves the IoT data unprotected from hackers and curious administrators on the Cloud. The data is vulnerable to breaches caused my malicious Cloud service providers, leading to the disclosure of sensitive information [2] or unauthorized sharing of data [4].

Privacy-Preserving Cryptographic Schemes

For previously proposed security solutions related to privacy-preserving cryptographic schemes [19,20,21,22], a practically implementable fully homomorphic encryption (FHE) scheme

[16] marks a breakthrough in cryptography. From that point forward, this scheme has been incrementally upgraded up to 6 orders of magnitudes by their research team [38]. Preceding this, the attention was on fractional homomorphic encryption, where there is only one kind of calculation, for example, addition or multiplication but not both are supported [39].

In spite of the fact that the Fully Homomorphic Encryption gives semantic security while also supporting computations over encrypted data, it is not the most appropriate for encrypted query processing. This is because both its very expensive and the Cloud must process every single existing data in database for requests, for example, equality check or correlation. This is the reason for utilizing weaker encryption schemes like, Order Preserving Encryption to allow the database to lessen the extent of computation. Secure multi-party calculation approaches [22] are productive for basic functions, however they turn out to be computationally costly for general functions. Also, these methodologies include huge interactions, extensive transfer speed, and coordination among the parties. Differential privacy [40] relies on a trusted server, which muddles the responses to abstain from leaking sensitive data and the query patterns.

Cloud Security

Cloud database administrations, for example, Google [41], encrypt information before storing, but, the queries are still handled over plaintext information. Secure information storage is a key measure. Using a nearby trusted machine [42, 43] at the database is an alternative way to encrypted query processing. This, however, relies on the fact that the client considers the trusted equipment dependable and trustworthy.

Several different approaches have been proposed that store IoT data in an encrypted form on the Cloud [8], and perform all the data encryption/decryption at the application user-side. This, however, has another drawback that it prevents any server-side manipulation of data which might be required for processing a query, leading to unanticipated application latencies. To cope with this drawback, many encrypted query processing techniques [5,6,7,9] were proposed. With a little extra processing at the user-side, these approaches allow server-side computations to be performed on encrypted IoT data, without revealing any secrets (like passwords, encryption/decryption keys) to the Cloud administrators.

These schemes integrate efficient encrypted query processing into database management in Cloud computing. These approaches overcome security attacks like: Network-based attacks [15], attacks due to Cloud database compromise [16], eavesdropping, modification attacks, etc. In these kind of approaches, the IoT data is stored on the Cloud database in an encrypted form and all the data encryption/decryption is controlled by the users who uniquely hold the underlying master secrets or encryption/decryption keys. This further led to computations on encrypted data [24] and query rewriting, to successfully pose a query over encrypted data stored on the Cloud database.

Different schemes implemented this approach at different elements of the IoT architecture, some at the client-side [10] and some on an intermediate proxy [6], leaving the client and the server free from all the additional computational overhead this processing may involve. These solutions address the Cloud security concerns in IoT while imposing a reasonable computational overhead, or low energy overhead, slight decrease in the throughput [6], etc.

Along those lines, CryptDB [6], is the first practical implementation of encrypted query processing in Cloud databases using SQL-like queries. The authors proposed a novel solution to execute SQL database queries on encrypted data and return the encrypted results. In this, the data encryption/decryption takes place on a trusted proxy, which sits between the client and the server. The trusted proxy is transparent to all the communications between user and the Cloud database. It intercepts every query posed by the client before it reaches the Cloud and performs appropriate query processing to protect the user data from the Cloud (server) and sends the modified query to the Cloud database. The returned results from the server in encrypted form are decrypted before sending back to the user. This way, the trusted proxy takes care of client data privacy and protection, while leaving the client and server unmodified. This technique comes with an additional computational overhead of around 25% but doesn't require any modification on the client and server-side.

While CryptDB is well suited to meet the needs of web applications, Talos [10] implements a similar data protection system over IoT data using optimized encrypted query processing techniques. It is a secure end-to-end system that stores encrypted IoT data securely on the Cloud and data encryption/decryption is performed at the client-side. Thus, the need of a trusted proxy which has access to all the secret keys is eliminated. This technique comes with a slight energy overhead [10] while integrating data security into IoT ecosystem. However, one drawback with these schemes is irrespective of where this encrypted query processing occurs (client-side or server-side [1] or on an intermediate proxy), the end user has to face certain latency before the requested results are made available, if the queried dataset is huge.

While the efficiencies of these encrypted query processing schemes are witnessed in terms of energy, computation and communication overhead, none of the above techniques discussed

about the scalability and latency issues involved in querying over encrypted data. For data intensive IoT applications needing scalability, decrypting large data sets using cryptographic schemes is significantly expensive in terms of latency as seen by the end user of the application. This thesis work addresses the latency issues involved in query processing over encrypted data, which will be discussed further in the next section.

