Självständigt arbete på avancerad nivå

Independent degree project – second cycle

Huvudområde
Computer Science

Titel

A distributed range query framework for the Internet of Things

Förnamn Efternamn

Congcong Zhang

Mittuniversitetet

MID SWEDEN UNIVERSITY

Abstract

With the rapid development of information technology, applications referring to the Internet of things are booming. Applications that gather information from sensors and affect the context environment with actuators can provide customized and intelligent behaviour to users. These applications have become widely used nowadays in daily life and have initiated the multi-dimensional range query demand referring to the Internet of things. As the data information is fully distributed and the devices like sensors, mobile phones, etc., has limited resources and finite energy, supporting efficient range query is a tough challenge. In this paper, we have proposed a distributed range query framework for Internet of things. In order to save energy costs and reduce the network traffic, we suggest a reporting data range mechanism in the sensing peers, which choose to report a data range and report again only when the peer senses an abnormal data instead of the common moving data method. In addition, we selected some strong peers to be used as the super peers to create a data index by collecting the reporting data range, which will be used for performing range queries. The study has shown that our proposal framework could reduce resource costs in the less strong peers like sensors and mobile phones, and reduce network traffic among all the peers within the network, as well as support a range query function. According the evaluation results, the reporting data range method could greatly reduce the data migration times and save energy costs, and the data index could significantly reduce accessing unnecessary peers and diminish the network traffic.

Keywords: range query, distributed data, Internet of things

Acknowledgements

First of all, I would like to thank my supervisor Prof. Tingting Zhang for her great guidance and advice. She has given me a golden opportunity and inspires me to work in this wonderful project. She also helped me with a lot of research and gave me suggestions about the thesis writing. Without her guidance and encouragement, it would not have been possible for me to write this thesis.

Secondly, I want to express a deep sense of gratitude to Prof. Jiajin Le. He is my supervisor in Donghua University. He introduced me to Mid Sweden University. I am grateful for his constant encouragement during these years.

Finally I would like to express my thanks to Stefan Forsström, Victor Kardeby and others, who have given me valuable assistance, for their generous help.

Table of Contents

Abstract	1
Acknowledgements	ii
Table of Contents	iii
Terminology	v
Abbreviations	v
1 Introduction	1
1.1 Background and problem motivation	1
1.2 Overall aim	2
1.3 Concrete and verifiable goals	2
1.4 Scope	3
1.5 Outline	3
2 Theory	5
2.1 Distributed systems	5
2.1.1 P2P systems	5
2.1.2 SensibleThings Platform	5
2.2 Range queries	6
2.2.1 Range query methods in P2P system	6
2.2.2 Range query methods in WSN	7
2.2.3 Caching algorithm	9
2.2.4 Index data structure	9
2.3 Programming methods	10
2.3.1 Modularization	11
2.3.2 OOP	11
2.3.3 Multi-Thread	12
3 Methodology	13
4 Design and Implementation	15
4.1 Sensing data Module	16
4.1.1 Real world data features	16
4.1.2 Architecture	19
4.2 Distributed Database Module	20
4.2.1 Index mechanism	21
4.2.2 Architecture	23
4.3 Range query process	25
4.4 Application Interface	26

4.5	Communication Module	27
4.5.1	Architecture	27
4.5.2	Main Functions	28
4.6	Platform	29
4.7	Application demonstration	31
5	Results.....	34
5.1	Scenario and test environment.....	34
5.1.1	Scenario	34
5.1.2	Test environment.....	35
5.2	System performance	35
5.2.1	Report Data Efficiency	36
5.2.2	Query Efficiency	37
5.2.3	Distributed Database Efficiency	39
5.3	Scalability	40
5.4	Balance	42
6	Conclusions.....	45
6.1	Ethical issues.....	46
6.2	Future work.....	47
7	References	48

Terminology

Abbreviations

DCS	Data Centric Storage
DHT	Distributed Hash Table
FIFO	First In First Out
GHT	Geographic Hash Table
GPSR	Greedy Perimeter Stateless Routing Protocol
IoT	Internet of Things
LRU	Least Recently Used
MRU	Most Recently Used
OOP	Object-Oriented Programming
P2P	Peer to Peer
WSN	Wireless Sensor Network

1 Introduction

This chapter will introduce the background of the research work and the motivation of the problem. The overall aim, concrete goals and research scope will also be discussed in this chapter.

1.1 Background and problem motivation

With the rapid development of information technology, increasing applications referring to the Internet of things (IoT) [1] are booming. These IoT applications are applied in a vast variety of fields, such as environmental monitoring, health care, military, biological research, factories, security, social contact, smart homes and so on, close the distance between man and the real world. Applications that gather information from sensors and affect the context environment with actuators can provide customized and intelligent behaviour to users, which have become widely used in daily life.

Increasing diverse devices including sensors, actuators, mobile phones, computers, etc., will be linked together to form an Internet of things, which is a large scale network connecting millions of devices. Based on these decentralized environments, data information is fully distributed to each point. It has been a great challenge for IoT applications to collect information from the distributed environments and provide customized services to users, especially as the various information generated by each device frequently changed in real time.

Multi-dimensional range query in this distributed environment is a foundation service used for the IoT applications to support intelligent behaviours to users. For example, if a biologist intends to study the growing environment of various plants in the jungle, he may need the IoT application services to assist him in monitoring the plants' growth and collect the essential growing data information about their growth. Thus, the applications should support some basic functions like multi-dimensional range query to provide service for the biologist to collect the sensed data generated by the sensors placed in the jungle within some given conditions. So the biologist may activate a range query request to ask for the information that contains conditions such as humidity stages and temperature levels ranging from a specified interval.

However, the resource and energy in some devices like sensors and mobile phones are limited and the data information that the users are interested in is commonly a limited set referring to fewer points in the distributed environment. If we do not employ any mechanisms, the user might need to visit a great number of points, even flooding, to find the desired information. This will lead to heavy network traffic and the network will soon be exhausted. There are many papers with suggestions as to how to solve the range query problem. The solutions in these papers always consider migrating the sensed data to some specific places in advance to ensure that similar data information will be stored nearby within the network, so that it could support an efficient range query service. But the cost of data migration could still create heavy traffic if the sensed data is produced frequently. While the case this research focus on is based on a distributed real time environment, thus, data migration is not appropriate. Therefore, since it is difficult to enable an effective multi-dimensional range query in such a distributed environment, this research work will be motivated by this distributed range query problem.

1.2 Overall aim

This research work's overall aim is to find new solutions to solve the multi-dimensional range query problems used for the user applications supporting more types of customized and intelligent services in the area of the Internet of things. As the data information is fully distributed in the whole environment and the things associated within the distributed network like sensors, mobile phones, etc., has limited resources and finite energy, supporting range query with lower resource cost in the less strong peers like sensors and mobile phones is the focus of the solution. Therefore, to support efficient range query based on the distributed environment is a challenge, but this research work will concentrate on this problem and propose a new solution to solve the range query issues for the Internet of things field.

1.3 Concrete and verifiable goals

In this research work, in order to solve the problem, we divide it into six concrete goals and aiming to accomplish these goals one by one.

(1) Read literature referring the range query subject in wireless sensor networks (WSN) and peer-to-peer systems (P2P). Classify and summarize the techniques of solving the problem proposed in these papers. Study and compare the disadvantages and advantages of these

technologies. Find out what has been done well and what should be improved.

(2) Study several distributed systems and consider achievable methods to enable range query based on one system chosen from these distributed systems.

(3) Design the range query framework and find out the functions of every module that constitute the range query system based on the distributed system chosen from the last concrete goal.

(4) Implement these modules that consist of the range query system including sensing data module, distributed database module, range querying module, application interface and communication module.

