Keio University Master's Thesis Academic Year 2009

SensingCloud: Open and Global Sensor Network using Distributed Aggregation Mechanism

Keio University Graduate School of Media and Governance

Naoya Namatame

Abstract

SensingCloud: Open and Global Sensor Network using Distributed Aggregation Mechanism

Summary

With rapid development of both hardware and software technologies, sensing devices are now dramatically smaller and able to communicate each other, which technology is known as Sensor Network. Sensor network has made computer systems be aware of the real world. This kind of computer systems can behave without explicit human input. However, current sensor network is not a common infrastructure like the internet. Sensor Networks are distributed all over the world and they are closed to the others. This situation not only decreases development efficiency but also restricts system scope.

As real-world sensing becomes more and more important in ubiquitous systems, there is an urgent need for sensor network to be a common infrastructure. The goal of this research is to create a platform that liberates data streams in currently closed sensor networks and enables the other applications to utilize them.

We propose "SensingCloud", which is a platform that interconnects sensor networks distributed over the world. By accessing SensingCloud, ubiquitous applications can collects sensor feeds from the membership sensor networks. In building SensingCloud, we have organized three requirements: "transparent connectivity to distributed sensor network," "meta-level access to arbitrary data" and "automatic scalability for growing number of sensors." SensingCloud automatically scales the system as it grows, by leveraging P2P communication to distribute its network and task loads. We will propose a new concept vision for sensor network as well as efficiency of sensor feed aggregation with an implementation.

Keywords:

1 Sensor Network 2 Cloud Computing 3 Distributed Computing 4 Peer to Peer 5 Ubiquitous Computing

Keio University Graduate School of Media and Governance

Naoya Namatame

修士論文要旨　　2009 年度 (平成 21 年度)

SensingCloud：
開放型分散センサネットワーク集約機構の設計と実装

論文要旨

近年のハードウェア，およびソフトウェアにおける発展により，小型なセンシング機器が相互通信を行い，高粒度の実空間データをコンピュータシステムに取り込むことができるセンサネットワーク技術が台頭した．この技術により，コンピュータシステムが実空間の状況を把握し，その場に適したサービス提供を行えるようになり，また，これまで人間が取得することが出来なかったデータの観測が可能となった．しかし，現在のセンサネットワークはインターネットのようにインフラ化されておらず，個々の閉鎖的なネットワークが世界中に散在している状況である．既存のセンサネットワークの利用が許されないため，サービス開発者は，センサネットワークの設置からすべてを行わなければならないため，開発効率の低下だけでなく，サービス自体が限られた範囲内でのみ動作する限定的なものになってしまう問題がある．今後，コンピュータシステムにとって実空間情報の考慮が必要不可欠になってゆくにつれ，センサネットワークの共有インフラ化が非常に求められる．本研究の目的は，現状，世界中に散在しているセンサネットワークのデータストリームを，どのようなアプリケーションからでも利用可能にするプラットフォームを構築することである．

本論文では，私たちは世界中に散在するセンサネットワークのデータストリームを共有するためのプラットフォームである SensingCloud を提案する．SensingCloud は，ユーザアプリケーションにより位置情報、およびセンサの種類をクエリとしたリクエストを受け付けると，該当する地域に存在するセンサのデータストリームを集約して転送する．本研究では，このプラットフォームの要件を，“分散センサネットワークのための透過的アクセス”，“メタ情報を利用したデータアクセス” そして “規模に応じた自律的スケーリング” とし，これらの要件を満たすアーキテクチャを設計する．具体的には P2P，および分散コンピューティング技術を用いて，ネットワークトラフィック、および集約にかかる計算コストの分散を図る．

本研究を通じて，私たちは次世代のセンサネットワークをあり方を，その効果的な手法と実装と共に提唱する．

キーワード

1 センサネットワーク　2 クラウドコンピューティング
3 分散コンピューティング　4 ピア・ツー・ピア
5 ユビキタスコンピューティング

慶應義塾大学大学院 政策・メディア研究科
生天目 直哉

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter introduces background information of the field of sensor network and describes a research motivation and a research goal.

1.1 Background and Motivation

Sensing real-space information has become very important in ubiquitous applications. Context-aware systems utilize real-space information to provide an appropriate service to appropriate users at appropriate timing. Also, various fields of researches or services leverage the contribution for detailed environmental analysis or human activity recognition using real-space information. These services that provide new value to the world are contributions of Sensor Networks. This technology shifts the existing computer system to the next paradigm.

Although sensor networks proved such a great potential in computer system society, they have not yet been established as true "infrastructure". Infrastructure should be open and able to be accessed by any users just like the Internet. On the other hand, current sensor networks are closed and still remain as private "tools". In current situation, one can utilize a sensor network only if he/she owns it.

In such situation, the applications are isolated in the private network. For example, a life-log application can only subscribe to and log environ-

ment sensor data at home. Once you go outside, environment sensor data can not be subscribed to. The user wants this application to collaborate with environmental sensors anywhere he/she goes to. Also, a situation on current sensor network systems prevent people to have global awareness. For instance, although thermometers are installed everywhere in Tokyo city, one can only acquire data from a very tiny number of sensors. If we can subscribe to data from these thousands of millions of thermometers, very detailed data of heat-island phenomenon can be acquired and effective action against heat-island phenomenon can be executed.

We envision an open and global sensor network, in which users can subscribe to data from arbitrary nodes in sensor networks distributed all over the world. This platform provides people a global awareness. It enables world wide detailed environmental studies such as earthquake or global warming monitoring. Also, on this platform, applications are not restricted to run on a single sensor network any more. For instance, Autonomous-traveling robots can subscribe to their surrounding environmental information anywhere they go. Under our platform, these robots can know various situations such as a crowd or an amount of traffic. Other than these examples, open and global sensor network platform has potential to enhance life log as mentioned above, entertainment, social security, social studies, and navigation/recommendation system as well.

1.2 Research Goal

The goal of this research is to liberate data streams in currently private and closed sensor networks and enable for applications to utilize them. For this purpose, we will develop an open and global sensor network platform that interconnects distributed sensor networks. This platform allows applications to utilize sensor data stream from arbitrary sensor networks. This provides applications a world wide mobility and a global awareness. We will propose the architecture that enables this goal with high performance and practically deployable way.

1.3 Thesis Organization

This thesis is organized as follows. Chapter 2 introduces the field of current sensor network systems and organized its architecture. Chapter 3 organizes related projects into 3 categories in architecture, and discussed the reason that we are proposing a new system. Chapter 4 describes SensingCloud and approaches that SensingCloud adopts. Chapter 5 introduces the implementation of SensingCloud. Chapter 6 evaluates the performance of the implementation with comparison to the system with traditional centric architecture. Finally, chapter 8 summarize this thesis by focusing the contribution of this thesis and future work of it.

Chapter 2

Sensor Network

2.1 Introduction

This chapter discusses and presents ideal sensor network architecture. Firstly, we introduce a basic concept of general sensor network and utilization example. Then, we will organize architecture of sensor network applications and introduce the problem of its implementation. Then at the end of this chapter, we will introduce requirements and ideal sensor network architecture.

2.2 Sensor Network

Due to recent development of sensing and wireless communication technologies, small sensing devices have had ability to communicate each other. This technology enables sensor devices to formulate a network autonomously, called "Sensor Network" [1]. Sensor network helps passing data or bringing commands among nodes, even the two can not directly communicate each other. Since the nodes used in sensor networks are typically low cost and small in size, they are distributed to an environment to enable dense resolution sensing. Due to these features, sensor network is usually utilized for detecting events or phenomena in an environment.

Imad Mahgoub and Mohammad Ilyas [2] have described the basic fea-

tures of sensor network as following:

- Self-organized capabilities
- Short-range broadcast communication and multihop routing
- Dense deployment and cooperate effort of sensor nodes
- Frequently changing topology due to fading and node failures
- Limitation in energy, transmit power, memory and computing power

The first paper that showed a concrete concept of sensor network was smart dust [3]. This research aims to acquire detailed real-space information of the field by distributing an enormous number of dust-sized sensors as shown in Figure 2.1. These tiny devices sense various types of information including temperature, light, humidity, radiation, CO2, and more. This helps people to get better understandings of environmental or geographical information. Figure 2.2 shows the concept of smart dust.

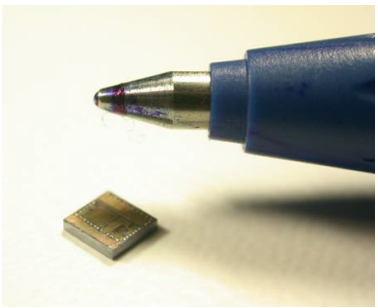

Figure 2.1: node for Smart Dust

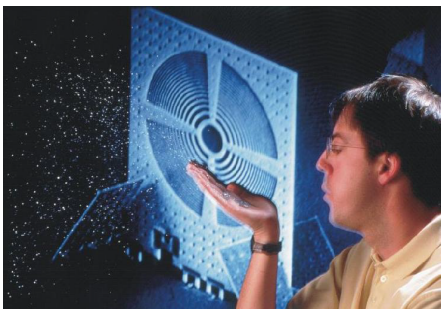

Figure 2.2: concept of smart dust

Issues described above were rather the research of enabling technologies of sensor network itself. In addition to these issues, researches of overlying applications of sensor network are also active these days.

Hardware related issue underlies at the bottom. This research issue includes hardware sophistication/miniaturization, power supply, and energy efficiency. Networking related issue lies next. This layer of research issues

includes routing, network topology, and in-network database system. This is the core layer of sensor network. Application layer includes context awareness, data-mining, computer-human interaction.

2.3 Applications on Sensor Network

This section describes sensor network applications and how they utilize sensor network features.

2.3.1 Environmental Monitoring

Environmental monitoring is one of largest classes of sensor network application. By distributing small and low cost sensor nodes all over the environment, the network of these small sensor nodes acquires a high-resolution sensing result that is impossible to acquire with an existing environmental sensing method. A traditional method for this kind of monitoring is to place a single high-specification sensor station that represents an area around it. This method can only acquire low-resolution data that represents a large area even though the data itself is so precisely sensed. The resolution of the data that is sensed by this method in a park is approximately $500m^2$ to $1km^2$. We can easily guess that there are some place colder/hotter than the other areas such as windy shade, or concrete-made road in the sun in such a large area.