CHAPTER 3. PROPOSED WORK

Adaptive Latency-Aware Query Processing

Background

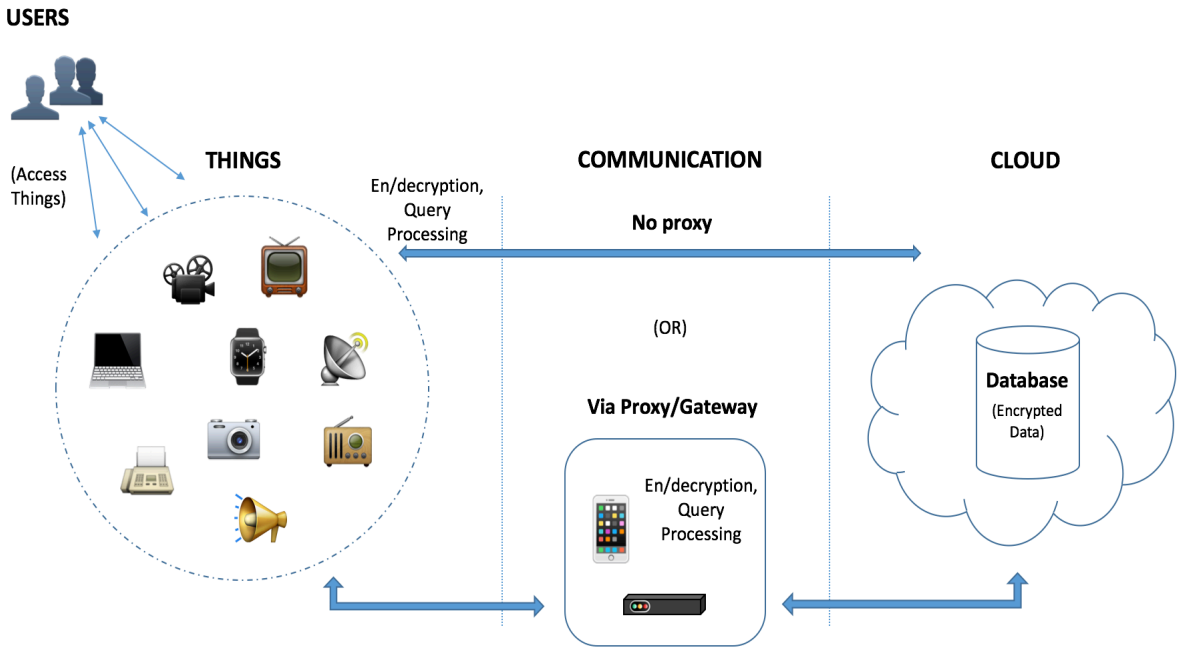


Figure 5. *Conventional Encrypted Query Processing architectures*

In the conventional encrypted query processing schemes proposed so far, whenever a user poses a query, the query is first processed and rewritten to be posed over encrypted data on the Cloud database. The database executes the query and returns the results onto a write-through cache (i.e. put the data in the cache and write it to the target device immediately). Finally, the user device retrieves the results using communication via network. The returned encrypted results are then decrypted (either at the user-side or at an intermediate proxy) and finally made available to the end user. Irrespective of where this encrypted query processing occurs (client-side or server-side [1]

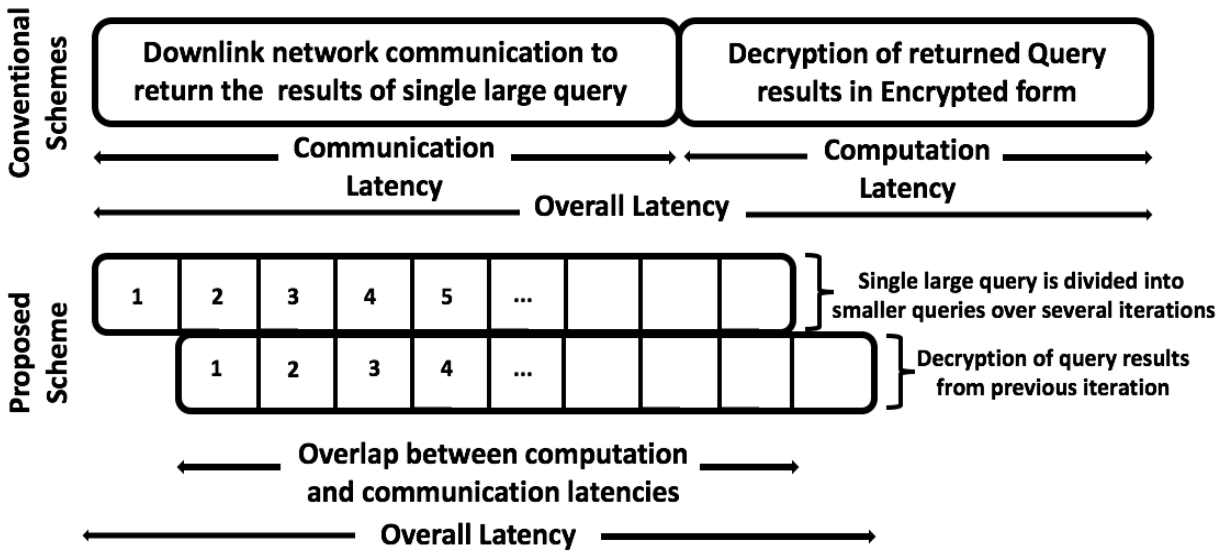


Figure 6. *Illustration of the proposed scheme by comparing with the existing encrypted query processing schemes in terms of computational and communication latency. By retrieving the results in smaller sizes and establishing concurrency between computation and communication, we can expect a significant latency gain.*

or on an intermediate proxy), the end user has to face certain latency before the requested results are made available, if the queried dataset is huge. This latency is broadly classified into: (i) computational latency involved in processing the query before posing on the database and in decrypting the returned encrypted results and (ii) the communication latency which includes the workload of Cloud database in returning the results plus the downlink network communication by the device for fetching the results. For data intensive applications, this latency might be significant and undesirable. To overcome this issue, this thesis work proposes a novel latency-hiding technique to break down the results of a single large query that returns huge data sets, into several small-sized data sets. The key idea is: instead of retrieving the whole data set at once and waiting

till all the data is decrypted, it is intuitive to retrieve small chunks of data and simultaneously perform the computational work (decryption of previously retrieved data) while this data is being retrieved. Thus, splitting the single task into concurrent tasks of computation and communication [see Fig. 6] in several iterations might result in significant savings in terms of latency and energy.

One issue to be addressed however is to decide on the appropriate data size (size of data set) to be requested in each iteration. It plays a crucial role in determining how best this scheme improves latency. For this, the proposed algorithm uses an approach to start with an initial data size and adaptively adjust this size to minimize the gap between computation and communication latencies in each iteration. Based on the experimental observations, depending upon how we choose the initial data size, two variants of this algorithm are proposed: (i) one in which the initial size is a fraction of the original large query size and (ii) the other in which the initial size is fixed, independent of the original query size (for much larger queries, it was observed to have it fixed). This is discussed in more detail in the later subsections.

Underlying Architecture

Figure 7 presents the proposed IoT architecture, that is suitable for meeting the scalability and latency issues while still being comparable to the existing encrypted query processing solutions in terms of energy efficiency. We mainly consider the following components: (i) the IoT device which has the client credentials; The client is any user who is interested in the IoT data, for instance the health information of a patient. The client might have access to a single user-related data or he/she might have access privileges to view the entire database, and (ii) the Cloud database that safely stores the encrypted IoT data coming from the IoT devices, (iii) a cache system that stores the results obtained from executing a user query over Cloud database. The proposed

algorithm will be implemented at the IoT device. These components together are capable of query processing over encrypted IoT data.