(5) Verify the range query system based on a demo with several querying nodes, sensing data nodes and distributed database nodes.

(6) Build a scenario and evaluate the report data cost, the range query cost, the index query cost, the scalability and the balance point of the range query system based on the established scenario.

1.4 Scope

This research work focuses on implementing the multi-dimensional range query with lower routing cost in the network for users to locate the interested peers and less energy cost for the producing data peers to report their data information. In the project, the effects of data security, user application design, historical data query, maintaining node failure, and handling the loss of data in failure peers are ignored, because some of these effects, like the data security and user application design, will be developed by other members in our research group, and some like the historical data query, maintaining node failure and handling the loss of data issues will be completed in future work. The research work is distinguished by the evaluation of the range query system performance with data report cost, range query cost, index query cost, the scalability and the balance point based on an established scenario. The outcome of this research will be a functioning framework of the range query system, including all the essential modules with basic functions.

1.5 Outline

The structure of this thesis is organized as follows. Chapter 1 introduces the background and problem motivation of this research work. The

overall aims, concrete goals and scope of this research work are also presented in this chapter. Chapter 2 discusses the theories that were used to complete the research work, including several related range query techniques proposed in a distributed system, several algorithms and several programming methods. Chapter 3 introduces the methodology and methods to accomplish the research work and to achieve each concrete goal. Chapter 4 begins with an introduction of the overall structure of the range query system. Then the design and implementation of the range query system will be described in detail. Finally, the interface of the user application and a range query demo will be presented. Chapter 5 summarizes the comprehensive results and presents the evaluation of the range system, including query performance, scalability and balance issues. Chapter 6 concludes the whole research work and puts forward future directions.

2 Theory

In this chapter, we will discuss the theories that were used to complete the research work, including the distributed system we used to build our framework, several related range query techniques proposed in P2P systems and wireless sensor networks used to reach our proposed solution and several programming methods.

2.1 Distributed systems

A distributed system consists of a collection of networked components that coordinate their actions and share resources as a single, integrated system for their users [2]. In this section, we will discuss several examples of the distributed system containing P2P systems and the SensibleThings platform.

2.1.1 P2P systems

P2P systems are always classified into two categories, unstructured P2P systems and structured P2P systems. In unstructured P2P systems, message flooding could be used to locate a node. It keeps forwarding the message to all the nodes within the network until it finds the node storing the data needed. But, it leads to heavy network load. Gnutella and Napster [3] are examples of unstructured P2P systems. In structured P2P systems, the most common way is to use the DHT to organize the P2P network. Some well known DHT-based systems includes Chord[4], CAN[5], Pastry[6].

2.1.2 SensibleThings Platform

Our lab has for many years been dedicated to an Internet-of-Things project called SensibleThings platform [7,8]. The project's idea is that the SensibleThings platform will connect various things such as actuators, sensors, and mobile phones and so on as one node in the platform in the future. It aims to be a fully distributed system with no centralized node. The SensibleThings platform is composed of several layers and components including an user application component, an interface layer, a sensor and actuator layer, an add-in layer, a dissemination layer and a network layer, as we can see from figure 2.1.

In the platform [7,8], the interface layer is a public interface provided for user application developers to develop applications and for sensors or

actuators to exchange data information through the platform. The add-in layer is designed for user application developers to add specific functionality customized by the developers themselves depending on the specific application desires. The following dissemination layer and network layer is the basement layers of the platform. The purpose of the dissemination layer is to disseminate information among all the entries connecting to the platform, and the network layer aims at handling the different IP network setups involving wireless and mobile.

Figure 2.1 The layered architecture of the SensibleThings platform

In this research work, we will enable a range query function on the SensibleThings platform and build a distributed range query framework based on the layered architecture.

2.2 Range queries

In this section, we will discuss the range query techniques regarding P2P systems and WSNs proposed in the references and illustrate the caching algorithm and data structure used to reach our proposed solution.

2.2.1 Range query methods in P2P system

There has been much research work done on enabling range queries in P2P networks. Some mainly focus on supporting range queries based on DHT [9] logic. Andrzejak et. al. [10] proposed a CAN-based method that extended the DHT functionality, and they use the Hilbert curve to map single dimensional data space to d-dimensional space. Dan W et. al. [11] also proposed an extension method of DHT, which combines DHT method with B+ tree. They use B+ tree to organize the multi-dimensional

data for each attribute and organize the nodes within the network based on the Chord [12]. Cai M, Frank M, Chen J, et al. [13] proposed a scheme called MAAN that supports range query by using a uniform locality preserving hashing method, which is also an extension of Chord. Chang Y I, Wu C C, Shen J H, et al. [14] proposed a structured segment tree method to support range query in P2P systems, which has a better load balancing.

While some researchers proposed schemes supporting range queries based other structures, Shu Y, Ooi B C, Tan K L, et al. [15] proposed a Znet scheme to support range query in P2P systems. They use a partition tree structure to split the data space and assign subspaces to peers and build a skip graph [16] to organize the nodes in the network that will be used for routing events or queries. This approach could guarantee a well-balanced load, because the data space partitioning and node assignment is processed dynamically. Gu Y, Boukerche A, Ye X, et al. [17] proposed a HD tree method aiming at solving the data locality to support range query in P2P systems. They use the space filling curve to partition the data space and the HD tree to organize the nodes within the P2P system to reduce routing costs.

The range query methods in P2P systems mentioned above usually solve the problems through three steps. Firstly, they always have to organize the data space, and the common method is using the space-filling curve. Then, they have to organize the overlay network to locate a node within the P2P network efficiently, and the common method is based on the DHT or using other structures like HD tree and skip graphs. Finally, they have to assign the organized data objects to the structured peer nodes to ensure that similar data will be stored nearly within the network. This is so that they could support range query effectively avoiding accessing unnecessary nodes within the P2P system. However, the cost of migrating data from the producing place to a specified place is still heavy, especially when the node generates data frequently.

2.2.2 Range query methods in WSN

In order to enable range query in wireless networks, there have been many solutions suggested in previous work. Basically, there are three typical data dissemination methods used for range query in the previous work. What is suggested is an External Storage (ES), a Local Storage (LS) and a Data Centric Storage (DCS) [18,19,20]. Every method

has its advantages and disadvantages as well as a suitable application situation. Based on ES every sensor sends data to a base station that is outside the sensor network, while the LS chooses to store the data produced by a sensor node locally at the producing sensor node. However, when we start information retrieval operations like range query, the ES scheme will create a bottleneck in the outside base station and the LS will need flood the query request within the wireless sensor network. As both ES and LS will result in traffic jams in the wireless sensor network, the third schema, DSC, is always used to support range query, which on one hand it also stores sensed data within the wireless sensor network, as well as a definite sensor node according certain rules rather than storing data locally.

There are many different schemes used in DCS which have been proposed in recent years. Geographic Hash Table (GHT) [21] is the first scheme of data centric storage proposed in 2002, which is an original data centric storage method and it is successful for point queries, so while some are probably extensions of GHT, others may be variations of GHT. Unlike GHT, Distributed Index for Multi-dimensional data (DIM) [22] assigns events of similar values into physically adjacent sensor nodes. The scheme [23] establishes a locality-preserving hash to store events and efficiently perform range queries based on the GPSR [24] routing algorithm using a bit code mapping technique, which is also an extended combination of a data centric storage scheme and a DIM scheme. The dynamic load balancing (DLB) [25] is a grid-based approach to achieve load balancing, avoid hotspot of the storage, and improve the data quality of GHT. Load Balanced Data-Centric Storage (LB-DCS) [26] is another scheme that attempts to solve the unbalanced loading problems in the non-uniformly distributed sensor network. Distributed Index for Features (DIFS) [27] provides a workload-balancing method to support range query by using a hierarchical index.