Airy Notes [4], which is a name of first practical challenge to place hundreds of sensors in a park, supports this fact. In Airy Notes, Ito et al. placed more than 200 of 1mm x 10mm x 0.5mm wireless sensors shown in Figure 2.3 in a park, and overlaid the sensed environmental data on the park's map as shown in Figure 2.4. The result shows even in $5m^2$, temperature differs widely. The high-resolution data will be utilized for park's environment management or to evaluate effects of greens to the temperature. Sensor network can be also utilized to various use such as fire monitoring [5] or human activity monitoring [6, 7, 8]. Human activity monitoring is an innovative

technology for the elderly care system [9] with privacy.

Figure 2.3: airy note sensor

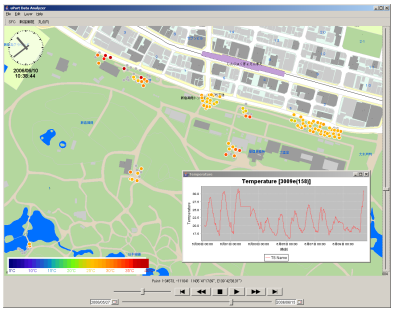

Figure 2.4: airynotes system

2.3.2 Context Aware Service

Sensor network contributes to the emergence of this class of applications, which is "Context Aware Service" [10, 11]. Due to sensor network, computer system can take various types of information into cyber space from real world. This gives computer systems to understand what is going on in the real world. Context aware system can actuate the most reasonable service under a context that is analyzed from real-world information. The difference between automatic-door or automatic-light which are the system that have been automated for long time, is that context-aware system calculate more abstract situation using various sensor data, on the other hand, traditional automated system only uses a single data.

For example, current situation of the room, "serious" or "relaxed", can be analyzed by brightness of the room, chairs' accelerometer, projector's on/off, and tone of the voice. And the situation can be utilized cell-phone mode switching service. The service will switch to vibration mode when you are in "serious" room and switch to ring-mode when you are in "relaxed" room. The context analysis methodologies are widely proposed. Some are based on probability calculation, other are based on exploratory algorithm.

2.3.3 Life Log Service

The concept of life logging is to self-record daily behaviors, such as visited places, purchased items, and encountered friends. This information is logged by various medias such as still camera, video camera, or text. Sensor network can enrich the media by providing environmental information as attendant circumstances, so that the users can recall the situation more vividly. ObjSampler [12] is a great example. it is stamp shaped life-logging device with RFID reader that can read and store objects' digital information. When a user stamps on an product which interests him/her, @Reader stores its digital information. @Reader can also receive real-world information such as location or behavior and add it to the product's information. This data can help the user to recall the detailed situation when he/she found the product.

2.4 Architecture of Sensor Network Application

We have organized architecture for sensor network applications. Figure 2.5 shows the architecture. On the bottom lies sensor network layer. This layer brings real-world information to an upper layer application. Data processing layer comes next. This layer processes raw sensor data and find out abstract contexts. Then, application layer comes on the top. This layer provides services to users by utilizing abstract context information from processing layer. This architecture model can be adapted to most of the sensor network applications. However the problem is each sensor network application has its original instance in each layer. Application A has its own sensor network, process method, and application. Application B has another sensor network, another process method and another application. This means sensor network and its application is hardly binded. What makes situation worse, each sensor network is closed to the others. This situation is undesirable for following reasons.

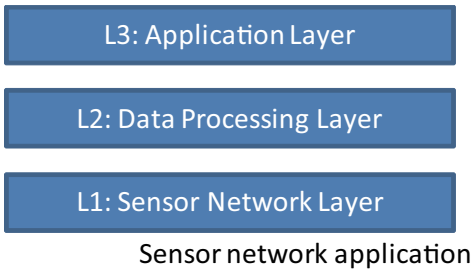

Figure 2.5: architecture for senor network application

- **Less Versatility**

  Currently, if one wants to create sensor network application, he/she has to build from the bottom layer to the top layer. This causes large development time and cost.

- **Less Mobility**

  Existing implementation of sensor network application can only operate on a specific sensor network, which means applications with mobility like Life Logging or Robot Support are available only in a limited area. Mobile application's mobility is restricted in current situation.

- **Less Visibility**

  Applications can only subscribe to sensor data from a sensor network which is specifically installed to it. It is unlikely to think that one application developer or organization can deploy sensor network to whole city therefore this limits the visibility of sensor network application.

As written above, this architecture is not enough for services that envision subscribing data from larger area that single sensor network can not cover. It is necessary to unify and open up the sensor network layer and some extent of data processing layer as a common platform.

2.5 SensingCloud

In this section, we will present an ideal common sensor network platform, which we named "SensingCloud." Based on discussions above, an ideal sensor network is "a sensor network that applications can request for arbitrary data in desirable form anywhere they are." As shown in Figure 2.6a, an application can subscribe to sensor data around it. Also, an application can subscribe to data from larger area,even if the networks are deployed in remote place as shown in Figure 2.6b.

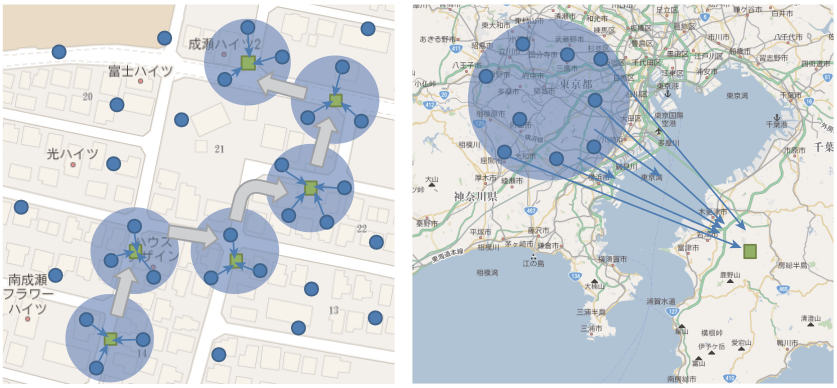

Figure 2.6: over view of SensingCloud

The feature of SensingCloud enhances the ability of the sensor network applications. The environmental monitoring applications can enlarge their target area from a park or community to city or country. Also, this platform enables an application to subscribe to feeds from sensors which are not deployed for its purpose. For example, an earthquake monitoring application can leverage acceleration sensors in computers whose initial purpose is for HDD protection. Detailed earthquake intensity map can be created by monitoring acceleration data from computers located in a city.

Context aware services can subscribe to sensor data anywhere they are. This enables mobile device can execute context aware services utilizing not only their embedded device but also variety of environmental information.

Context aware services on mobile devices needs to utilize sensors embedded to the device themselves, since they can not subscribe sensor feeds in strange place. The same thing can be said to Life log applications.

2.5.1 Requirements for SensingCloud

Based on the overview of the ideal sensor network platform, we have organized 3 major requirements.

Transparent Connectivity to distributed sensor networks

Applications should not be aware of each individual sensor network distributed in the world. The application should only request for its necessary data to SensingCloud, even if the necessary data comes from multiple sensor networks which belongs to different domains. SensingCloud requires abstracting these cross domain sensor networks and provide applications transparent access.

Meta-Level Accessibility to arbitrary data

Applications should access to data from arbitrary nodes in the SensingCloud. However, applications usually do not know the existence of circumjacent sensors. Applications can not specify sensor or sensor network to access with identifier such as sensor's unique id, IP address or MAC address. SensorCloud should search for the target sensor or sensor network from queries from application which contains meta-level information such as Location and sensor type.

On the other hand, reply of the SensingCloud should be Meta-Level too. Applications want to know the semantics of the sensor data, not only sensor data itself. SensingCloud should answer not only concrete data but also data with meta-level.

Automatic Scalability for growing number of sensors

The number of networked sensors is dramatically increasing and it is assumed to be for the future. SensingCloud needs to be scalable to manage the increasing number of sensors and their data streams all over the world. In addition, from a perspective of management cost, SensingCloud should scale automatically as the number of sensors grows. It is hard for one authoritative organization to scale up the system as SensingCloud gets bigger and bigger.

2.5.2 Utilization Scenario

Following applications are utilization scenario of SensingCloud.

- **Precise Heat Island Effect Monitoring**
  Using SensingCloud, monitoring applications can acquire temperature data from sensors distributed in the city. By mapping the data to the city map, the effect of heat island and anti-action can be visualized.

- **Precise Rain Map**
  By subscribing humidity information from SensingCloud, real-time and high resolution rain map can be created. By extrapolating the future area of the rain, the application can provide information that the remaining time until rain starts.

- **Real-Space Computer Game**
  Computer games that take into account real-world information can be developed. Currently world in games are totally separated from real world. SensingCloud enables to connect those separate two worlds. For example, there can be an item that players can only get in places under -5C.

2.6 Summary

In this chapter, we have described the basic concept of sensor network and introduced traditional sensor network and applications running on it. Currently when an application developer implements a sensor network application, he/she has to deploy a new sensor network only for the application. What makes the situation worse, each sensor network is closed to the others. This situation indicates that sensor network has not yet became a common infrastructure. This restricts service area of the applications and diminishes development efficiency. To defeat this situation, we figured out an ideal sensor network platform what we named SensingCloud. SensingCloud should 1) abstract cross domain sensor networks, 2) provide worldwide accessibility to sensor data from sensor networks which belong to different domains, 3) provide basic controllability for incoming data flow that can be enormous and 4) respond quickly to data request. In the next paragraph, we will introduce our approach to create SensingCloud.

Chapter 3

Related Works

3.1 Introduction

In this chapter, we introduce existing novel researches that share the goal with us. The researches aims to create an architecture that connects distributed sensor networks in the world and make them accessible to other applications and users. In reading the papers of the work, we have organized their approaches into 3 categories: "High Performance Computer (HPC) Based Architecture," "Cloud Based Architecture," and "Grid Based Architecture." In this chapter, we will acquire hints to design our system by examining that approaches of existing researches can solve our initial requirements of our system.