IoT data typically includes sensor-readings (e.g., time, location, temperature), any meta-data (e.g., time-stamp), images, audio/video files. Note that the IoT devices are of several types:

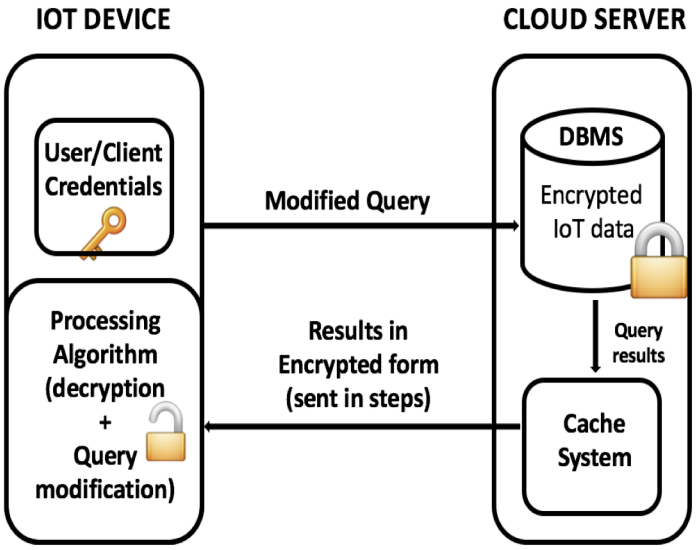


Figure 7. *Underlying architecture of the proposed algorithm: IoT data is stored on the cloud in encrypted form. When the user sends a query, it is executed by the DBMS and the results are stored in a cache. Then these results are returned sequentially in steps. The device establishes concurrency between computation and communication, until all the results are successfully retrieved*

data producing IoT devices, data consuming IoT devices and hybrid devices. In this design, the data consuming IoT devices are considered which potentially have a need to request the IoT data from the Cloud for further processing or to take certain actuation decisions. The amount of IoT data they query can be of any size: from just a few rows to the entire database (if they have the necessary privileges). It is also further considered that a data producing IoT device can be a data consuming IoT device or vice-versa. Whichever is the case, the data is always encrypted before

storing on the Cloud and can be queried as and when required by the IoT devices. Only those clients with the right access privileges (e.g., decryption keys, master secrets) will be able to successfully decrypt the returned encrypted data.

It is known that public cloud systems come with higher security concerns compared to private and hybrid cloud systems [11], by sharing the cloud resources with other, potentially hostile users. So, in this work, it is considered that the IoT data is stored on a public cloud. However, the proposed algorithm can be extended to private/hybrid clouds. Further, it is considered that the Cloud service providers offer their resources under the infrastructure as a service (IaaS) model [12]. In IaaS environments, the cloud service provider delivers resources at the request of users in the form of Virtual Machines and facilitates deployment of application-level caching service as and when required such as memcached [13] using the virtual machines, enabling better utilization of existing resources.

Adaptive Data Size Determination

Implementing query processing over encrypted data involves (i) query encryption and decryption of data which leads to computational latency and, (ii) posing the modified query over the Cloud database and fetching the encrypted results which leads to communication latency. By establishing concurrency between computation and communication through a latency-hiding technique, we try to minimize the latency seen by the user. For this, the algorithm adaptively determines the suitable size of data to be retrieved in each cycle of computation and communication.

To give a better understanding of the need to have an appropriate size of data to be retrieved in each iteration and to unfold the intricacies involved, let us consider a simple use case in which

we have a Cloud database table and a query that results in 10000 records (rows). We have two options to return them to the client: (i) Return all at once and (ii) Return 1000 rows (say) each time using *limit* and load the next 1000 when the user scrolls down. Now the question is, which is a better option? Does the second option really improve latency performance? Does the query execution time depend on the number of rows it has to return? Both options are equally slow *for the database* [14]. Let's explain *why*.

Option 1: Return all at once

This takes up some time to send through the network, and most likely a while before the browser parses the data. 10000 records of data is not really that huge and so it's most likely going to be quick - however, that principle won't hold up if the number of results queried grows.

Option 2: Return in steps

Using *limit* and *offset* for queries over databases like MySQL will not reduce the execution time [14]. Here, what a database does is find all the records satisfying the request, buffer the data, return the requested number (1000) of records and discard the others. It basically does almost the same work as in option 1. When it comes to performance - to let something work *fast* means to make it work less. If we download 2000 rows of data instead of 1000, it means we are doing twice the work (communication) essentially. Hence, in terms of communication, sending 10000 rows at once will be much slower in general because the network will have to send more amount of data, and the client will have to receive more data at once (might include client buffer latencies as well). Using option 2 of sending in small chunks at a time is slow for the database, but rather quick and easy for network/browser. Hence, an important element when it comes to performance is to find satisfying trade-offs. The goal at hand is to have the least work for our server and client. How we achieve it depends on the way we retrieve the information, the architecture while constructing the

application and the underlying algorithms or other infrastructure associated with it.

To address the trade-offs discussed above, in the proposed algorithm a simple rule is followed: query all the results at once and cache the large amount of returned data on the server (using memcached [13]), and pull from that list in several iterations. This way, the Cloud database does the same amount of work as before and thus exhibits the same latency performance (as in Option 1). Thus, the server cache together with the option 2 of sending in small chunks at a time prevents additional database latencies that are foreseen earlier. Another advantage from this is that: one of the heaviest parts of making a query against the Cloud database is actually connecting to the database. Since databases are optimized for data aggregation, returning huge datasets is not tedious for them. Extracting the required data in a single connection and caching it prevents us from adding any extra energy overhead if we try to connect to it multiple times; it also leaves the Cloud database available for other potential clients.

The next goal is to find an appropriate data size for each iteration. For this, an Adaptive Latency Aware Query Processing algorithm is implemented at the IoT device. It considers both the computational delay and the communication delay. The algorithm gives the evaluating procedure to adaptively find an appropriate data size in each iteration, to balance these two delays for concurrent execution.

First, through an availability evaluation, the algorithm (Figure 8) examines whether the Cloud server has available storage resources to cache the results of a user's query request. The algorithm evaluates the Cloud server in terms of the cache storage available to a client by comparing with the storage size of the client request. It is to be noted that cache is used only as a buffer and once the results are returned to the client, the cache is cleared off those results. Hence,

the amount of available cache changes with time, and with the number of queries posed by the logged-in clients at a given instance of time.