In the DSC schema, the sensed data with the similar values are always stored nearby at the sensor nodes that are closest to each other. Therefore, user queries with specific values could be sent to the sensor node that stores the specified information data directly without flooding queries to unnecessary sensor nodes. It avoids reckless flooding, however, if there is a great amount of sensed data that have similar values are huge, the workload balancing cannot be guaranteed, and the overloaded sensor node may explode.

2.2.3 Caching algorithm

Caching algorithms are always used to manage the caching data stored in the computers or other electronic devices, which contains various examples such as LRU, FIFO and MRU [28]. When the memory is full, the caching algorithms should discard some items to save space for the new items.

LRU stands for least recently used, which is one of the cache algorithms in computer memory management aiming at improving the usability of the system memory [28]. The algorithm has to abandon several items for new ones, when the memory cache is not available as it is full of items. The LRU algorithm is designed to remove the least recently used items primarily. In order to save space for new items, this approach chooses to remove the frequently used items first. However, the LRU algorithm has to track every item wherever and whenever so as to compute the frequency of each item, the cost of computing the frequency is still costly.

In the range query system, the starting query peer is designed to cache other peers that the querying peer often visits. Since there could be thousands of peers in the platform, the querying peer cannot remember every peer it has visited, therefore, some mechanisms are required to make space for caching new peers that the querying peer may frequently visit later on. Thus using this method could guarantee the cache requirements of the querying peer.

2.2.4 Index data structure

There have been many classic data structures that have been proposed in previous research works, structures that are used for searching, including B+ tree [29], AVL tree [30] and segment tree [31].

A segment tree is a kind of data structure built with segments stored in each node of the tree, which is commonly used for handling some problems regarding intervals such as computing the length of a given interval or the numbers of a specified range. A segment tree, actually, is a divide-and-conquer ideology based on tree data structure [31].

A segment tree is also a binary tree, which belongs to a balanced tree structure as well. It organizes the segments into a tree structure, and each node represents a sub-segment. For example, as we can see from figure 2.2, an interval $[1,10)$ has been structured into a segment tree. Each node in the tree represents a segment like $[a,b)$, which is a half-closed and half-open interval. The root node of the tree represents the largest

interval to be handled, and the leaves represent the minimum unit interval $[a, a+1)$. For each non-leaf node, including the root node, is represented by the interval $[s, t)$, so that $m = (s + t) / 2$, then the left side child node represents the interval $[s, m)$, right side child node represents the interval $[m, t)$.

Figure 2.2 An example of segment tree structure

For each segment we could use a bisection method to process the given segment, so that in a tree that has n segments the query complexity could always be limited as $O(\log n)$, and the storage cost of segment tree structure is $O(n \log n)$. The advantage of using segment is the efficient query cost.

In the range query system, we are planning to let the distributed database peer establish a data index for each type of data supported by the platform. Moreover, when the user starts a range query, the distributed database peer will look up the index by an interval required in the query firstly. Thus, the segment tree structure is a good approach to assist the distributed database peer in establishing the data index.

2.3 Programming methods

There have been numerous programming techniques proposed to use for projects, including modular programming [32], object-oriented programming [33] and multi-threading [34]. This section will discuss these programming methods.

2.3.1 Modularization

Modular Programming is a method for software development that divides the software organization into several independent sub-modules according the different functionalities of each sub-module. Therefore, objects with similar functions will be organized into the same module which is a distinct component of the programs [32].

The advantages of this approach is that it could improve the efficiency of the whole software system by reusing the sub-modules and it is convenient for all the developers involved in the project to invoke or extend the related module. Hence, since the modules could be reused, the amount of whole code could be reduced greatly. By diving the entire program into several small parts each developer group takes responsibility for several small parts. Thus, as the related work is focused in one module, each developer group can conveniently perform maintenance on their program such as fixing errors and debugging. The disadvantage of this approach is that it is difficult for project managers to divide it into several logical and appropriate sub-modules.

Implementing the range query system by using this approach can benefit from the above advantages. The range query system should be composed of querying clients, data producing clients and distributed database servers. Every component should communicate with each other and share the communication module. Consequently, this method could be of assistance in structuring the range query system.

2.3.2 OOP

OOP stands for Object-Oriented Programming. It is a general framework consisting of several objects used for designing programs. Conceptually, an object is always grouped of associated behaviours and actually one object is similar to a real object in the real world. Programming languages like Java, C#, C++ and so on are the examples representing this object-oriented framework. For example, the object concept reflected in Java is called class, which is a principal kind of object in Java language. In general, an object is created with some static items and several actions or behaviours [33].

The advantage of designing software with object-oriented language rather than traditional process-oriented programming languages is that it improves the understandability. Moreover, with the fundamental concepts of dynamic dispatch, subtyping, encapsulation and so on, it is useful for maintenance and ease of reusability [33].

By using the OOP idea to implement the program, the extendibility, reusability and maintainability of the range query system could be enhanced. The range query system is actually a kind of extending work belonging to the extension layer under the application layer in the platform. Hence, it is appropriate to guarantee extendibility and reusability of the extension layer with this approach.

2.3.3 Multi-Thread

Nearly all kinds of operating systems support running multi-tasks concurrently, one task can always be called a program, the running program is actually a process in the operating system, which contains multiple execution streams internally and each stream is a sequential execution thread [34].

Multi-threading is similar to the concept multi-process in the operating system, which enables the process to handle multiple tasks simultaneously. Thread is also called a lightweight process, and it is an executing unit to the process. Every thread is an independent and concurrent execution stream in the program, which is exactly the same as the process in the operating system. Application programs with multi-threading can execute several sections concurrently just like multiple independent applications. However, the operating system can scarcely maintain the multi-threading such as scheduling management and resource allocation [34].

The advantage of this approach is that a multi-threaded program can use the shared resources efficiently. The program realized with multi-threading techniques assembled in multiple CPU systems could run faster through concurrent execution. Multi-threading could also allow the program to keep the responsive feature while the main execution is blocked. Therefore, it is convenient for the application to answer the user input whether the main thread is blocked or other tasks are running simultaneously in the background. However, multi-threading require the developers to be more careful with the thread's operation related shared resources to avoid causing deadlocks [34].

In the range query system, the applications should be able to answer several different types of requests at the same time. For instance, the peers who also support sending data has to answer the querying requests while sensing data. Hence, implementing the range query system with a multi-threading mechanism can guarantee the above requirements.

3 Methodology

An appropriate methodology is essential for the research work to achieve the goals mentioned in chapter 1. This chapter will present the methodology used to complete the range query project and complete this research. The research methodologies are described as follows.

(1) In order to achieve the goal of studying previous literature proposing to solve the difficult query problems of the Internet of things, I will research related literature by searching to keywords at the university library and classifying the documents according to the subjects. I will note down the main idea, purpose and contribution of each document, label and highlight the key techniques for solving the problems proposed in these papers. I will review the disadvantages and advantages of the approaches proposed in these papers when one subject paper group has been completed.

(2) To approach the goal of studying several distributed systems, I will search the distributed system documents, learn about the distributed systems through reading, lectures, experiments and study the SensibleThings platform developed by our lab group through reading the references, discussion with the group researchers and attending weekly seminars.

(3) The goal of designing the range query framework will be accomplished by considering the range query solutions based on the disadvantages summarized in the reading literature part. In order to sort out the modules consisting of the range query system, I will use software development methods like modular programming.

(4) To reach the goal of implementing the range query system based on the SensibleThings platform, I begin by learning the platform interface code by practicing sample materials and through discussions with the group researchers, then I will implement all the modules with Java language by using the caching algorithm, index data structure and programming methods like OOP and multi-threading.