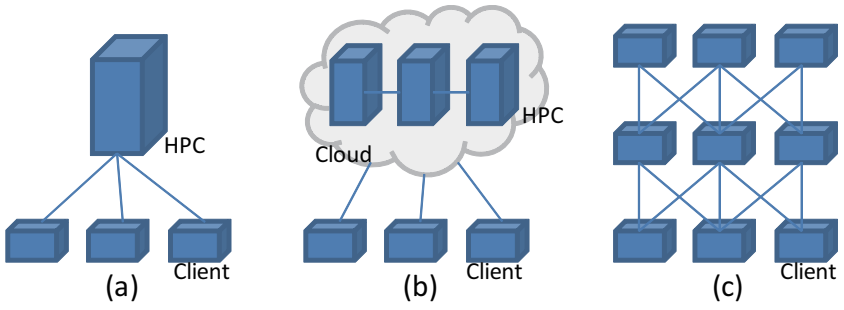

Figure 3.1: a) HPC Based, b) Cloud Based, c) Grid Based Architecture

3.2 HPC Based Architecture

High Performance Computer (HPC) Based Architecture is traditional way to integrate a large-scale system. Figure 3.1a shows HPC based architecture. Placing a HPC in the middle and concentrate data and computation into the machine, the other components of the system are not required to have a large computational power.

3.2.1 SensorWeb

Liqian et al. has proposed SensorWeb [13, 14], which is an open and scalable infrastructure for sharing and geocentric exploration of sensor data. Their initial motivation is to create a SensorMap that enables spatio-temporal sensor data exploration. The software image of SensorMap is shown in Figure 3.2. When a user specifies the area of interest by drawing a polygon or input the name of the city, SensorMap automatically aggregates sensor feeds within the area at an appropriate granularity based on the zoom level as shown in Figure 3.3. The SensorWeb project is to create a back-end system to enable large scale sensor sharing and exploration for SensorMap. In the research, they have addressed two research challenges, namely heterogeneity and Scalability.

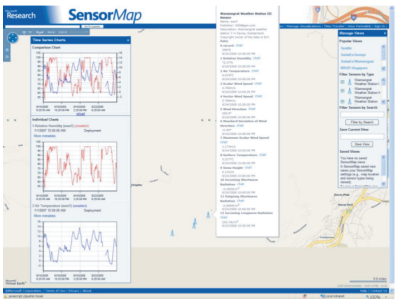

Figure 3.2: Screendump of SensorMap

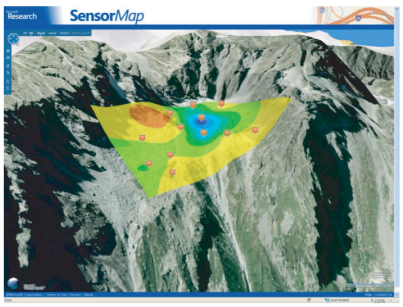

Figure 3.3: 3D data distribution

As for heterogeneity issue, unlike monoclinic sensor network, a large-scale data stream sharing system has a variety of sensors, data types, and

application needs. They have named 3 heterogeneities existed among the system, which are sensor heterogeneity, data heterogeneity and application heterogeneity. They need to be abstracted user from applications.

Scalability issue is important for SensorMap interface. Since SensorMap is highly interactive application, the SensorWeb needs to respond to the request in real-time although the application needs a large computational cost. Special visualizations based on contour map is the good example. It requires interpolation of unobserved sensors data and it needs large computational cost. In addition, they can not be pre-calculated.

They have proposed system architecture for heterogeneity issue and scalable algorithm to attack the scalability challenge. Here, we focus on the system architecture and introduce the approach. The architecture is shown in Figure 3.4. Their components are "Sensor Gateway", "Coordinator" and "Transformer". Sensor Gateway interfaces sensors to SensorWeb. It forwards sensor data stream to Coordinator, which is like DNS for SensorWeb. Coordinator transmits the stream into Transformer, which converts data semantics, for example unit translation, data fusion. At last, transformed values are provided to user applications from transformers.

As for SenseWeb, the research project has the same goal with ours, but research challenges are slightly different. Especially for scalability issue, while we addressed that automatic scalability is required for this architecture, they do not mention "automatic". Their architecture is scalable, however, someone in the middle need to reinforce the hardware to scale up the system. Ensuring scalability with algorithmic approach is appropriate for SenseWeb since it has a concrete utilization example such as SenseMap. On the other hand, in SensingCloud, the algorithm should be flexibly replacable therefore, the approach is not appropriate for our assumption. It is necessary for us to consider the approach to ensure the scalability with architecture. The same thing about automatic scalability as SenseWeb can be said to Sensor Andrew. Although the architecture supports extensibility, it does not scale automatically when the system grows. Someone needs to

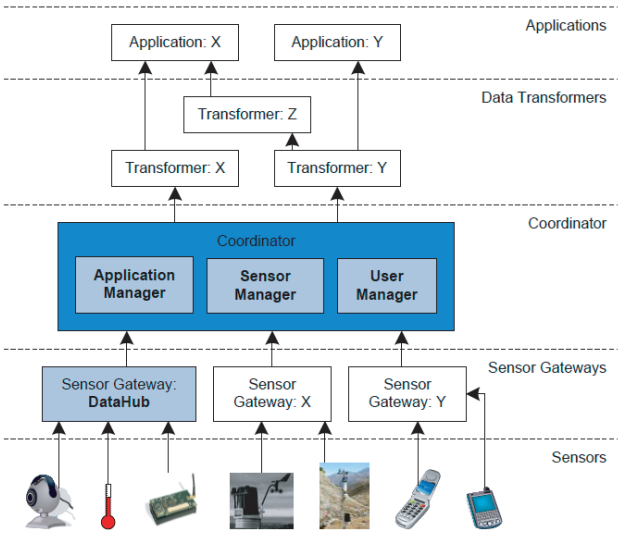

Figure 3.4: Architecture of the SenseWeb system

upgrade the server layer. Therefore, even though it is extensible, the architecture rely on high performance computer is not appropriate to achive our ideal scalabile architecture.

3.2.2 Sensor Andrew

Sensor Andrew [15, 16] is a framework which enables large-scale campus wide sensing and actuation. The goal of this project is to develop a framework to support large-scale monitoring, operation and control of infrastructure with extensible, easy to use, and secure manners. Different from the other related work, this project supports controlling actuators, not only publish/subscribe to sensor feeds. The architecture of Sensor Andrew is shown in Figure 3.5. There are 3 layers in the architecture, namely “Transducer Layer”, “Gateway Layer”, and “Server Layer”. Transducer layer includes sensors and actuators with little or no computational resource. Gateway layer includes medium to desktop class computers that have the Internet connectivity. They bridges data from transducers via variety of bass or communication protocols to campus network. Then server layer contains high-performance systems with

extensive storage capabilities. They are responsible for administration of entire system such as control, data aggregation and storage. This layer also have a component named "Agents", which can analyze sensor data and provide high-level service or meta-sensors which provides abstract events.

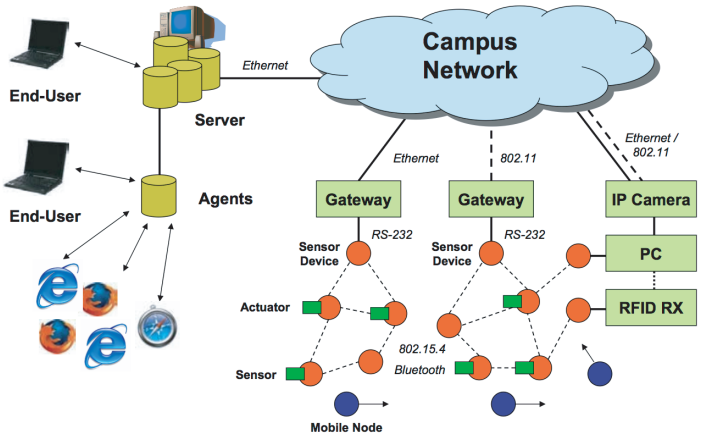

Figure 3.5: The Sensor Andrew architecture

The feature of this project is that Sensor Andrew leverages eXtensible Messaging and Presence Protocol (XMPP) [17] as a communication protocol. According to the paper, they adopted XMPP since not only that it is already the internet standard, but also it provides 1) security feature, 2) both point-to-point and broadcast capabilities, 3) public-subscribe to functionality, 4) organized event messages with an internal database for storing transaction records and 5) clustering or replication to meet scale demands as well as provide backup fault tolerance. Each functionality meets the Sensor Andrew's communication requirement, which are 1) standard messaging format, 2) extensible message types, 3) point-to-point and multicast messaging, 4) support for data tracking and/or event logging and 5) security, privacy, access control.

3.3 Cloud Based Architecture

Although Cloud Based Architecture as shown in Figure 3.1b has a high performance component in the middle like HPC based architecture, the component consists of many computers. The component is called "cloud". The scalability is high since the cloud is usually easily extensible. For the same reason as HPC approach, cloud based architecture seems to be appropriate for a sensor network integration.

3.3.1 Sensor-Cloud Integration

Hassan et al. has proposed an framework of sensor - cloud integration [18]. In this framework, distributed wireless sensor networks forwards data to the cloud and cloud do the rest. Tasks such as data aggregation, data monitoring, and even application is done in this cloud. As shown in Figure, they leverages the concept of Software as a Service (SaaS), they includes applications layer tasks in the cloud. The users only get the result of the process of SaaS. The Figure 3.6 shows a framework for a cloud provider.

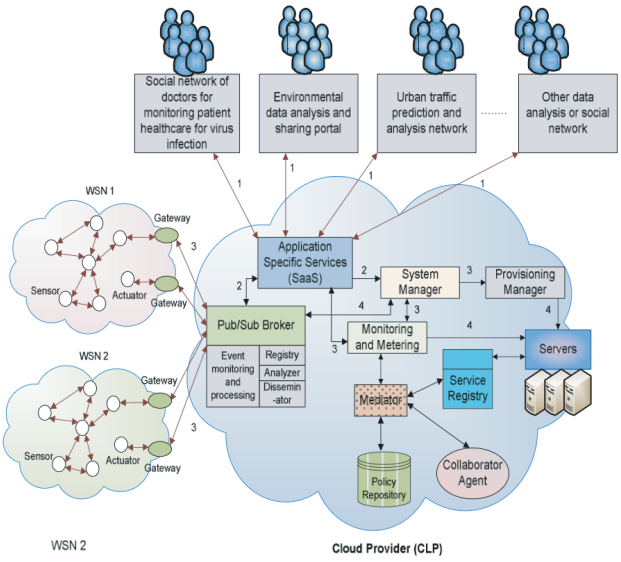

Figure 3.6: A framework of Sensor-Cloud Integration

However, they point out the scalability problem when huge sensor data processing tasks are concentrated on the cloud. For the problem, they have proposed the mechanism for collaboration between different cloud providers named "VO (Virtual Organization) Based Dynamic Collaboration". By this cloud-cloud collaboration realizes the dynamic scalability. However, they have addressed serious problem which is the cost and trust. This collaboration is among companies and business related issue is a large challenge.