Depending upon the number of users currently logged in, every client gets a fair share of the server cache while still leaving room for fresh clients. Let V be the storage size of the server cache that is available for a client at a given instance of time. Let S be the storage size in a user's query request, and R be a total storage capacity of the server cache. Since the user's query requests cannot exceed the storage capacity of the allocated cache at any instance of time, the proposed algorithm should check the cache availability before executing the query. Let j be the number of clients currently logged in (other than our client) and V_i be the share of server cache allocated to i^{th} client. Also, let Δ be the part of server cache that's reserved for future clients. Evaluating the cache resource availability is as follows:

$$S \leq V, \tag{1}$$

$$\text{where, } V = R - \left(\sum_{i=1}^{j} V_i\right) - \Delta$$

If the above constraint (1) is satisfied, the encrypted query can be posed as is, to retrieve all the results (say T rows) at once and cache them for use later on. If this isn't the case, we query for only those results (< T) just enough to fill the available cache storage (V). When this cached list is close to being used up, we need to asynchronously pull more data (next V from remaining S-V) from the database to keep the client going forward.

Now the task at hand is to request the cached results in steps (iterations). In each iteration, the algorithm queries for a certain length of data and simultaneously processes the returned results from previous iteration. For the first two iterations, the algorithm chooses the data size 't' (i.e., the

amount of data requested) as a fraction of the original query size (T), and this fraction is called step-factor (*st*). The algorithm then evaluates the decryption delay time (D) involved in decrypting and rendering the results of previous iteration, and the communication delay time (C) involved in retrieving the results of current iteration. The difference in these two delays is called the *lag* (τ). The equation for this lag is as follows:

$$\tau = |C - D| \tag{2}$$

The goal is to minimize this lag as much as possible. In case if C > D, it is considered that the workload of the IoT device is light and the downloading delay is more. Now by increasing the data size in small steps, we try to see if the *lag* is decreased by any amount. If the *lag* is decreased by increasing the data size, we try to increase the data size further in the next iteration, to see if the *lag* is decreased further. If increasing the data size instead increases the *lag*, we then reduce the data size in steps, from the next iteration. This way, the algorithm adaptively determines a data size for which the *lag* tends to zero (or to a minimal value). The *lag* is continually monitored till the end of the last iteration, to account for any fluctuations in the IoT device CPU utilization (resulting in slowing down decryption) or the device network delays. This way, the data size (t) is adjusted accordingly.

The timing diagram of the Adaptive query processing protocol is indicated in Figure 9. As per the protocol, the client first poses the query which gets executed at the cloud database and the resulting data is stored in the cache list. Next the client chooses an initial step size (t) and requests for results of size (t) in the first two iterations. From the second iteration onwards, the client performs computation and communication simultaneously and it determines the communication delay involved in returning the results of second iteration (C) and the computation delay involved

in decrypting the results of first iteration and computes $\tau = C \sim D$. From the third iteration, it adaptively determines the data size t' to be requested depending upon τ and requests for t' results. It again computes τ for this iteration and the whole process repeats until τ is minimum.

```
begin
    if (cache storage available for a user ≥ required storage for the user query)
{
        Execute the query and store the results on cache list
        Initial iteration data size (t) = (step-factor) * (original query size) OR
                                        = a fixed-value (for larger queries)
        while (remaining cache list is not empty) {
            request for results of size t
            clear the cache list off these returned results of size t
                estimate the communication delay (C)
            estimate the computation delay (D)
            compute τ = | C – D |
            while ( τ not minimal) {
                minimize τ by adjusting t adaptively (either by increasing
                or decreasing t )
                }
        }
    }
    else {
        Query for results size just enough to fill the available cache list
        goto begin
        Asynchronously query the remaining results
    }
end
```