(5) The goal of verifying the range query system will be completed by setting up a small-scale network with several meaningful nodes and

testing each module of the range query system by performing significant range queries and exceptional inputs.

(6) To accomplish the goal of evaluating the range query system, I will build a scenario by searching for meaningful real world data from the Internet and assign the real world data to a number of nodes statistically, and then evaluate the performance of the range query system by comparing our solution with other proposed methods and analysing the system stability by using certain indexes.

4 Design and Implementation

This chapter will specify the range query system in detail. Firstly, the overall structure of the range query system will be presented. Secondly, the design and implementation of the range query system will be described module by module. Thirdly, a range query demo will be presented.

Figure 4.1 The overall structure of the range query system

Based on the observations of most real world data features, we proposed a range query framework to solve the problem and built a range query system based on the SensibleThings platform. The overall structure of the range query system is shown in figure 4.1. The range query system is composed of five components containing a sensing data module, distributed database module, range query process, application interface, and communication module. As we can see from figure 4.1, the sensing data module takes charge of the sensors and produces data; we suggested a reporting data range method in the sensing data module, which will be illustrated in detail in section 4.1. For the distributed database module, we will choose strong peers called super peers within the platform to form a complete distributed database and establish data indexes, which will be illustrated specifically in section 4.2. The peers not chosen as super peers will be called normal peers. Based on the sensing data module, the distributed database module and communication module, we could support range queries for user applications through a process, the range query process will be

presented in details in section 4.3, and the communication module will be specified in section 4.5. For the user applications, we provide an application interface illustrated in section 4.4 for user application developers.

4.1 Sensing data module

Based on the observations of most real world data features, we built a range query framework. This section will present the features of the real world data and how can we take advantage of these features to establish the distributed range query framework.

4.1.1 Real world data features

The real world data people commonly need to deal with is always data such as temperature, humidity, noise and air pollution indexes. These kinds of real world data change continuously. We use temperature as the data sample in the following part of this section.

Figure 4.2 Monthly trend of temperature in Sundsvall

Based on the observations and statistical analysis, we could summarize some features of these real world data. Firstly, the data in a certain area may have a small range of variation. For example, the temperature during one day in our house often varies little. Secondly, the data may vary in a wide range, but we could acquire the changing rules. For example, we already know the ranges of winter temperature in the city of Sundsvall. It could be more than minus 20 and less than 5 degree centigrade. Thirdly, in a specific situation, the data changing may have some kind of relationship with parameters like time, geographic

location or something like that. For example, the temperature is related to the seasons, in winter temperature is low, in summer temperature is high.

Figure 4.3 Monthly trend of temperature in Haikou

Figure 4.2 and figure 4.3 present two actual real world data examples. The dataset [35,36] was downloaded from the Yahoo weather forecast website. Figure 4.2 is about the temperature in Sundsvall, which is a relatively northern city in Sweden, therefore the temperature data has a wide range of variation during one year. Figure 4.3 is about the temperature in Haikou, the capital city of Hainan province in South China. It has a mild climate, therefore the temperature has a small range of variation during one year.

No matter whether the temperature changes in a wide range or in a small range, we could acquire the changing regulation from these figures. Figure 4.4 demonstrates the temperature trend in Sundsvall hourly during five days. The hourly data is download from a Swedish meteorological observation website [37]. From figure 4.4 we can observe the fact that the temperature is always in a growing or declining trend during a period, so if we made a prediction, we could learn about a data range in which it is likely to be during a future period.

Figure 4.4 Hourly trend of temperature in Sundsvall during 5 days

We suggest a mechanism called reporting data range method, instead of the previous methods, totally partitioning the data space and totally moving the data from the producing place to a specific place, which is commonly proposed in the references. Based on the observations and analysis, we could study the changing regulation of the data and predict a data range in which it is likely to be in during a future period. Thus, we use this data range to establish a data index for each data type. Therefore, each peer attached to sensors could only report the abnormal data that exceeds the data range last reported. For instance, when the producing data peer senses irregularly data exceeding the old data range last reported, the peer learns that the current abnormal data is under the minimum of the old data range or above the maximum of the old data range. If it is higher than the old maximum, we could predict a new maximum computed by increasing the current abnormal data with an increment and use the current data and the new maximum to form a new data range of growth trend to be reported, otherwise, we could forecast a new data range of decline trend to be reported the same way. In fact, using the current abnormal data directly as the reference for predicting the trend of the changing data is just one way to report a data range of growth or decline trend. We could also employ some algorithms to acquire the new maximum and minimum from the most recent period as the new data range, or we could acquire the new maximum and minimum during the most recent period firstly and then add the growth or decline trend to the new data range.

In addition, according to different characters of data type, distinctive reporting mechanisms could be accepted in the range query system. For

example, the humidity data in a coastal city will always be stable in a relatively higher level; therefore, the reported data range will be updated again during a long time. Another example is if a mobile sensor sensing air pollution data is always employed in two places, one of which has a perfect air quality, and one of which has a bad air quality, the data produced by the mobile sensor will always be in two opposite data ranges. Consequently, we take advantage of the features of different data types and employ a diverse reporting mechanism.

There are various methods used for reporting data range. The critical difference between these reporting methods is the interval size of the data range. The larger interval could greatly reduce the reporting times but increase the number of messages taken by a user query, while the smaller interval could improve the query efficiency by paying the reporting costs.

4.1.2 Architecture

The sensing data module is composed of three components, including caching component, sensing and reporting data component and query process component, as we can see from figure 4.5. The caching component aims at reserving the super peers where the local peer frequently sends report messages. The super peers are selected to form a distributed database, which will be illustrated in detail in section 4.3. The sensing and reporting component is responsible for detecting data, predicting a data range that the local peer most probably could be in during a future period and reporting the data range to the super peers who take charge of this kind of data. The query process component is in charge of answering the range query request sent by the user application and replying to the request with a notification contained in the current data.

Caching the peers frequently used as the report destination is done in order to minimize the access of the default super peer. Each peer will be assigned a default super peer when first joining the platform, which will be illustrated in section 4.6. The normal peer attached to sensors has to send the initial report message enclosing the data type, the maximum value, the minimum value and some related information to the default super peer. If the default peer does not take charge of the data type in the report, messages should be forwarded to the right destination and then the right super peer who is responsible for the kind of data contained in the report will accept the report message and then notify

the reporting peer. Hence, the reporting peer will cache the IP address information collected from the response of the right super peer. Thus, the reporting peer will directly contact the right super which is in charge of the data type that the reporting peer frequently reports and avoid to contact the default peer again.

Figure 4.5 The architecture of a sensing data module

The reporting and sensing data component is the core part of this module. We implement the reporting mechanism suggested in section 4.1.1 in this component, which focuses on reducing the moving data costs from one place to a specific place by predicting a data range that the local peer is like to be in during a future period and reporting the data range to the super peers.

When the peer attached to sensors receives a range query executed by a user application, the peer uses the query process component to answer the range query request and reply the result. This component is to complete the function of selecting the data that satisfied the conditions required in the query and send a notification to the querying peer if it meets the requirements.

4.2 Distributed Database Module

The distributed database is composed of a number of super peers, which are chosen from the normal peers to form the entire distributed database used for storing the data index created by collecting the data range reported by each peer attached to sensors. The building data index mechanism and the architecture of this module will be discussed in this section.

4.2.1 Index mechanism

In the range query system, we suggested that the super peers chosen to form the distributed database establish a data index for each type of data supported by the platform. The segment tree structure is a good approach to assist the distributed database peer in establishing the data index because the data in both report and query are referred to the segment format. Every super peer takes responsibility for one kind of data information or a group of data information, so all the super peers build a global data index together, used to support range queries.

The building and maintenance of a segment tree is composed of three functions including construction, update and search. As the purpose of building the index tree varies, the workflow of each function may change. The three functions in our approach are described as follows.