However we both use a term "cloud", the two researches have a major difference, which is the architectural difference from the definition of the term cloud. Our definition of cloud includes distributed local sensor networks. On the other hand their definition of cloud does not include distributed local sensor network. Also, we envision the cloud that grows spontaneously by end-users' participations without business related issues.

3.4 Grid Based Architecture

Grid Based Architecture Figure 3.1c is based on a concept that gathering relatively small computational resources and generate a large computation power [19]. This approach is opposite approach from the other two but also suitable for the sensor network.

3.4.1 Global Sensor Network

Aberer et al. has proposed Global Sensor Network [20] which they describe it as Sensor Internet. The work trying to interconnects heterogeneous cheap and smart wireless sensor devices such as MICA Motes [21] or BTNodes [22]. They advocates the importance of interconnecting these sensors. In their paper, they refer the success of the publication of documents on the World Wide Web. By referring WWW's logical abstractions (URL, hyperlinks, and HTML) and basic communication protocols (HTTP), they have proposed 1) simple and strong xml abstraction for sensor devices, 2) adaptive container based system implementation which allows dynamic reconfiguration without

stopping the system, 3) scalable peer-to-peer architecture to handle a very large number of data and 4) light-weight implementation for easy deployment in standard computing environment. In the paper, they mainly focus on simple and strong sensor abstraction that they named "Virtual Sensor". In our research, referring to their Virtual Sensor abstraction, we will also focus on whole scalable architecture, including sensor discovery.

3.4.2 TomuDB

There are another instance from Grid Based Architecture. TomuDB [23] is a project which aims to provide a platform that people can query sensor data from wide area such as city or country with multiple-resolution. When the system can not match the resolution that user has demanded, it interpolate [24] the data to match the resolution. TomuDB's is very similar to SensorWeb but it has different architecture. The authors of TomuDB points out that sensor feeds from all the distributed sensor network are sent to a centric storage of SensorWeb and the approach tends to occur bottoleneck. TomuDB proposed to use distributed approach, on the other hand SensorWeb adopted centric approach.

The TomuDB's system architecture is shown in Figure 3.7. In the very front, "Web API Server" converts human readable query to XML format and send the query to "DBSN Requester". DBSN Requester converts XML query into SQL query and forwards to an Overlay Network consists of "DBSN Peer". DBSN Peer is responsible for 1) collecting data from sensor nodes and 2) answering SQL queries.

This system fulfills our system requirements in a architectural point of view. However, we consider that requiring database system to the peers which are managed by sensor network owners are to heavy assumption. In addition, we think that data storage is the responsibility of software in the application layer. The sensor network should support bridging data stream.

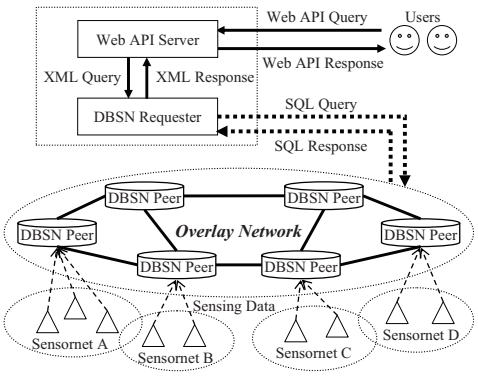

Figure 3.7: The TomuDB architecture

3.4.3 IrisNet

Phillip et al. has proposed IrisNet [25]. The project envisions a worldwide sensor web, in which user can query sensors distributed in the world. In their research, they proposed architecture to enable a planet-wide data collection with easy, secure, robust, and real-time manner. The architecture is shown in Figure 3.8.

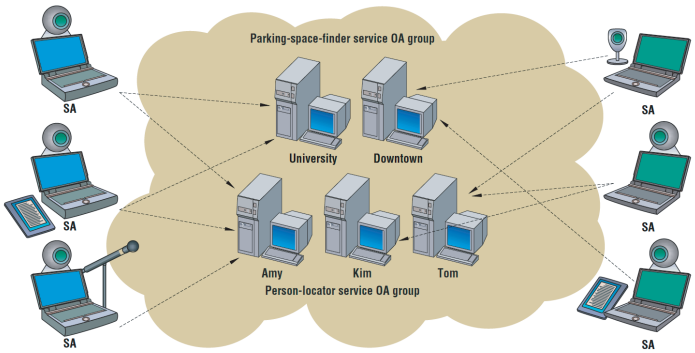

Figure 3.8: The IrisNet architecture

The system composed of 2 components, which are “Sensing Agent” (SA) and “Organize Agent” (OA). SA is a nodes with genetic data acquisition interface. SA processes and forwards data from the sensors connected to the

node to OA. The processes of SA is described in a code named "senselet". On the other hand, OA is the node with a part of distributed database system which stores service-specific data from SAs. OAs are hieratically organized as shown in Figure. This hierarchy enables efficient processing of geographically scoped queries.

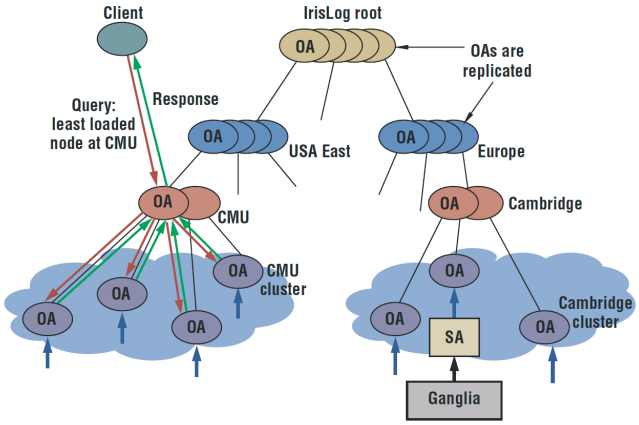

Figure 3.9: The IrisNet OA Hierachy

3.4.4 Sensor Grid

Sensor Grid is a project that aims to share sensor data just the same as the other research projects. The project leverages the technique of grid computing to enable efficiently process, analyze and store the vast amount of sensor data using the computational resource and data storage in the grid. Also, the grid computing technique enables different users and applications to share the sensors in flexible usage scenarios. The grid provides seamless access to the sensors distributed world-wide anywhere and anytime. As the computational resources in the sensors getting larger, this sensor grid scenario gets more realistic.

Their proposing method is to share the sensor data with existing grid computing technologies such as OGSA [26]. OGSA is a specification that enables grid service in web services using technologies such as XML, WSDL

and SOAP. This framework supports standard interfaces for dynamic service creation, destruction and finding. Using OGSA, the vision of sensor grid described above is realized without developping whole new system architecture.

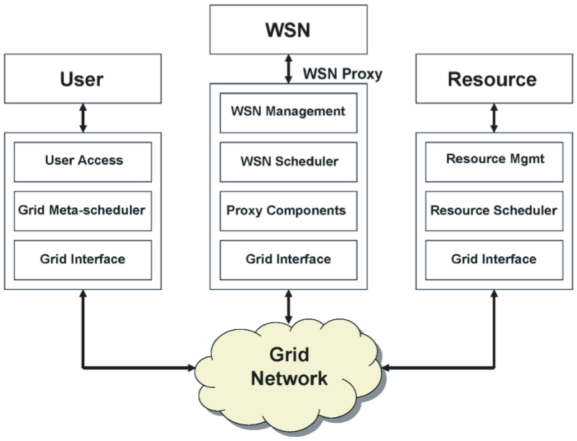

Figure 3.10: The SPRING framework

In order to build such sensor grid, they have proposed Scalable Proxy-based Architecture for Sensor Grids (SPRING). Existing grid services are too complex to be implemented in Sensor nodes that are commonly utilized among sensor network developers. Therefore, they put "Sensor Proxy", which acts as an interface between wireless sensor network and the grid. The proxy not only exposes the sensor resources as a grid services, it bridges the data from local wireless sensor network protocol to the Internet Protocol and provides an ability to join/leave autonomously from the grid, and other necessary services such as power management, scheduling, security, and QoS management. The framework architecture is shown in Figure 3.10. The sensor resources are accessible in a similar manner as other compute or data resources.

3.5 Summary

In this section, we have introduced the project and researches which shares the project goal with us. We organized these researches into 3 categories by its architecture, which are "High-Performance Computer Based Architecture", "Could Based Architecture" and "Grid Based Architecture". For each category and its instance, we discussed whether they fulfill our system requirements. As a result, we have found that SensingCloud should be accessed as easy as we do to the cloud, and inside of the cloud should be grid architecture whose components are belongs to different organizations or individuals. In the next chapter, we will introduce the detailed system architecture of SensingCloud.

Chapter 4

SensingCloud

4.1 Introduction

In this chapter, we will propose SensingCloud. We will describe 3 design principles of SensingCloud that fulfills the requirements of ideal sensor network as we discussed in Chapter 2. Then, after introducing hardware and software architecture, we will introduce and explain the designs of SensingCloud components which are "Distributed Process Client", "Index Server" and "User Application".

4.2 System Overview

SensingCloud enables applications to receive data from arbitrary membership nodes. Major component of SensingCloud are "Distributed Process Client" and "Index Server". Distributed Process Client is a GUI based sensor network management tool. Although the front end of this client is GUI based network manager, it bridges the data stream from local sensor network to IP network. Moerover, they dynamically turns to be a server which integrates data stream from other clients and provide sensor feeds to the applications. Distributed Process Clients is a part of distributed computational resource in SensingCloud. Index Server resolves IP addresses of relevant Distributed Process Clients from meta-level request from applica-

tions and sends control commands to them. Sensor networks distributed in the world function as a large scalable sensor network when administrators of these sensor network utilize Distributed Process Client.