Figure 8. *Pseudo code of the Adaptive query processing Algorithm*

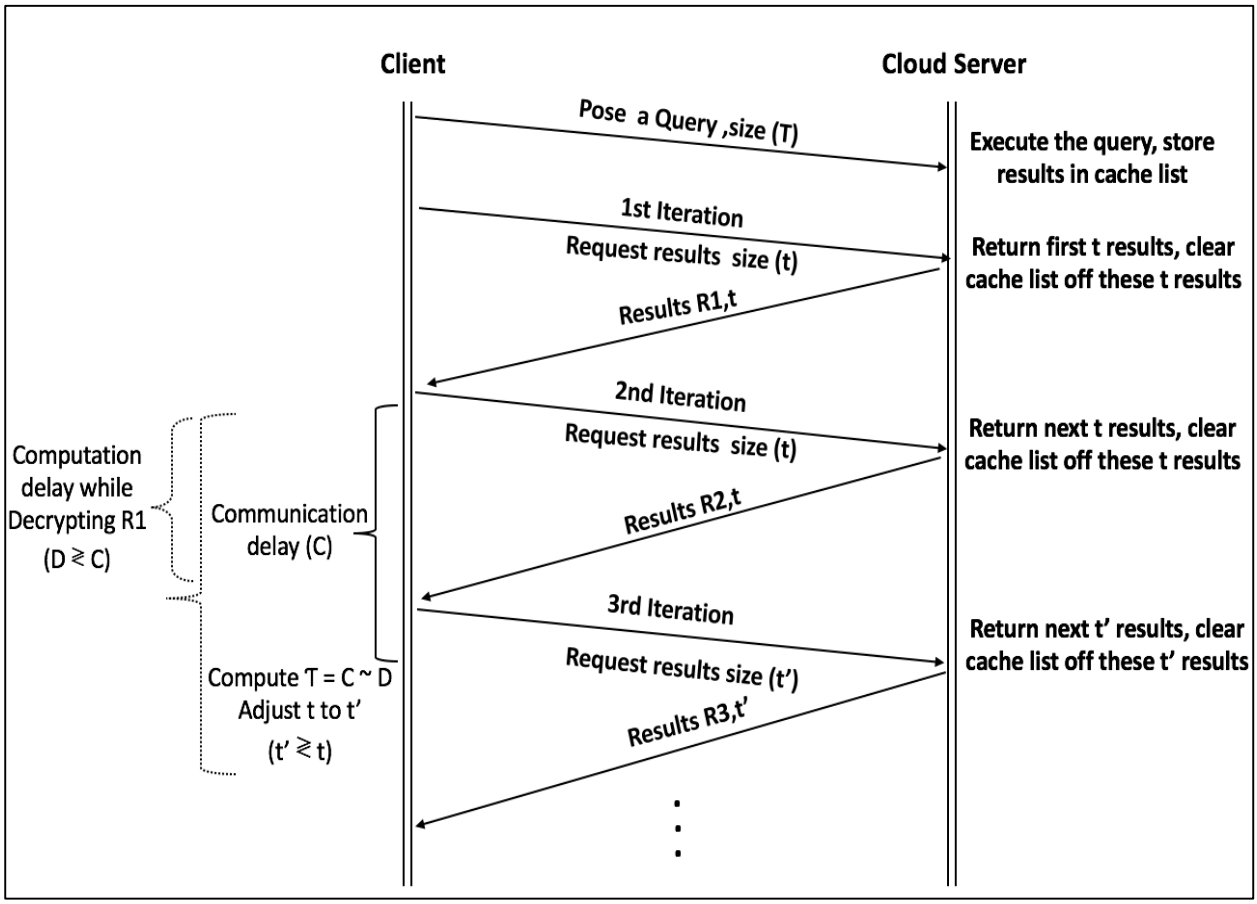


Figure 9. *Timing Diagram of the Adaptive query processing protocol*

CHAPTER 4. PERFORMANCE EVALUATION

Experimental Setup

The proposed algorithm is implemented in an evaluation setup (see Table. 1) that simulates a cloud computing environment adopting the adaptive algorithm at the IoT device. The implementation consists of two main components: (i) An Android Asus Google Nexus 7 that is used as an IoT device capable of processing over encrypted data and (ii) the Cloud database component, which is an implementation of MySQL server with cache enabled. This can cache information in a memory buffer which is tweaked in the algorithm. The IoT data consists of sensitive health record entries, patient monitoring information, and the details of their recent in-hospital treatments, like diagnosis, blood sugar levels, BP, allergies, etc., which includes text (String data), metadata (date, timestamp), and numeric data. This data is encrypted using Advanced Encryption Standard(AES) in Cipher Block Chain mode (AES-CBC) for String data and Order-Preserving Encryption schemes for numeric data and stored on the server database. For evaluation purpose, a database with 10 columns and more than 1 million rows of encrypted data is used, with each row (record) taking around 0.1 KB storage on a MySQL Community Server version 5.6. The current scheme is compatible with any kind of DBMS as long as the appropriate connection driver and cache system are available.

The algorithm is platform independent and can be implemented on any platform. The experiment is performed with the database server running on a Linux Machine with Intel core i5 (2.7 GHz) processor. The system had 8 GB 1867 MHz DDR3 SDRAM and a 121 GB SDD, with 802.11ac Wi‑Fi wireless networking, IEEE 802.11a/b/g/n compatible. The proposed algorithm is implemented as an Android application on Asus Google Nexus 7 device running Android OS v6.0

(Marshmallow) with Quad-core 1.5 GHz Krait CPU and 2 GB RAM. The device is equipped with Wi-Fi 802.11 a/b/g/n, dual-band RF transceivers. The database server and the Android application were on the same local area network. The algorithm adds a negligible memory overhead of 2-3 kB on the IoT device platform with 32-bit memory registers.

Table 1. *Experimental Setup*

Component	Specification used
Client (IoT device)	Device: Android Asus Google Nexus 7
	Processor: Android OS v6.0 (Marshmallow) with Quad-core 1.5 GHz Krait CPU
	Internal Memory: 2 GB RAM
Server	Machine: Linux Processor: Intel core i5 (2.7 GHz) Memory: 8 GB 1867 MHz DDR3 SDRAM and a 121 GB SDD DBMS: MySQL Community Server version 5.6 Cache: MySQL cache enabled
Encryption Algorithm	String data: AES-CBC Numeric data: Order Preserving Encryption
Performance Metrics	Latency, energy consumption
Algorithm(s) evaluated	Adaptive Latency-Aware Query Processing technique; fixed- step querying; no-step querying

Performance Evaluation

The proposed solution is evaluated in terms of delay performance and energy consumption, by posing several queries over the database. To measure the energy consumption, an Android application called Power Tutor from Google Play Store [31] is stored. Several queries are

generated with the result sizes ranging from 10k records to 100k records. A series of experiments were carried out for evaluating and comparing the adaptive algorithm with two other schemes: (i) executing the query and returning the results all at once and processing the results only after all the results are successfully retrieved, which is called *no-step* querying, (ii) executing the query and caching the results, and sending them sequentially in steps with a fixed step-size instead of determining the suitable step-size adaptively, which is called *fixed-step* querying. This step size is considered as a fraction of the original query, which is called *step-factor* (as discussed in section III). This step-factor is varied from 0.01 to 0.5 and the queries are repeated in fixed-step mode. Extensive experimental studies have been conducted to get the latency performance and energy consumption of all these three schemes [see Figures 10, 11 and 12]. The results demonstrate that the proposed adaptive solution outperforms the other two schemes in terms of latency and is comparable to the other schemes in terms of energy consumption.

To further analyze the importance of initial step-size in the latency performance of the proposed adaptive scheme, we rigorously evaluated the performance of the scheme by varying the step-factor that is used to compute the initial step-size. The experiments depicted that: for queries with results size less than nearly 50k - 60k, a higher step-factor of up to 0.2 gives a good latency performance. However, a too-lower step-factor (0.01 or lower) for such queries degrades the performance. This is intuitive because the device would undergo too many iterations unnecessarily. For queries with results size greater than 60k -70k, a higher step factor (of 0.2 or greater) might not show a good performance improvement. This is of course not a limitation of the algorithm, but the Android device doesn't have a sufficient internal cache to store the returned large results (>15k with a step-factor of 0.2 or higher) in each iteration, which it needs to decrypt before rendering on the user interface. Since the IoT devices are resource constrained, adding extra

storage to overcome this issue is not an ideal solution. So this issue is addressed by finely tweaking the algorithm such that when the application internal cache is not sufficient to store the results in each iteration (for larger queries), a fixed-size is used (which is slightly less than the available cache) for the data retrieved in each iteration, that is safe for the application in terms of internal cache. This is called the *fixed-size* mode.

It is to be noted that adaptive algorithm is proposed to address the latency issues in data intensive applications. Hence, for queries with results size less than 5k – 10k records, it is advisable not to use the adaptive algorithm because the actual query itself can execute faster. The advantage of adaptive algorithm can be leveraged for queries with larger results (see Figure 13).