Figure 4.6 The workflow of constructing a segment tree

Figure 4.7 The workflow of inserting an item

The workflow of constructing a segment tree is shown in the figure 4.6. We built a segment tree for each data type supported by the platform according to the initial report message when one peer begins to report data. The construction of a segment tree uses the recursive method.

The update of a segment tree consists of two operations including inserting item procedures and deleting item procedures. The workflow of inserting an item into the segment tree is illustrated in figure 4.7. The workflow of deleting an item is similar to the inserting of an item. The inserting process also uses the recursive method. Therefore, when the segment tree receives a data range from a reporting peer, the reporting peer's ID will be stored separately in several nodes according to the reported data range through the inserting procedure.

Figure 4.8 An example of a search result [4,6]

The search function is the core part of this index mechanism. When a user application starts a range query, we first inform a peer list that might contain all the interested peers to the querying user by execute this search function. In order to guarantee the accuracy of the peer list, we employ a greedy algorithm searching all the nodes, not only the nodes whose interval contains the query range but also the nodes whose intervals intersect a tiny bit with the query range. Figure 4.8 illustrates an example of the search result. As we can see, if a user desires the range [4,6], all the nodes surrounded by the red curve should be accessed and added into the peer list.

4.2.2 Architecture

The distributed database module is composed of three components, including a synchronization component, building data index component and query process component, as we can see from figure 4.9. The synchronization component aims at keeping all the super peers in touch and guaranteeing that each super peer has the knowledge of the distribution of all the data types supported by the platform among all the super peers by maintaining a global data type table. The building data index component is responsible for collecting reported data information and establishing a data index used for answering query request. The query process component is in charge of answering the query requests sent by the user application and replying the request with a peer list.

Figure 4.9 The architecture of the distributed database module

When the peers attached to sensors report a new data type that no super peer knows, the default super peer assigned to the reporting peer will take this data and establish a tree index for the new data type, then the super peer needs to synchronize the data type with every other super peer to keep all the other super peers updated. When a user application intend to trigger a range query, the user should first of all know what kind of data types are supported by the platform, so the user will first need to perform a querying data type request, therefore the super peer should answer this request by checking the global table stored locally. Note that the synchronization is only used to synchronize the data types themselves, it has nothing to do with the data, and so the synchronization price should be not very excessive.

The building data index component is in charge of the indexes of each data type stored in the local super peer. It maintains the index by constructing, inserting or deleting operations based the reported data type messages and the reported data range messages it has collected. The index mechanisms have already been discussed in detail in section 4.2.1.

The query process component in this module is composed of two procedures, including the querying index procedure and the querying data types procedure. In the querying index procedure, the super peer should look up the index trees it built, finding out the peers might satisfy the conditions in the user query, and notify the querying peer with a peer list computed by checking the index trees. In the querying data types procedure, the super peer should search the global index

table, figure out the data types currently supported by the platform, and notify the querying peer about data type collection.

4.3 Range query process

The range query process is completed through several procedures based on the foundation of the sensing data module, distributed database module and communication module. This section will specify the procedures that were used to accomplish the range query process.

Figure 4.10 An example of the range query process

Usually, an application client like a mobile phone or computer starts a range query. Figure 4.10 illustrates the range query process beginning with a mobile phone. As we can see from figure 4.10, the solid circle represents the normal nodes that are linked with sensors or installed using a query client. The solid square represents the super nodes that were chosen to form the distributed database. Some normal peers attached to sensors frequently generates data in a real time and report data information to the super nodes. Every normal peer will be assigned a default super peer when first joining the platform. When the user with a mobile phone triggers a range query, the query request will be sent to the default super node as a query index request. After contacting the default super peer, the querying peer with the mobile phone could receive a peer list, and will then start queries for all the peers in the peer

list. The peers in the peer list will not answer the query request if it is not within the querying range. Then, the querying peer could receive answers and get the information of interest from the interested peers.

4.4 Application interface

The application interface is designed for user applications. This section will present the architecture of the application interface. It is composed of three components including a resolving user query unit, caching unit and query unit, as we can see from figure 4.11. The resolving user query component is designed to pre-process the user query and translate the request into a uniform format. The caching component is exactly same with the caching part in a sensing data module, which is used to cache the super peers taking charge of the data types that the users are often interested in. The query component is to complete the range query task by executing several operations through two phases including querying index and querying interested data.

Figure 4.11 The architecture of the application interface

The resolving query module has two main tasks. The first is to check the user inputs, including input format and semantic integrity. Next it should translate the query into a standard format and forward it to the next querying component.

The querying component in the application interface contains two phases. In the first phase, the user application needs to figure out the super peers storing the indexes of these data types required in the user request and send query index requests to the super peers, after that it will get peer lists from several super peers. In the second phase, the application interface needs to compute the intersection of all the peer lists and get a final list, then send query data requests to all the peers in

the final peer list. After that it should wait and collect all the replies. Consequently, the data results of interest will be displayed to the users.

In order to guarantee the accuracy of the query results, we also employ a mechanism in the querying component to avoid the results from confusing a situation when the user starts range queries extremely frequently during an intense period. When the user triggers a query, we set a query state including several flags to judge if the currently received message is outdated or not.

4.5 Communication module

The communication module is a station in the range query system, which is used to send messages and receive messages by the peers who join the platform. This section will present the architecture of the communication module and the message types.

4.5.1 Architecture

The communication module of each peer is mainly composed of four components, including two sending message actions and two receiving message actions, as we can see from figure 4.12.

Figure 4.12 The architecture of the communication module

When one side performs an operation like starting a range query or starting an update of the data types and then sending a message, the destination side will receive the operation message and handle it in an

event process. When the event process has ended, the destination side might load the results into a reply message and notify the reply message, the source side will receive the reply message and handle it in a response listener process.

4.5.2 Main functions

According to the architecture of the range query system we designed, the messages fall into eight categories including reporting data type, reporting data range, querying index, querying data, getting global data types, synchronizing data type, forwarding data range, forwarding index query. Among the eight categories, querying index, querying data and getting global data types travel one round trip between two sides. While others travel almost one way between two sides. Figure 4.13 illustrates the various routing paths between different peers.

Figure 4.13 The routing paths between different peers

In figure 4.13, the nodes with the letter N represent the normal peers attached to sensors or the normal peers installed with the application client, the nodes with the letter S represent the super peers chosen from the normal peers to form the distributed database. Figure (b) illustrates the routing path when a peer starts to detect an abnormal item and report the data range again. If the super peer is not in charge of this kind of data it will forward the report to the right super peer, and then the right super peer will send a notification to the reporting peer which will cache the right peer so that the next time it can access the right peer directly. Otherwise, the second routing path could be ignored. And figure (e) is similar to (b).

4.6 Platform

We established the range query system on the SensibleThings platform, which is a fully distributed system with no centralization. All the entities that appeared in the platform will be structured in a circle as shown in figure 4.1. The entity, which is a node of the platform, can be peers like computers, mobile phones, the devices attached with sensors or actuators. We divided the peers into two categories, normal nodes and super nodes, the former is illustrated as a solid circle and the latter is illustrated as a solid square in figure 4.14. The normal nodes are mainly surrounding the peers that are connected to sensors generating sensed data or the peers that are installed with range query clients being able to execute range queries. The super nodes chosen from the normal nodes are stable and strong peers used to form a distributed database storing the data index.

Figure 4.14 Overview of the realized architecture using range query framework

Logically we classify the peers, and in fact every node in the platform is capable of combining the sensing data role, the performing range queries role and the building data index role together when conditions such as attached with sensors or installed with the query clients or being strong and stable are met. Of course, each node could also choose to combine two roles or just the one role. In addition, the nodes could not only have one sensor to produce single dimensional data but could also have several sensors to sense multi-dimensional data. Therefore, the user installed query client could query the data of interest by performing range queries on the platform.