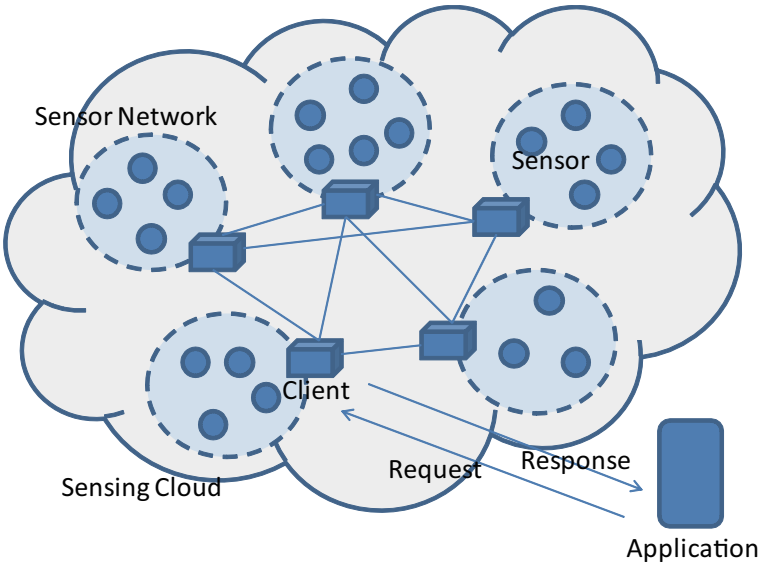

Figure 4.1: System Overview of SensingCloud

4.3 Design Principle

There are 3 design principles: "Transpaerent Cross Domain Communication," "IP Bridged Communication" and "Abstraction of Sensor Data."

4.3.1 Transpaerent Cross Domain Communication

There are two heterogeneity problems in sensor network which are network heterogeneity and device heterogeneity. For the system to behave as if it is single sensor network, we enables transparent cross domain communication with 2 approaches, which are IP Bridged Communication and Abstraction of Sensor Data.

IP-Bridged Communication

There are various network protocols for sensor network such as Zigbee or Bluetooth. It is impossible for sensor networks with different protocols to communicate each other. To link up these sensor networks that adopts different communication protocol, we have IP network stand in the middle of heterogeneous sensor network. By connecting a computer with a sink node for its sensor network, all data coming from the sensor network is IP reachable. Usually sink nodes for sensor networks are connected to a computer for applications to use the data collected from sensor network. We have chosen IP as a bridge assuming that the computers are connected to the Internet.

Abstraction of Sensor Data

The other heterogeneity is various sensor feeds. SensingCloud includes various feed types such as temperature, humidity, acceleration and so on. In addition, each has several types in unit. For instance, temperature information can be described not only in Celsius but also in Fahrenheit. Sensor feed should be abstracted in order to be able to locate target information or to aggregate target data described in different units.

XML 4.3.1 shows sensor feed abstraction. Sensor and Feeds are two major elements in this XML. “sensor” element shows the sensor device information such as id and location. “feeds” element contains more than one “feed” elements since one sensor device can have multiple sensing elements on its board. “feed” element contains “type” and “value” element. “type” specifies a data type such as, temperature, humidity and acceleration. “value” represents the sensing data in a measurement that specifies in its property.

4.3.1Sensor Feed XML

```
<sensorFeed>
    <sensor>
        <id>abc12345</id>
        <location>
            <latitude>35.38827</latitude>
            <longitude>139.427326</longitude>
        </location>
    <sensor>
    <feeds>
        <feed>
            <type>temperature</type>
            <value unit="celsius">18</value>
        </feed>
        <feed>
            <type>humidity</type>
            <value unit="percent">30</value>
        </feed>
    <feeds>
</sensorFeed>
```

4.3.2 Meta-Level Request Processing

Location Based Sensor Discovery

Applications can request SensingCloud for sensor feeds by specifying target location and data type. This enables applications to acquire target feeds not knowing the identifier of the target sensors. In the index server, sensors are described with id, data type, sensor network it belongs to and location information. Therefore, the index server can resolve the sensors and sensor network to collect feeds. Sensor network administrators put those necessary information when they register the sensor through sensor network management tool in Distributed Process Client. Sensors with mobility such as cars and cell-phones need to update location information up to date periodically assuming that they have GPS and IP connectivity.

Characteristic Value Processing

"Characteristic Value Processing" translates raw data into meta-level information. Providing only raw data without any processing has a problem that applications can not expect how much data will be transformed. Applications will not be atble to extimate processing time, especially devices with less calculation resource.

Figure 4.2 shows a detailed architecture of data processing layer. Defining that the lowest-level expression of sensor data is raw data, the most meta-level expression is what we call context such as situation, behavior or mood. Expression stands between the two is characteristic value of sensor data, such as max/min, average, variance of a certain area. The discussion here is that what extent of the processing layer should be included in SensingCloud.

The bottom layer is named raw data layer. This layer provides the data without being processed. This layer transform unit of sensor data, such as meter to centi meter. The layer in the middle is named characteristic data layer. This layer executes basic numerical operation such as average and variance using raw data. The output data of this layer is still in numerical form. Then, context layer transform characteristic values to more abstract information.

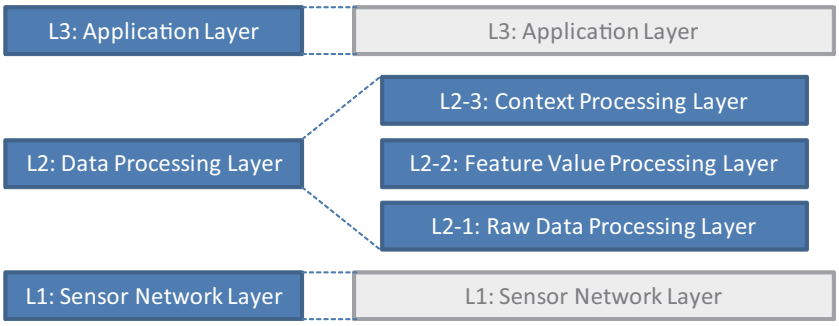

Figure 4.2: architecture of sensor data processing

The lightest design of sensor network platform only includes raw data

layer. Applications are responsible for the characteristic calculation and context inference with this sensor network. Applications receives raw data from sensor network, therefore the query would be “Temperature in Tokyo in C” or “Humidity in Amazon in %”. This design has a problem that applications can not expect how much data will be transformed. Applications can not infer processing time, especially devices with less calculation resource. As written above, this design is not practical.

On the other hand, a sensor network that includes raw data layer and characteristic value layer seems to be reasonable. The applications can restrict the number of incoming sensor data since the network is responsible for those numerical procedure. In this way, sensor network application can expect the execution cost before receiving data from the network. Query will be “Average noise level in Los Angels in Db” or “Highest Temperature in Galapagos in C with 1km2 resolution”. Also, this design is reasonable from the aspect of implementation matter, since the number of basic numerical operation is limited.

The richest design of sensor platform includes all three layers. Although this design includes everything, it is impractical. Query of applications to such sensor network would be “A situation of this room” or “Today’s weather of New York City”. In this way, application developers seem to be able to create sensor network applications very easily and application itself becomes compact. But answer to such query will be too abstract that there can be possibly an infinite numbers of answers and impossible for applications to handle. Therefore this design is impractical.

As a result, sensor network platform should be responsible for tasks in first and second layer that are transformation data units and execution of basic numerical operation as shwon in Figure 4.3. Based on these discussions, next section proposes ideal sensor network platform.

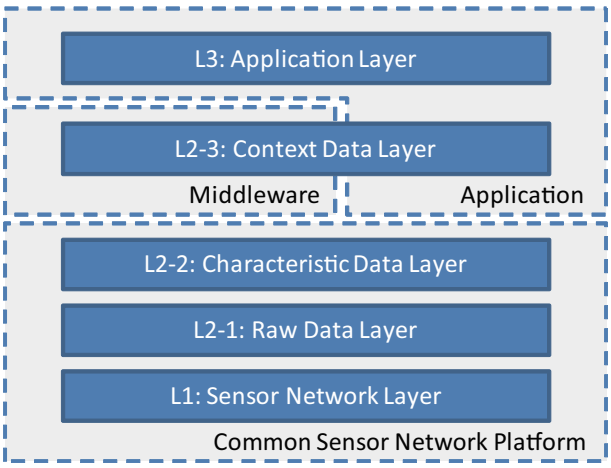

Figure 4.3: architecture image for common sensor network platform

Query Design

The XML format of data request from Application to SensingCloud is described in XML 4.3.2. The element named "geography" is to define interested geographical area. "feedCondition" element represents filtering condition for feed forwarding in FeedProcess. "feedAggregation" element defines how to aggregate the feeds filtered with the condition described in condition element. "name" parameters for both feedCondition and feedAggregation are corresponding each other. In this XML example, feedAggregation named "temp" is applied to feeds which matches to feedCondition named "temp". Not defining a feedAggregation element for a feedCondition means application wants data without any operation.

4.3.2Feed Request XML

```
<request>
    <client>
        <ip>133.27.170.224</ip>
        <port>23456</port>
    </client>
    <geography>
        <area type="radius">
            <value type="km">5<value>
        </area>
        <longitude>139</longitude>
        <latitude>36</latitude>
    </geography>
    <feedConditions>
        <feedCondition name="temp">
            <type unit="celcius">temperature</type>
            <condition type=">=">30</condition>
        </feedCondition>
        <feedCondition name="noise">
            <type unit="decibel">sound</type>
            <condition type="*"></condition>
        </feedCondition>
    </feedConditions>
    <feedAggregations>
        <feedAggregation name="temp">
            <interval>1500</interval>
            <operation>ave</operation>
        </feedAggregation>
    </feedAggregations>
</request>
```

Table 4.1 is a list of conditions and operations which SensingCloud supports currently.

4.3.3 Distributed Data Aggregation

SensingCloud has to be designed with scalability since it manages a great number of sensor networks in the world. SensingCloud leverages distributed

Table 4.1: Operation Set for Data Aggregation

Type	Symbols	Description
Condition	<, <=, ==, >=, >, *	Filter unmatched values
Operation	sum, ave, var, max, min, idw	Calculate values in a period

computing to make the system scalable. We will present the merit and demerit of both centralized and distributed computing and introduce the reasons why we adopted distributed manner. Then, we will propose the scaling mechanisms for SensingCloud.

Centralized and Distributed

There are two possible models for SensingCloud. One is "Centralized Computing Model" and the other one is "Distributed Computing Model". In centralized computing model, all the tasks are executed in a single strong calculation resource which can be a single server or a cluster of servers, as shown in Figure 4.4. On the other hand, tasks are divided into pieces and each is executed on relatively small calculation resources in distributed computing model as shown in Figure 4.5. Table is a comparative chart for characteristics of both centralized and distributed computing model.