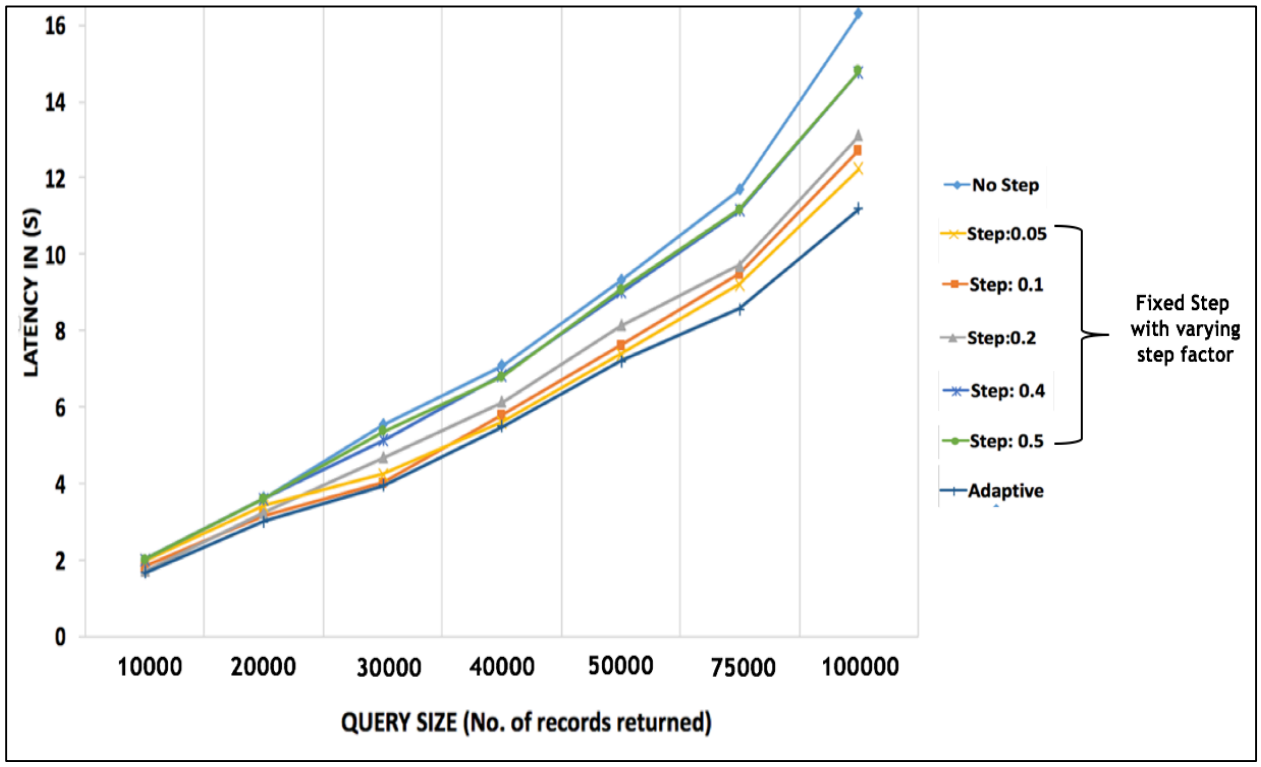


Figure 10. *Comparison of the latency performance of No-Step Vs Fixed-Step Vs Proposed Adaptive Schemes*

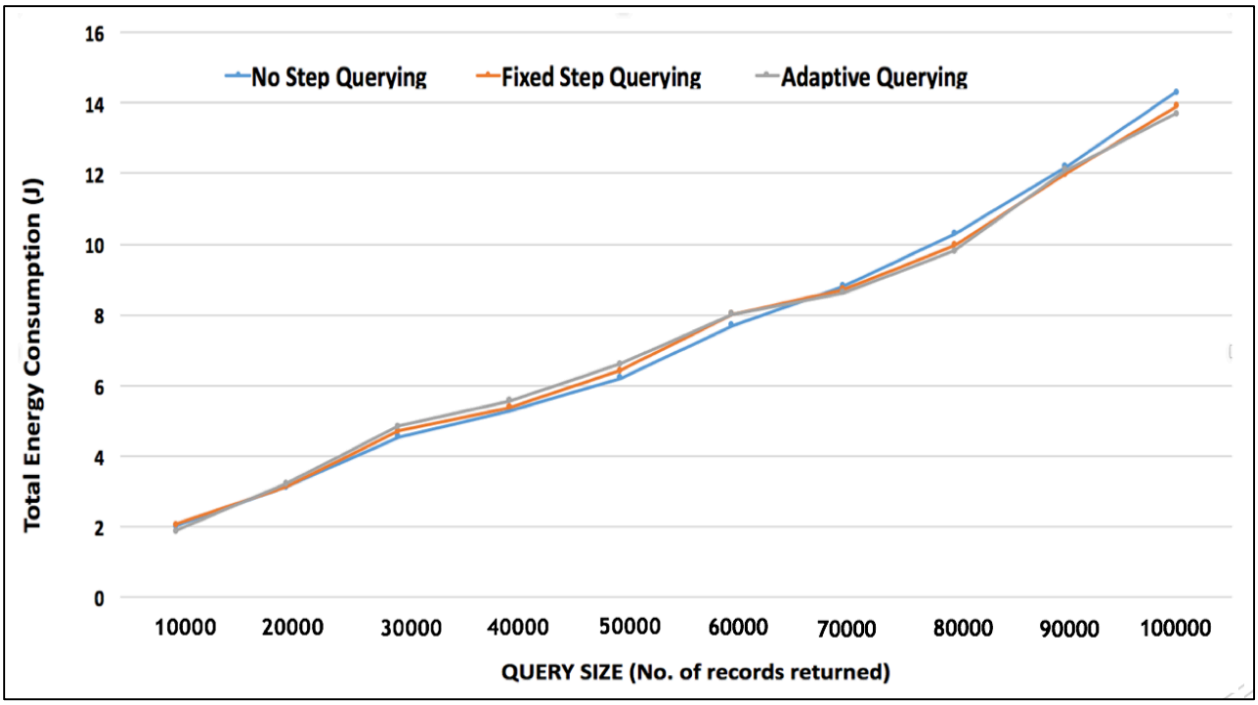


Figure 11. *Comparison of the energy performance of No-Step Vs Fixed-Step Vs Proposed Adaptive Schemes*