The SensibleThings platform is composed of several layers and components including a user application component, an interface layer, a sensor and actuator layer, an add-in layer, a dissemination layer and a network layer. We built the range query system relying on these basic layers. The add-in layer is designed for user application developers to add particular functionality customized by the developers themselves depending on their specific application desires. Each particular function organized in the add-in layer is established as an extension, which could be caching, security, publish/subscriber or range query. Obviously, the

add-in layer extends the functionality of the platform by offering different services, and the range query is one of these services.

Figure 4.15 The location of the range query system in the layered architecture

The range query system is based on this layered structure including a user application component, an interface layer and an add-in layer. The core of the range query is built as an extension of the add-in layer, as we can see from figure 4.15. In this research work, we enable the platform to support range query functionality by providing a user application interface in the application component, temporally emulating a producing data environment in the interface layer, implementing the core query functionality in the add-in layer.

4.7 Application demonstration

To test the functionality of the range query system, the application demo based on the SensibleThings platform is created. The application demo is only implemented with basic functions in a small-scale network size because other participants in the research group are working on more complicated client applications. The purpose of the application demo is mainly about demonstration and testing.

Figure 4.16 The network structure of the example demo

In this application demo, as shown in figure 4.16, we establish five nodes, three as normal peers attached to sensors, two as super peers chosen to form the distributed database. The sensing data function is emulated and the data is downloaded from the Swedish meteorological observation website [37]. Normal peer one is attached to one sensor sensing the temperature in Sundsvall, which is also installed in the query application client. Normal peer two is attached to two sensors sensing the temperature and humidity in Stockholm. Normal peer three is attached to three sensors sensing the temperature, humidity and wind speed in Kiruna. Super peer one is the default super peer of normal peer one and super peer two is the default super peer of normal peer two and normal peer three.

Initially, the two super peers and normal peer one run first. Therefore, the super peer one will build the temperature index. We have no actual client application, so we emulated a query client as shown in figure 4.17. At first, normal peer one has no knowledge about the global data types, the user needs to update the data types first. If we press the data type button now, we could only get a temperature data type because only peer one is reporting temperature data.

Then we trigger normal peer two and three, so these two peers join the platform and start reporting data. As the humidity and wind speed data type that no super peer recognize, default super peer two will take

charge of these two kinds of data and build the data index. At this time, if we press the update data type button, we could acquire temperature, humidity and wind speed because now the other two peers have joined the platform and started to report these two new data types.

Figure 4.17 The emulated query client

If we enter a larger range and select the temperature data type, we could retrieve three peers' current data. If we select all the data types, we could only search peer three because there is only peer three who has three dimensional data types, we can see this result from figure 4.16.

In the SensibleThings platform, each peer when it joins the platform will register with a UCI, which is almost like an e-mail address used as identity. Therefore, every time we retrieve the peer's UCI and the current value as the returned results.

5 Results

This chapter will discuss the comprehensive results of this research work based on the evaluation of the range query system we acquired from a specific scenario, including query performance, scalability and balance issues.

5.1 Scenario and test environment

To evaluate the range query system, a scenario was first created. Then a large-scale emulated network with thousands of nodes was established. The scenario and testing environment will be discussed in this section.

5.1.1 Scenario

In this scenario, we try to emulate the weather conditions of the country in Sweden from north to south, as we can see from figure 5.1.

Figure 5.1 An example of a range query scenario

The factors contained in this scenario are as follows. Firstly, I download the four cities' weather data from a Swedish meteorological observation

website[37], including Kiruna, Sundsvall, Stockholm and Malmö. And I use four group peers to represent these four areas, every peer in each group start sensing data at a different time, therefore, we could emulate the weather conditions across a large region. Secondly, the weather data contains three dimensions, temperature, humidity, and wind speed, which are organized hourly. Thus, each peer should sense one data item every hour to simulate the scenario. Thirdly, the data I downloaded is the weather data of the year 2013. As the data is recorded every hour, one-year equals 8,000 hours, hence, the data size of each peer is about 8,000 items. Finally, to emulate a large area across the entire country, in this scenario I try to run thousands of peers to represent the four group peers. According to the different concrete testing purpose, we set various quantities.

Based on this scenario, we could use the established application to look for a comfortable city before travelling around in Sweden by performing range queries on these conditions; temperature, humidity, wind speed, air pollution index and so on.

5.1.2 Test environment

In order to test the performance, scalability and balance issues of the range query system, a large-scale emulated network was established. The emulated network will simulate the scenario discussed in section 5.1.2. Because there are not enough physical peers, we modify the parameters in Java virtual machine to increase the capacity for running large amount of peers on one computer.

The computer we used to build the emulated environment is configured as follows.

- Operating system: Windows
- Processor: Intel(R) Core(TM) i7-3770 CPU 3.40GHz
- Installed memory: 8.00G
- System type: 64-bit Operating System

5.2 System performance

This section focuses on evaluating the performance of the range query system based on the scenario we established in section 5.1. We test the

three main modules of the range query system including the reporting data module, query module and distributed database module.

In order to ensure the accuracy of the results, all the queries used to test each performance index will be executed 20 times and each index mentioned in this section and following sections will be computed by averaging the 20 results. In addition, the querying range used in the testing is built on a special value as the range center. The special value is calculated statistically, which is selected from the average of the data source. The method of acquiring the testing query's range is also the same in this section and following sections.

5.2.1 Report Data Efficiency

For the report module, we test the message cost of the reporting data range in the peers attached to sensors and compare it with other moving data methods. The test is based on the sum of reporting messages in 250 peers processing the sensed data as the data size of each peer increases.

Figure 5.2 Comparison between moving data and reporting data

Figure 5.2 shows the difference between the data migration method and data report method. The blue line represents the data migration method's report messages. The red line and the green line represent our reporting data range method. It is true that all the three lines grow as the data size in each peer increases, but the reporting method is growing slowly as the data amount of each peer increases, while the data migration method grew rapidly with a straight line, because every time they sense a data item, they had to migrate it to a specified place.

Actually, in figure 5.2 both the red and the green line are a sum of the report messages plus query messages. The red line adds one query's message cost; the green line adds 250 queries' message cost. We can see that even though we add the querying messages into the reporting data method, there is still a distance between the two methods.

5.2.2 Query efficiency

In the query module, I test the querying cost of performing single dimensional range queries and performing multi-dimensional range queries as the range span in the query request increases with running 250 peers and sensing three kinds of data. The range span means the interval of the range in the user request and always starts to increase from the minimum unit of each data type in the testing.

To test the query efficiency, we propose a definition of utility ratio to evaluate the query cost. The utility ratio (UR) is calculated by this the formula 5.1.

$$UR = 1 - \frac{UM}{TM} \quad (5.1)$$

In formula 5.1, UM stands for the unnecessary messages it takes to perform one range query, TM stands for the total messages it takes to perform one range query. The unnecessary messages means that when the user starts a query, he will get a peer list from one of the super peers, but the peer list is actually not the exact final peers the user expected, there are some peers that are not in the range that the user queried still on the peer list, so some useless messages may be sent to these uninterested peers.

We can see from figure 5.3 that the utility ratio is stable in the range [0.6, 0.8] as the range span in the query request increases. However, it is almost under 0.5 at the beginning, which is a relatively small range span. Because the index we built stored in the super peer is actually an obscure index, not a very accurately index, as we always report a data range instead of an exact value to cut down the reporting times. However, it is true if we employ a mechanism.