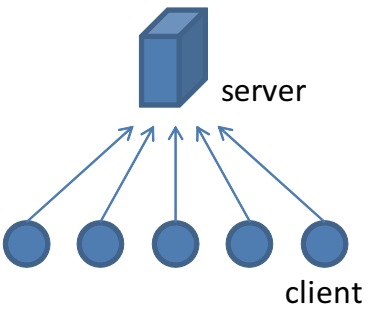

Figure 4.4: Centralized Model

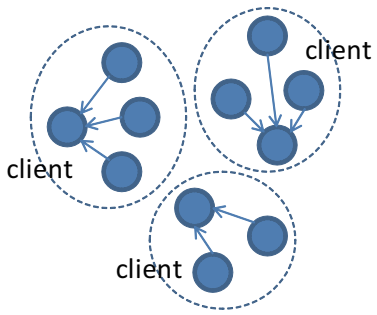

Figure 4.5: Distributed Model

The reason for SensingCloud to leverage Distributed P2P model is that

the model can distribute the computation and network transaction. If designing this system with centralized server client model, all the data must transmit to a remote central server and processed on the server. On the other hand, the system designed with distributed P2P model can complete the process with utilizing adjacent computational resource. The system does not have to transmit the data to the remote server and the computation do not concentrate on a single spot. In addition, the system scales automatically in propotion to the size of the system, by participatory sensor network being a part of computational resource. If the system is designed with centralized server client model, the administrator has to add computational resource to the central server as the system gets bigger. Also, distributed P2P model distributes data without concentrating information to a specific organization. This feature avoids some organization to take advantage or to control information. Table 4.2 shows summary of the comparison between the two design methods. For SensingCloud to be scalable, we have adopted distributed P2P computing model.

Table 4.2: comparison of centralized and distributed computing

	Centralized Server Client	Distributed P2P
Transaction	all to a remote server	to adjacent resource
Computation	all on a single server	on participatory resources
Scalability	scales manually	scales automatically

Pure P2P and Hybrid P2P

There are two types of P2P designs: “Pure P2P” and “Hybrid P2P”. Pure P2P model shown in Figure 4.6 is completely server less architecture. In this architecture, peers exchange their information and enables node searching autonomously. However, this design enhances the systems scalability and robustness, network load increases. Moreover, this design does not guarantee finding target node and respond time varies too. On the other hand, Hybrid

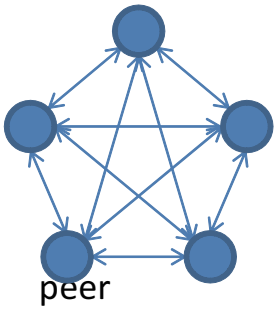

Figure 4.6: Pure P2P Model

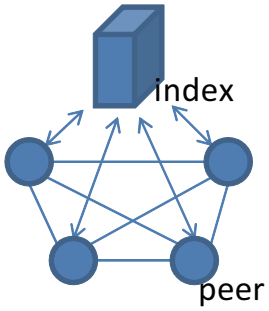

Figure 4.7: Hybrid P2P Model

P2P model shown in Figure 4.7 has an index server which contains information of peers. In this model, firstly a client asks the locations of peers which have target contents, and then start to communicate peer to peer. Although this architecture decreases the system scalability and robustness, they can find target node without failure with stable respond time. Table 4.3 shows the summary for the design comparison. For the reliability of node discovery and respond time, we have adopted Hybrid P2P design.

Table 4.3: comparison of Pure P2P and Hybrid P2P

	Pure P2P	Hybrid P2P
Robustness	○	×
Scalability	○	×
Responsiveness	×	○
Reliability (Discovery)	×	○

SensingCloud Scaling Mechanism

Based on Distributed P2P Model with hybrid P2P architecture, we have designed SensingCloud's auto scaling mechanism. Figure 4.8 shows the procedure of SensingCloud Scaling Mechanism.

When the user application request for sensor feeds, 1) it accesses to index server to resolve which sensor network provides the target feeds. 2)

The index server resolves the IP addresses of the clients connected to the target sensor networks in a specified area, and nominates a master node from the nodes geologically close to target clients. 3) The index server sends "Aggregation Request", which includes contents of query, IP address of the source sensor networks and IP address of destination, which is the IP address of the application that has requested the feeds. 4) Then, the master node sends "Feed Request" to the clients connected to target sensor networks. Feed request contains IP address of master client, types and conditions of the feeds to be forwarded to master client. After this procedure, 5) the clients forward the feeds from their sensor networks to master client. 6) The master client aggregate the feeds and send to the user application.

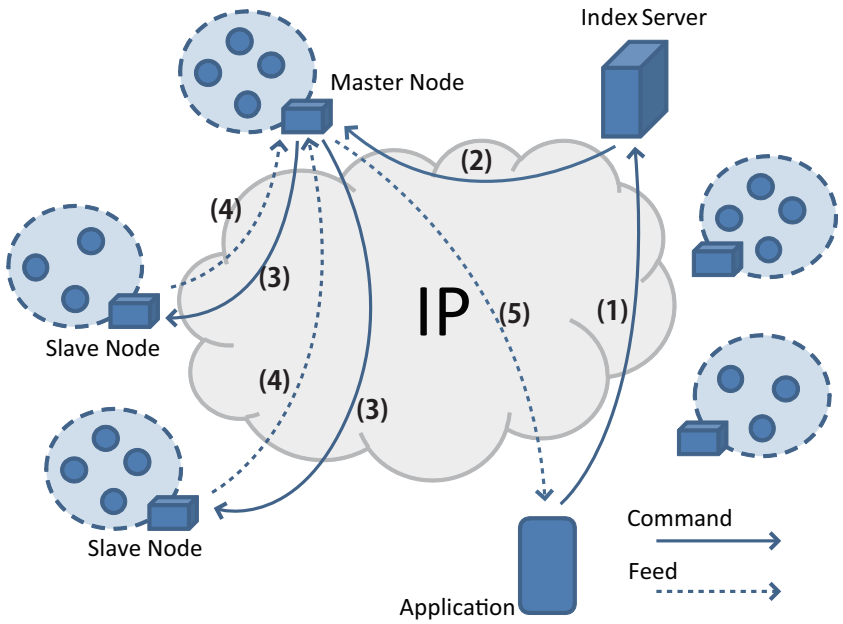

Figure 4.8: Procesures of SensingCloud Scaling Mechanism

As for the master client, the client which located at the closest to the centroid coordinates of the slave clients since data transaction can be completed in smaller area. We have chosen centroid coordinates as a barometer of the closeness, since the location of slave clients can vary as shown in Figure 4.9.

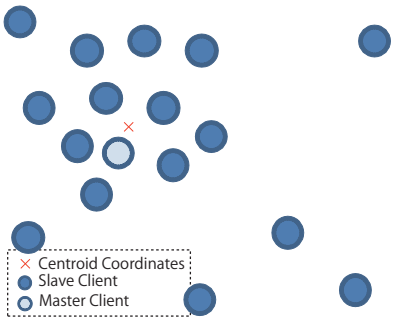

Figure 4.9: Location of Slave Client and Master Client

SensingCloud Scaling Mechanism has two types of scaling, namely "horizontal scaling" and "vertical scaling" as shown in Figure 4.10,4.11. In the figure, each circle represents clients, and the square at the bottom represents user application. Two different mechanism will be described below.

- Horizontal Scaling

  Horizontal scaling is for assigning the client in charge of the aggregation task for each request. This is what we have introduced above. The idea is shown in Figure 4.10.

- Versical Scaling

  Vertical scaling is for task distribution in one request when the data to process is too large for a single master client to aggregate. The idea is shown in Figure 4.11. This has a hierarchical structure. A master client is also a slave client of another master client in higher level. This ensures the responsiveness of aggregation task even if a number of target feeds increases.

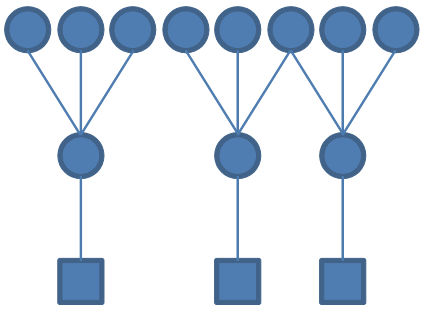

Figure 4.10: Horizontal Scaling

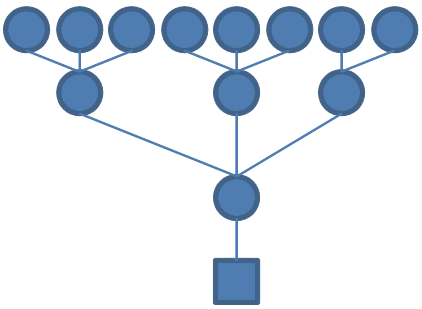

Figure 4.11: Vertical Scaling

Spontaneous Participation in SensingCloud

Sensor networks are usually deployed for a specific reason. SensingCloud integrates those sensor networks and make them a single open platform. To make SensingCloud large enough to provide global awareness, it has to be easy for administrators to participate in the system. For those reason, we have developed sensor network management software as a front end of distributed process client. This helps sensor network administrators to manage their local sensor network and at the same time, enables the sensor network to be connected to SensingCloud.

4.4 System Architecture

4.4.1 Hardware Architecture

The hardware architecture of SensingCloud is shown in Figure 4.12. Index Server is running on a server computer that has high computational resource. Distributed Process Clients are running on participants' PC. The PCs are connected to a sink node which collects sensor feeds from its sensor network. Sensor networks which participating in SensorCloud have variety of sensor network communication protocols. One may have zigbee, and one may have bluetooth. Data from those heterogeneous sensor networks is bridged to IP network through Distributed Process Clients. We assume mobile sensors which do not belong to sensor network, for example sensor on a cars or cell

phones, are IP reachable.