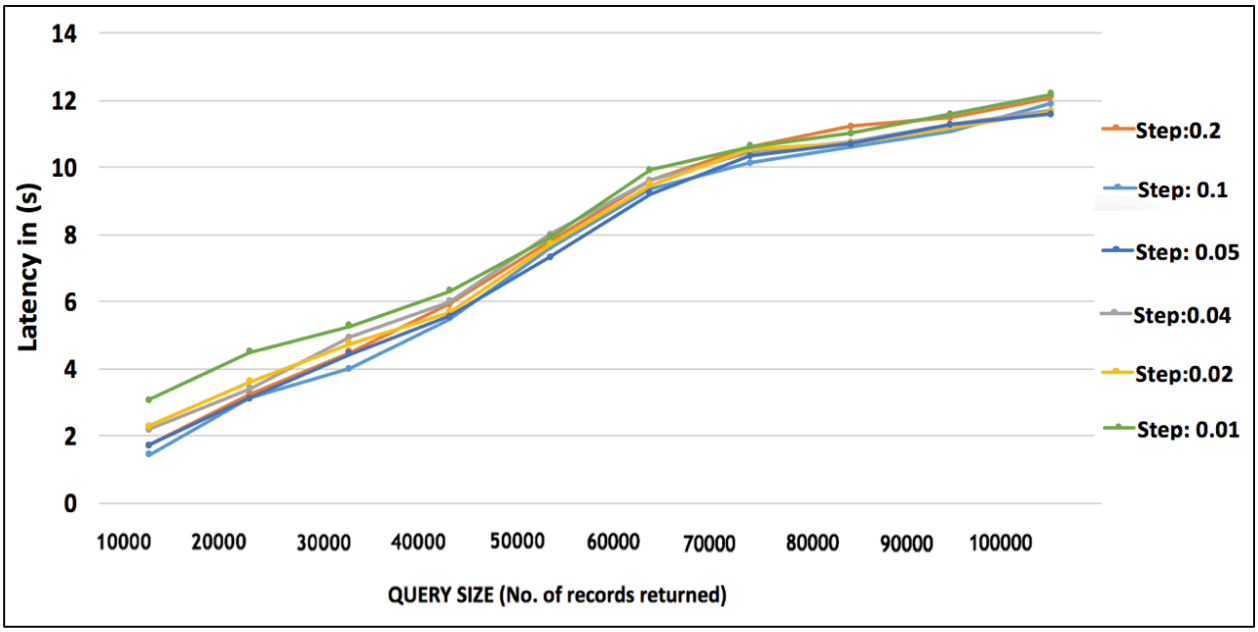


Figure 12. *Comparison of latency performance of the Adaptive scheme by varying initial step factors*

Table 2. Range of *Initial Step-factors for better latency performance of Adaptive latency aware algorithm*

Query Size (No. of rows returned)	**Range of Initial Step – Factors for better Latency performance**
< 10000	No need for Adaptive Algorithm
10000 - 70000	0.05 to 0.2
>70000	0.01 to 0.2

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

Summary

The wide future vision of IoT is to make the things ready to respond to physical events with reasonable behavior, to comprehend and adjust to their surroundings, to gain from, work together and manage other things, and all these are self-driven with or without human interaction. With all the new innovative improvements, significantly more integrated communication between people and machines is made conceivable. Today, IoT devices are everywhere. They are physically integrated into our surroundings and are associated with Cloud administrations for data aggregation and processing. Thus these gadgets are a source of private and sensitive data which could be stolen to deduce security abusing data. As a result, this emerges numerous protection and security worries in the IoT world.

A detailed literature survey is outlined that discusses about the former research addressing the security issues in the Internet of Things. Advancements in the field of security in the Internet of Things appear to be increasingly significant to ensure data privacy and protection. Our work differs from prior work in this domain by addressing the latency issues of IoT applications consuming large sizes of data.

To address the latency issues of data intensive IoT applications, this thesis work proposed a novel IoT architecture that implements a latency-hiding technique to establish concurrency between computation and communication, and an adaptive latency aware query processing algorithm. The proposed solution was successfully implemented using the experimental setup, and the performance of this solution is compared with the no-step variant that implements existing

query processing over encrypted data, and fixed-step variant that implements the concurrent execution solution but with a fixed step-size.

From the experiments, for queries with higher data sizes, the adaptive solution shows a better performance than the other schemes, in terms of average latency. Especially, the proposed solution is shown to be comparable with the existing schemes in terms of energy consumption for queries of any data size. A key point to be noted is that the proposed adaptive solution is advantageous in terms of latency *only* for queries with higher data sizes. For queries with smaller data sizes, there is no need to go for the adaptive algorithm because there wouldn't be a significant latency benefit.

Future Work

This work opens up several avenues for further research in optimizing both latency and energy in secure IoT applications. In our work, we concentrated on data-consuming IoT applications where the data consumer is the data owner itself, trying to retrieve its information stored in the Cloud. So this IoT device is the only component in the entire architecture that has access to all the Master secrets required to decrypt the results.

This research work can further be extended to data-sharing IoT applications in which there would be more than one user involved, with a 3-tier architecture (data-owning IoT device, the Cloud and data-consumer IoT device). In such an extension, the data owning IoT device authenticates the Cloud to transfer data to the data-consuming IoT device. Later the data-owning IoT device can access a third party for key-sharing and frequent key-updates.

Another possible extension could be to use multi-user access control by imposing attribute-based encryption along with the proposed architecture and then perform data-sharing based on user access privileges. This way, a user is allowed to view/edit only those data fields that he/she

has privileges to access. This scheme can be implemented at the Cloud so that it knows which user has access to which data fields in the database and hence the Cloud shares data accordingly.

REFERENCES

1. Privacy Rights Clearinghouse. Chronology of Data Breaches from 2005 to Present Date. Online: www.privacyrights.org/data-breach.

2. R. Hummen, H. Shafagh, S. Raza, T. Voigt, and K. Wehrle. Delegation-based Authentication and Authorization for the IP-based Internet of Things. In *IEEE Sensor and Ad Hoc Communications and Networks (SECON),* 2014.

3. W. Henecka, S. Ko¨gl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-party Computations. In *ACM Computer and Communications Security (CCS*), 2010.

4. Kate Kaye. FTC: Fitness Apps Can Help You Shred Calories – and Privacy. Online: *http: //adage.com/article/privacy-and-regulation/ ftc-signals-focus-health-fitness-data-privacy/ 293080/*, 2014.