Figure 5.3 The utility ratio in a single dimensional range query

As we can see from figure 5.4, it compares the message cost the user application takes to perform a range query between the different numbers of dimensions as the range span increases. The blue line uses only one-dimensional temperature data, the red line contains temperature data and humidity data, while the green line combines temperature, humidity and wind speed data. The 3-dimensional queries need the smallest number of messages, because more dimensions means more querying limits, thus, fewer peers satisfy all the limits.

Figure 5.4 The message cost among multi-dimensional range queries

Figure 5.5 The utility ratio in multi-dimensional range queries

Figure 5.5 shows the utility ratio among multi-dimensional queries. The three lines shown in the figure seem to be same and the utility ratio is still stable in the range [0.6,0.8]. In this figure, the range span begins with 4 scale, because if we start testing in a small range, there will be no peers satisfying three dimensions and also in such a small interval.

5.2.3 Distributed database efficiency

In the distributed database module, I test the time cost for searching the peers satisfying the range required by the query request, which is the response time of the index tree we built. The testing is based on the performing one-dimensional range queries as the range span in the query request increases by running 250 peers. The meaning of the range span was mentioned in section 5.2.2.

As shown in figure 5.6, we can see that as the range span grows, the response time increases in a relatively slow range. Because the larger range span means more results, it takes time for the index to look up the results. In the testing, I use the nanosecond as the time unit, but it is converted into microsecond in figure 5.6.

Figure 5.6 The index time cost of performing a range query

5.3 Scalability

Scalability is the capability of the system to handle a growing amount of peers. In the scalability testing, I used 250 to 1,500 peers and tested three kinds of range spans as the network size increased. The range span used in this testing has a relatively average interval. Broad range or narrow range is rarely used and make no sense in real world.

Figure 5.7 The message cost of three kinds of range spans

As shown in figure 5.7, we can see that the number of messages does not grow very rapidly as the network size increases. The green colour definitely takes most message counts, because it performs a larger range. In this scenario, we assume that the user will query the global

information. In fact, we could support a large number of data types in the SensibleThings platform. Therefore, the user may be interested in few types of data, which means that the index stored in the super peers could help the users to avoid visiting unnecessary peers to a greater extent.

Figure 5.8 The utility ratio of the three kinds of range spans

As shown in figure 5.8, we can see that the utility ratio is still stable, above 0.6. We can also see that the green curve with the larger range span is always above the blue curve with the smaller range span. Even if it is not very obvious we can still conclude that the utility ratio depends on the interval of the range span. The larger the range span, the higher the utility ratio. Thus, it also proves that the index we established is not a very precise index, which is not suitable for exact queries.

As we can see from figure 5.9, the response time of the index also grows in a slow range as the network size increases.

Figure 5.9 The index time cost of the three kinds of range spans

5.4 Balance

The balance test in our proposed range query framework is to find the balance point between using a different size for the reported range. It depends on the data amount in each peer and the querying frequency.

Figure 5.10 Difference between reporting cost and querying cost

As we can see from figure 5.10, it compares the reporting messages cost with the querying messages cost. The blue line represents the reporting times of one peer and the red line represents the query messages it needs to take for performing only one range query. In this figure, the querying cost is far under the report cost, especially as the data size of each peer increases. This provides room to reduce the report times.

The reporting frequency depends on the prediction mechanism we used to report the sensed data. If we adjust the mechanism by extending the reporting data range, the reporting cost will decrease but the querying cost will increase.

Figure 5.11 Difference between report costs with different range spans

Figure 5.12 Difference between query costs using different range spans

As we can see from figure 5.11 and figure 5.12, they compare the message cost between using different sizes for data range. In figure 5.11, as the data size of each peer increases, obviously the large range we used for reporting takes the lowest number of messages. In figure 5.12, as the network size increases, if we use a larger range for reporting, the number of querying messages will rise a little faster than the smaller reporting range. But figure 5.11 and figure 5.12 use different scales as

the message count unit, the maximum count in figure 5.12 is almost only one scale in figure 5.11. Therefore, we could conclude that if we report a larger range, the reporting times will be reduced greatly, while the querying messages will increase relatively slowly.

Figure 5.13 The balance points of reporting different sizes of data range

As we can see from figure 5.10, the querying cost is far under the report cost, thus, it provides room to extend the reporting data range to reduce the report times. Therefore, if we start to catch the balance point with a small reporting range, it will take a long time to find the first point. Hence, we start by designing an original reporting method with which we report the maximum and minimum as a data range of the latest ten values the peer sensed. Then, each time we add a few values to the maximum or minimum to form a new reporting data range. Based on this reporting mechanism, the balance point between the querying cost and reporting cost we got is shown in figure 5.13. In this figure, as the reporting range span increases, for a different size of data in each peer, the balancing point is changed. The red line is the query messages as the reporting range span increases; others are reporting messages of different size of data. We can see that for the 2,000 data size, we could choose the 0.5 range increment for the reporting data range, for ten thousand data, we can choose the 2.5 range increment.

Reaching these balance points is only from starting one range query, however, it is still very close with how frequently the user may want to start queries. In future work, we could make the peers automatically adjust the reporting range according to the number of queries received during a current period.

6 Conclusions

With the rapid development of information technology, increasing applications that gather information from sensors and affect the context environment with actuators have initiated the multi-dimensional range query demand referring to Internet of Things. As the data information is fully distributed and the devices like sensors, mobile phones, etc., has limited resources and finite energy, supporting efficient range query is a challenge. The objective of the research project is to find new solutions to solve the multi-dimensional range query problems. We proposed a distributed range query framework for the Internet of Things and developed a prototype system based on the SensibleThings platform using the proposed framework. In this research work, in order to solve the problem we divided it into six concrete goals and complete the goals one by one.

Firstly, we set reading related literatures as a goal. Therefore, we have learned from dozens of documents referring the range query subject in WSN and P2P and summarized the disadvantages and advantages of these methods proposed in the previous literatures. Consequently, we have come to understand that if the peers generate data frequently, the moving data method commonly used in the references will increase the network traffic and lead to excessive energy costs in less strong peers.

Secondly, we consider the goal of studying several distributed systems. Thus, we have learned about several P2P systems and the SensibleThings platform and decided to enable a range query on the SensibleThings platform.

Thirdly, we aim to design the range query architecture. Therefore, we figured out that the range query system is composed of five components containing a sensing data module, a distributed database module, a range query process, an application interface, and a communication module.

The fourth goal is to implement the range query system and all the modules we classified in the last goal. Consequently, we have completed the implementation and built a range query system on the SensibleThings platform.

The fifth goal is to verify the range query system based on a demo. Therefore, we have tested the range query system by establishing a demo with several functioning emulated nodes.

The last goal is to build to a scenario and evaluate the range system. Accordingly, we have emulated the weather conditions of the country of Sweden from north to south, tested the report data cost, the range query cost, the index query cost, the scalability and analysed the balance point of the range query system based on the established scenario.

According to the evaluation results, it indicates that the reporting data range method suggested in our proposal framework could reduce resource costs in the less strong peers like sensors, mobile phones and reduce network traffic among all the peers within the network, as well as support the range query function. Comparing with the moving data method proposed in the previous papers, the moving data or reporting data range times is reduced greatly. In addition, with the reporting data range method, when the user perform a range query, the utility ratio is always stable in a relatively higher range, which means that it could avoid accessing unnecessary peers and reduce the network traffic.

6.1 Ethical issues

The research project we conducted on the SensibleThings platform will enable a multi-dimensional range query function for the applications developed within the platform and provide better features for the services of the platform. These applications could be applied in a vast variety of fields such as environmental monitoring, health care, biological research, social contact, smart homes and so on. It creates a bridge between man and the real world. The applications with a range query function could provide more customized and intelligent behaviour to users, which can be widely used in our daily life and improve the quality of life for people. The range query system with the framework proposed in this research work could reduce the communication cost and save battery life and energy resources, which benefits to the environment.