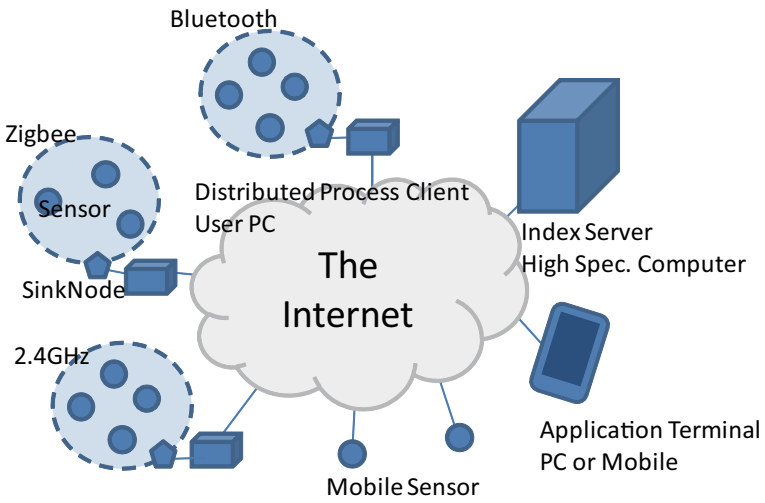

Figure 4.12: Hardware Structure of SensorCloud

4.4.2 Software Architecture

Figure 4.13 shows software architecture for SensingCloud. In the Index Server, Data Resolver Server and Sensor Database is running. In client nodes, Distributed Process Client and Sensor Proxy are running. Sensor Proxy is the process that is directly receives feeds via Sink Node from sensor network. Users must implement Sensor Proxy unless they use well known sensors since it is hard for Distributed Process Client to understand the raw packet from sensors. Sensor Proxy parses raw packet from sensors and formats the sensor feed according to sensor abstraction xml introduced in XML 4.3.1. The xml is sent via IP network to Distributed Process Client. Feeds between Distributed Process Client are utilizing UDP. On the other hand, commands and replies such as Aggregation Request, Feed Request, and Reply/Request to User Applications are utilizing TCP.

The Figure 4.14 shows software structure for back-end process of Distributed Task Processer. The software is composed of 2 major components which are “Aggregation Manager” and “Feed Manager”. Firstly, Aggrega-

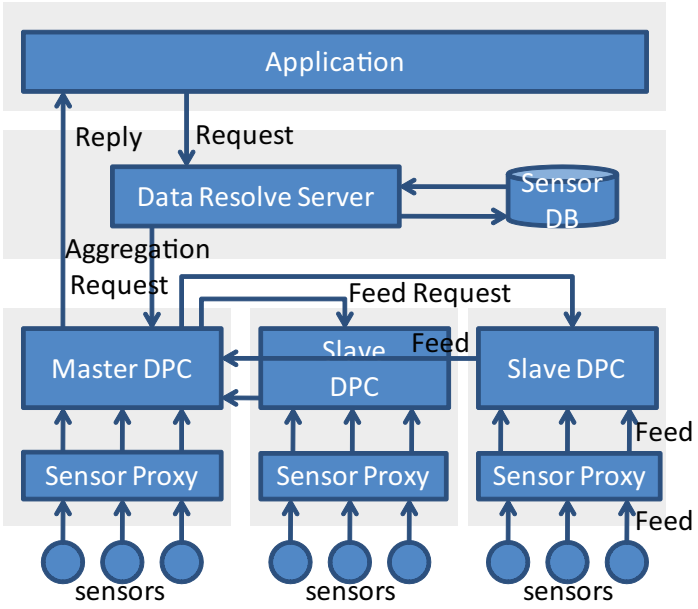

Figure 4.13: Software Architecture of SensorCloud

tion Manager is a component that runs when the Distributed Task Processer is assigned to be a master task processor. When Aggregation Manager receives the request, it creates a new thread instance of "Aggregation Processer". This thread 1) sends "Feed Request" to Feed Manager of relevant Distributed Task Processer, 2) receives the feeds from them and 3) aggregate and reply the result to a user application. On the other hand, Feed Manager is the component which runs when the distributed task processor is assigned to be a slave one. When Feed Manager receives Feed Request from Aggregation Process, it creates a new thread instance of "Feed Process". This process is connected to a local sensor network and forwards the incoming feeds which matches to the application's interest described in the Feed Request. By distributing this software to sensor networks in the world, they will collaborate each other and create SensingCloud.

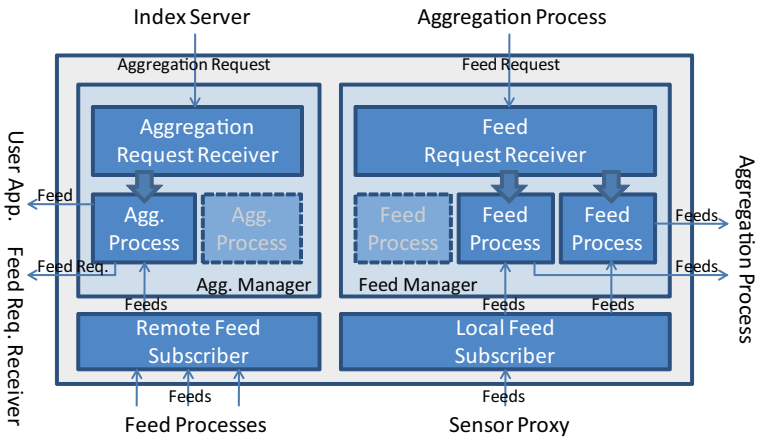

Figure 4.14: Distributed Task Processer

4.5 Summary

In this chapter, we proposed and described design approaches to satisfy the requirements of SensingCloud. The approach includes "Transparent Cross Domain Communication", "Meta-Level Request Processing" and "Distributed Data Aggregation".

As for Transparent Cross Domain Communication, we focused on 2 heterogeneity problem, which are "Network Heterogeneity" and "Device Heterogeneity". We proposed that IP should bridge the distributed sensor network all over the world to solve network heterogeneity. Also, we have defined sensor device abstraction as a solution of device heterogeneity.

In Meta-Level Request Processing, we proposed "Location Based Sensor Discovery", "Characteristic Value Processing" and "Query Design".

We designed index sensor resolver server that resolves application's meta request with geographical location, and sensor type in Location Based Sensor Discovery section. We have discussed the abstraction level of data that SensingCloud should provide to application in Characteristic Value Processing layer. Then, we have designed query to satisfy the approach.

In proposal of Distributed Data Aggregation, we have discussed the design of the module, and decided to leverage Hybrid P2P technology and

distributed computation model. Also, we realize spontaneous participation for this open and global sensor network by distributing client software for SensingCloud with sensor network administration software. At the end of this chapter, we have introduced both hardware and software architecture for SensingCloud. In the next chapter, we will introduce details of software implementation of SensingCloud.

Chapter 5

Implementation

5.1 Introduction

In this section, we will describe how implemented SensingCloud. We have implemented the system components with Java. Since world's sensor network administrators will use client software of SensingCloud, Java is an appropriate for its feature of platform independency. First, we will introduce the implementation of common models for SensingCloud. Then, we will explain the software implementation of Distributed Process Clinet and Index Server.

5.2 Data Models

In this section, we will introduce the major and common models for the system, namely "Sensor", "SensorFeed" and "SensorData". They are the data models that are commonly used in all components. The class diagrams for each class are shown in Figure 5.1. They all extends super class which represents the abstract form of each class.

5.2.1 Sensor

"Sensor" class represents a sensor as the name of the class says. Currently we have only focused on sensors which provides numerical data. Sensor

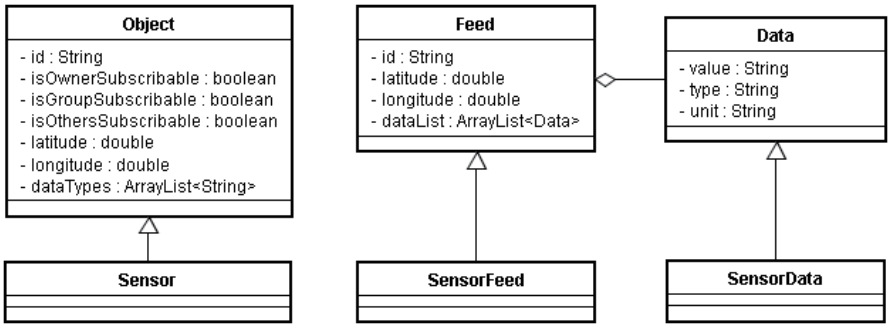

Figure 5.1: Class Diagrams for Common Models

class supports this type of sensor. However, in the future, we are considering that to support other type of data such as stream data. Therefore, sensor class is a subclass of abstract class "Object", and overrides serialize method, since we have to change serialize description to support some other type of data. Object has its id, and location information described as latitude and longitude. Also, it contains access authority fields described as isOwnerSubscribable, isGroupSubscribable and isOthersSubscribable. We have designed this model after the UNIX file authorization model.

5.2.2 SensorFeed

"SensorFeed" class represents a set of data that an instance of Sensor class produces. Therefore, if a sensor has thermometer and accelerometer, a sensor feed contains temperature and acceleration data. For the same reason that Sensor class is a subclass of Object class, SensorFeed class is a subclass of "Feed". Feed has its sequential Id, and longitude and latitude as its property. And it has a list of instances of Data.

5.2.3 SensorData

"SensorData" class is a model for data that consists a SensorFeed. Since one sensor can have more than one sensing module, one SensorFeed can have more than one instance of SensorData class. Again, with the same reason,

SensorData has a super class named "Data". Data has 3 fields namely, value, type and unit. Type represents a type of data such as temperature, and unit represents the scale of the value, which is Fahrenheit Celsius or Fahrenheit.

5.3 Distributed Process Client

Figure 5.2 shows the screen dump of distributed process client's front end. Orange points on the map represent sensors. In sensor add mode, the dialog will pop up when a user clicks on a map. The user has to type sensor id, and the feed's data type, and access property information. On the left pane, the user can see sensor list which the distributed process client manages.

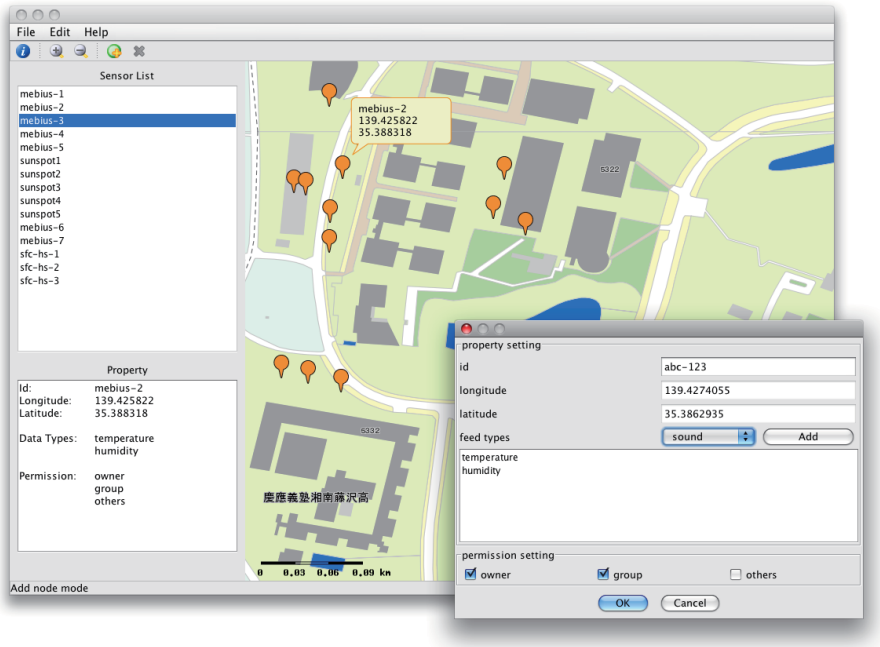

Figure 5.2: Screen Dump for Distributed Process Client

When sensor network administrator added a new sensor using the tool, it registers the sensor information to Sensor Network Resolver running on the Index Server. The registration for SensingCloud is done by this procedure.