5. Y. H. Hwang, J. W. Seo, and I. J. Kim. Encrypted Keyword Search Mechanism Based on Bitmap Index for Personal Storage Services. In *IEEE Trust, Security and Privacy in Computing and Communications (TrustCom),* 2014.

6. R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: Protecting Confidentiality with Encrypted Query Processing. In *ACM Symposium on Operating Systems Principles (SOSP),* 2011.

7. R. A. Popa, E. Stark, S. Valdez, J. Helfer, N. Zeldovich, and H. Balakrishnan. Building Web Applications on Top of Encrypted Data Using Mylar. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI),* 2014.

8. D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Symposium on Security and Privacy*, 2000.

9. D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private Database Queries Using Somewhat Homomorphic Encryption. In *Applied Cryptography and Network Security (ACNS),* 2013.

10. H. Shafagh, A. Hithnawi, D. Simon, Wen Hu.Talos: Encrypted Query Processing in the Internet of Things. *in ACM Conference on Embedded Networked Sensor Systems (Sensys),* 2015.

11. T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09), pages 199–212, Chicago, Ill, Nov.* 2009. ACM Press.

12. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. *Above the clouds: A berkeley view of cloud computing. Technical report, February* 2009.

13. Memcached - a distributed memory object caching system. [Online] January, 2015: http://memcached.org/ [accessed January 31, 2015].

14. S. Kulshrestha, S. Sachdeva. Performance Comparison for Data Storage - Db4o and MySQL Databases In *IEEE Seventh International Conference on Contemporary Computing (IC3),* 2014 .

15. N. Modadugu and E. Rescorla. The Design and Implementation of Datagram TLS. In *Network and Distributed System Security Symposium (NDSS)*, 2004.

16. C. Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2009.

17. A. Liu and P. Ning. TinyECC: A Configurable Library for Elliptic Curve Cryptography in Wireless Sensor Networks. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN),* 2008.

18. P. Szczechowiak, L. Oliveira, M. Scott, M. Collier, and R. Dahab. NanoECC: Testing the Limits of Elliptic Curve Cryptography in Sensor Networks. In *Embedded Wireless Systems and Networks (EWSN),* 2008.

19. M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and Efficiently Searchable Encryption. In *Advances in Cryptology (CRYPTO),* 2007.

20. D. Boneh, C. Gentry, S. Halevi, F. Wang, and D. J. Wu. Private Database Queries Using Somewhat Homomorphic Encryption. In *Applied Cryptography and Network Security (ACNS),* 2013.

21. A. Lopez-Alt, E. Tromer, and V. Vaikuntanathan. On-the-fly Multiparty Computation on the Cloud via Multikey Fully Homomorphic Encryption. In *Annual ACM Symposium on Theory of Computing (STOC)*, 2012.

22. W. Henecka, S. Ko¨gl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-party Computations. In *ACM Computer and Communications Security (CCS),* 2010.

23. D. X. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In *IEEE Symposium on Security and Privacy,* 2000.

24. R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine Learning Classification over Encrypted Data. In *Network and Distributed System Security Symposium (NDSS),* 2015.

25. S. Madden, M. Franklin, J. Hellerstein, and W. Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems (TODS*), 30(1), Mar 2005.

26. A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neill. Order- preserving symmetric encryption. In *Proceedings of the 28th Annual International Conference on the Theory and Applica- tions of Cryptographic Techniques (EUROCRYPT), Cologne*, Germany, April 2009.

27. S. S. M. Chow, J.-H. Lee, and L. Subramanian. Two-party com- putation model for privacy-preserving queries over distributed databases. In *Proceedings of the 16th Network and Distributed System Security Symposium,* February 2009.

28. M. Cooney. IBM touts encryption innovation; new technology performs calculations on encrypted data without decrypting it. *Computer World*, June 2009.

29. E. Damiani, S. D. C. di Vimercati, S. Jajodia, S. Paraboschi, and P. Samarati. Balancing confidentiality and efficiency in un- trusted relational DBMSs. In *Proceedings of the 10th ACM Con- ference on Computer and Communications Security, Washing- ton, DC,* October 2003.

30. T. Ge and S. Zdonik. Answering aggregation queries in a secure system model. In *Proceedings of the 33rd International Con- ference on Very Large Data Bases, Vienna, Austria*, September 2007.

31. Power Tutor - A Power Monitor for Android-Based Mobile Platforms. http://ziyang.eecs.umich.edu/projects/powertutor/-#overview.

32. NIST. Cloud Computing Definition. [Online] Jan, 2015. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf. (accessed January 21, 2015)

33. Leach, P. J., Mealling, M., and Salz, R. A universally unique identifier (uuid) urn namespace.

34. Want, R. Near field communication. Pervasive Computing, IEEE 10, 3 (2011), 4–7.

35. Verdone, R., Dardari, D., Mazzini, G., and Conti, A. Wireless sensor and actuator networks: technologies, analysis and design. Academic Press, 2010.

36. Miorandi, D., Sicari, S., Pellegrini, F. D., and Chlamtac, I. Internet of things: Vision, applications & research challenges. Ad Hoc Networks (2012).

37. W. Hu, P. Corke, W. C. Shih, and L. Overs. secFleck: A Public Key Technology Platform for Wireless Sensor Networks. In Embedded Wireless Systems and Networks (EWSN), 2009.

38. C. Gentry, S. Halevi, and N. P. Smart. Homomorphic Evaluation of the AES Circuit. In Advances in Cryptology (CRYPTO), 2012

39. D. Boneh, G. Segev, and B. Waters. Targeted Malleability: Homomorphic Encryption for Restricted Computations. In Theoretical Computer Science Conference (ITCS), 2012.

40. C. Dwork. Di↵erential Privacy. In International Conference on Automata, Languages and Programming (ICALP), 2006.

41. Google Cloud Database. Security and Integration with Google Cloud. [Online] November, 2015: https://cloud.google.com/sql/ (accessed November 20, 2015).

42. A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal Security with Cipherbase. In Conference on Innovative Data Systems Research (CIDR), 2013.

43. S. Bajaj and R. Sion. TrustedDB: A Trusted Hardware Based Database with Privacy and Data Confidentiality. In ACM Special Interest Group on Management of Data (SIGMOD), 2011.

44. K. Reshma and G. Manimaran: Adaptive Latency-Aware Query Processing on Encrypted Data for the Internet of Things. In 25th International Conference on Computer Communication and Networks (ICCCN), 2016.