In order to prevent the misuse of the system, our research group have established a security mechanism to protect the privacy of users. When a user intends to join the system, it should pass a strict authentication and encryption process, which ensures the security of the related data distributed in the network.

6.2 Future work

Our future work will continue with exploring the issues regarding supporting range query in a distributed environment and deal with improvements for the proposed framework.

Firstly, the approach used for choosing and maintaining super peers needs to be improved. Currently, we assume that there are several super peers and these are designed so that one super peer takes charge of one data type or several data types. In the future, we should employ mechanisms to select the super peers efficiently and make such developments that several super peers could take only one kind of data together to balance the workload of each peer.

Furthermore, we could develop a mechanism to control the whole network traffic. According to the evaluation results in the section 5.4, in the future, we could make the peers automatically adjust the size of the reporting range according to the number of queries received during a current period to control the network traffic.

Moreover, the query of loss data and historical data could be supported in the future. Presently, in the range query system we provide the service of querying the current data in each peer. In the future, we could consider exploiting a lightweight database in the strong peers for data backup, which could assist the system to avoid data loss and support historical data query.

Last but not least, in this field there is still a lot of difficulties and challenges that need to be addressed. In the future, a lot of research could be carried out regarding these issues.

7 References

- [1] Weber R H, Weber R. Internet of Things[M]. Springer, 2010.
- [2] Coulouris G F, Dollimore J, Kindberg T. Distributed systems: concepts and design[M]. pearson education, 2005.
- [3] Saroiu S, Gummadi K P, Gribble S D. Measuring and analyzing the characteristics of Napster and Gnutella hosts[J]. Multimedia systems, 2003, 9(2): 170-184.
- [4] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications[C]//ACM SIGCOMM Computer Communication Review. ACM, 2001, 31(4): 149-160.
- [5] Ratnasamy S, Francis P, Handley M, et al. A scalable content-addressable network[M]. ACM, 2001.
- [6] Rowstron A, Druschel P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems[C]//Middleware 2001. Springer Berlin Heidelberg, 2001: 329-350.
- [7] Forsström S, Kardeby V, Walters J, et al. MediaSense--an Internet of Things Platform for Scalable and Decentralized Context Sharing and Control[C]//ICDT 2012, The Seventh International Conference on Digital Telecommunications. 2012: 27-32.
- [8] Forsström S, Kanter T. Enabling ubiquitous sensor-assisted applications on the internet-of-things[J]. Personal and Ubiquitous Computing, 2013: 1-10.
- [9] Kaashoek M F, Karger D R. Koorde: A simple degree-optimal distributed hash table[M] //Peer-to-peer systems II. Springer Berlin Heidelberg, 2003: 98-107.
- [10] Andrzejak A, Xu Z. Scalable, efficient range queries for grid information services[C]//Peer-to-Peer Computing, 2002.(P2P 2002). Proceedings. Second International Conference on. IEEE, 2002: 33-40.

- [11] Dan W, Maozeng L. A range query model based on DHT in P2P system[C]//Networks Security, Wireless Communications and Trusted Computing, 2009. NSWCTC'09. International Conference on. IEEE, 2009, 1: 670-674.
- [12] Stoica I, Morris R, Karger D, et al. Chord: A scalable peer-to-peer lookup service for internet applications[C]//ACM SIGCOMM Computer Communication Review. ACM, 2001, 31(4): 149-160.
- [13] Cai M, Frank M, Chen J, et al. Maan: A multi-attribute addressable network for grid information services[J]. Journal of Grid Computing, 2004, 2(1): 3-14.
- [14] Chang Y I, Wu C C, Shen J H, et al. Range queries based on a structured segment tree in p2p systems[C]//Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on. IEEE, 2010: 91-99.
- [15] Shu Y, Ooi B C, Tan K L, et al. Supporting multi-dimensional range queries in peer-to-peer systems[C]//Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on. IEEE, 2005: 173-180.
- [16] Aspnes J, Shah G. Skip graphs[J]. ACM Transactions on Algorithms (TALG), 2007, 3(4): 37.
- [17] Gu Y, Boukerche A, Ye X, et al. Supporting multi-dimensional range query in HD Tree[C]//Distributed Simulation and Real Time Applications (DS-RT), 2010 IEEE/ACM 14th International Symposium on. IEEE, 2010: 71-78.
- [18] Ahmed K, Gregory M A. Techniques and challenges of data centric storage scheme in wireless sensor network[J]. Journal of Sensor and Actuator Networks, 2012, 1(1): 59-85.
- [19] GAO C, HU X, GUO L, et al. A Data Dissemination Algorithm using Multi-Replication in Wireless Sensor Networks[J]. no, 2013, 1: 91-93.
- [20] Gao C, Wang B, Hu X, et al. An Efficient Index-based Data Storage Method for Wireless Sensor Networks[J]. Information Technology Journal, 2011, 10(10): 1934-1941.

- [21] Ratnasamy S, Karp B, Yin L, et al. GHT: a geographic hash table for data-centric storage[C]//Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications. ACM, 2002: 78-87.
- [22] Liao W H, Chen C C. A multi-dimensional data storage algorithm in wireless sensor networks[C]//Network Operations and Management Symposium (NOMS), 2010 IEEE. IEEE, 2010: 854-857.
- [23] Li X, Kim Y J, Govindan R, et al. Multi-dimensional range queries in sensor networks[C]//Proceedings of the 1st international conference on Embedded networked sensor systems. ACM, 2003: 63-75.
- [24] Karp B, Kung H T. GPSR: Greedy perimeter stateless routing for wireless networks[C]//Proceedings of the 6th annual international conference on Mobile computing and networking. ACM, 2000: 243-254.
- [25] Liao W H, Shih K P I, Wu W C. A grid-based dynamic load balancing approach for data-centric storage in wireless sensor networks[J]. Computers & electrical engineering, 2010, 36(1): 19-30.
- [26] Albano M, Chessa S, Nidito F, et al. Dealing with nonuniformity in data centric storage for wireless sensor networks[J]. Parallel and Distributed Systems, IEEE Transactions on, 2011, 22(8): 1398-1406.
- [27] Greenstein B, Ratnasamy S, Shenker S, et al. DIFS: A distributed index for features in sensor networks[J]. Ad Hoc Networks, 2003, 1(2): 333-349.
- [28] Silberschatz A, Galvin P B, Gagne G. Operating system concepts[M]. Wiley, 2013.
- [29] Ullman J D, Garcia-Molina H, Widom J. Database systems: the complete book[M]. Upper Saddle River: Prentice Hall, 2001.
- [30] Jin-ling L, Hong Z. Study of the AVL-tree index range query based on P2P networks[C]//E-Business and E-Government (ICEE), 2010 International Conference on. IEEE, 2010: 1699-1702.

- [31] Bentley J L. Solutions to Klee's rectangle problems[R]. Technical report, Carnegie-Mellon Univ., Pittsburgh, PA, 1977.
- [32] Gosling J, Joy B, Steele Jr G L, et al. The Java Language Specification[M]. Addison-Wesley, 2013.
- [33] Lewis J, Loftus W. Java software solutions: foundations of program design[M]. addison-Wesley, 1998.
- [34] Lea D. Concurrent programming in Java: design principles and patterns[M]. Addison-Wesley Professional, 2000.
- [35] <https://weather.yahoo.com/sweden/vasternorrland/sundsvall-906742/> (2014/5/05) "Monthly trend"
- [36] <https://weather.yahoo.com/china/hainan/haikou-2162779/> (2014/5/05) "Monthly trend"
- [37] <http://opendata-download-metobs.smhi.se/explore/> (2014/3/15) "open data"