As long as this software is running with IP connectivity, the client is a part of computational resource too.

5.3.1 Software Implementation

Figure 5.3 shows a part of class diagram for Distributed Process Client. This diagram shows the main module of Distributed Process Client. Class named "DistributedProcessClientManager" in the figure, manages all GUI, data aggregation, and data forwarding. Classes for GUI components are not in the Figure, since we do not explain GUI modules in this section. DistributedProcessClientManager has 2 major class named "AggregationManager" and "FeedManager". AggregationManager has a list of "AggregationProcess" whose instance is created when "AggregationRequest" is received. Also, FeedManager has a list of "FeedProcess" whose instance is created when "FeedRequest" is received. "FeedSubscriber" is a class for subscribing local and remote sensor feeds via IP network. The events of FeedSubscriber are observed by instances of AggregationProcess and FeedProcess. In following section, we will describe details of each class.

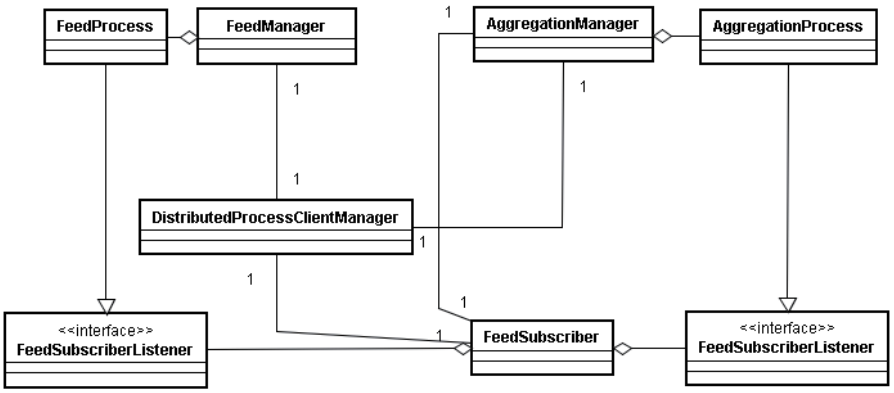

Figure 5.3: Class Diagram for Distributed Process Client

5.3.2 Aggregation Manager

Figure 5.4 shows a class diagram for AggregationManager. The class implements Runnable interface and run as a thread. In run() method, it waits for AggregationRequest and creates AggregationProcess. Here, listen port is statically assigned. AggregationRequest XML is shown in XML 5.3.2. If the value of "type" element in "property" block is "greatMaster", that means the request is for the AggregationProcess which provides aggregated sensor feed to the user application. If the element has value "master", the request is to be sent to AggregationProcess which sends its data to another AggregationProcess for vertical process distribution. "Destination" block is appeared only when the type represents "greatMaster" to let the AggregationProcess to know the network address of the user application.

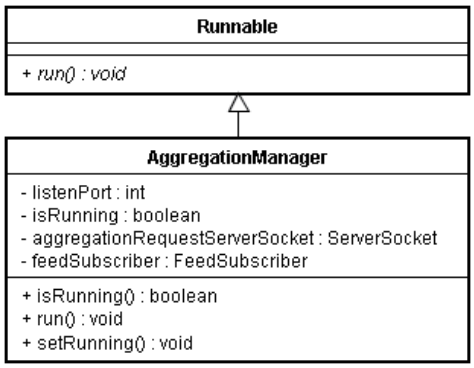

Figure 5.4: Class Diagram for AggregationManager

5.3.2AggregationRequest XML

```
<aggregationRequest>
    <property>
        <id>12</id>
        <type>greatMaster</type>
    </property>
    <destination>
        <ip>133.27.170.222</ip>
        <port>37456</port>
    </destination>
    <clients>
        <client type="slave">
            <ip>133.27.170.224</ip>
            <sensors>
                <sensor>iota-temp</sensor>
                <sensor>delta-temp</sensor>
            </sensors>
        </client>
        <client>...</client>
    </clients>
    <feedConditions>
        <feedCondition name="temp">
            <type unit="celcius">temperature</type>
            <condition type=">=">30</condition>
        </feedCondition>
        <feedCondition name="noise">
            <type unit="decibel">sound</type>
            <condition type="*"></condition>
        </feedCondition>
    </feedConditions>
    <feedAggregations>
        <feedAggregation name="temp">
            <interval>1500</interval>
            <operation>ave</operation>
        </feedAggregation>
    </feedAggregations>
</aggregationRequest>
```

5.3.3 Aggregation Process

A class diagram of AggregationProcess is shown in Figure 5.5. This class implements 3 interfaces, which are Runnable, FeedSubscriberListener and FeedRequestReceiverListener. AggregationProcess is an abstract class that is extended by "MasterAggregationProcess" and "NormalAggregationProcess". The difference between these two is whether it forwards the aggregated sensor feed to user application which is described in the AggregationRequest or AggregationProcess from which received FeedForwardRequest. MasterAggregationProcess is instantiate when type of aggregationRequest is "greatMaster" and NormalAggregationProcess is instantiate when the type is "master".

In their run() method, first, it generate instance of FeedRequestReceiver and start its thread. Then, it decides the number of listenport, and start feed subscriber on the port. After that, it sends FeedForwardRequest for other slave clients, and receives and aggregates sensor feeds from them and send them to the destination. The destination can be user application or the master client stands on upper hierarchy. The XML format for FeedForwardRequest is shown in XML 5.3.3

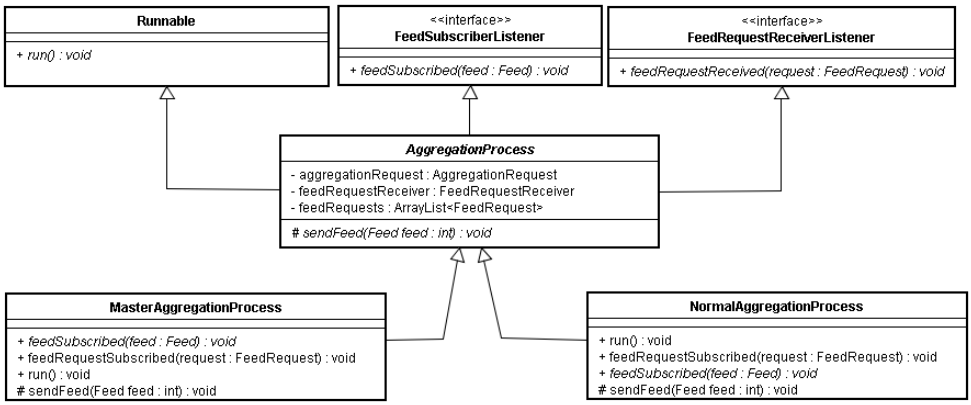

Figure 5.5: Class Diagram for AggregationProcess

5.3.3FeedForwardRequest XML

```
<feedForwardRequest>
    <property>
        <id>12</id>
        <type>start</type>
    </property>
    <sensors>
        <sensor>iota-temp</sensor>
        <sensor>delta-temp</sensor>
    </sensors>
    <feedConditions>
        <feedCondition>
            <type unit="celcious">temperature</type>
            <condition type="+=">30</condition>
        </feedCondition>
        <feedCondition>
            <type unit="decibel">sound</type>
            <condition type="-=">200</condition>
        </feedCondition>
    </feedConditions>
</feedForwardRequest>
```

5.3.4 Feed Manager

FeedManager is just like AggregationManager. Figure shows a class diagram for FeedManager. This class runs as a thread. In the thread loop, it waits for FeedForwardRequest, and when it receives the request, it generates FeedProcess. FeedManager has a FeedSubscriber which subscribe to sensor feeds from local sensor network. When the subscriber receives the feed, it notify the event to each instance of FeedProcess.

5.3.5 Feed Process

FeedProcess is the class which filters and forwards the feeds according to FeedForwardRequest. Class diagram is shown in Figure 5.8. FeedProcess implements FeedSubscriberListener. when local FeedSubscriber receives the

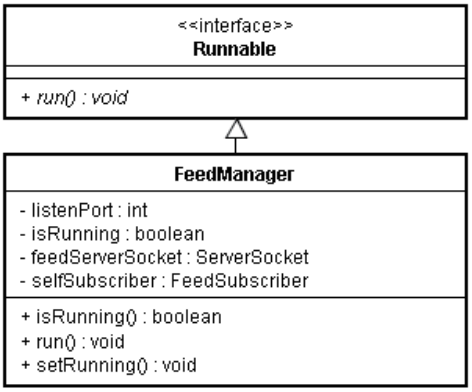

Figure 5.6: Class Diagram for FeedManager

sensor feed, the event will be notified with the feed to FeedProcess. In the call-back method of FeedProcess, it sends the feed to a master client.

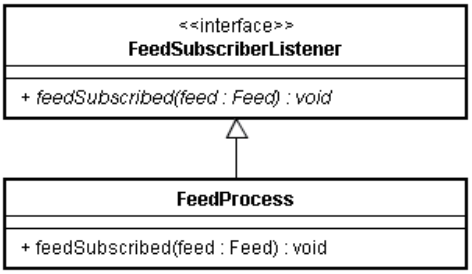

Figure 5.7: Class Diagram for FeedProcess

5.3.6 Feed Subscriber

FeedSubscriber subscribes to data via UDP. The class diagram is shown in Figure 5.8. The instance for this class can be 2 types. One is local feed subscriber and the other is remote feed subscriber. Local feed subscriber is for receiving feeds from sensor proxy. Remote feed subscriber is for feeds from another slave clients. Feeds are based on sensor feed XML shown in XML 4.3.1. As for feeds from local sensor proxy, they do not include location information since it can be found by matching id in Distributed

Process Client.

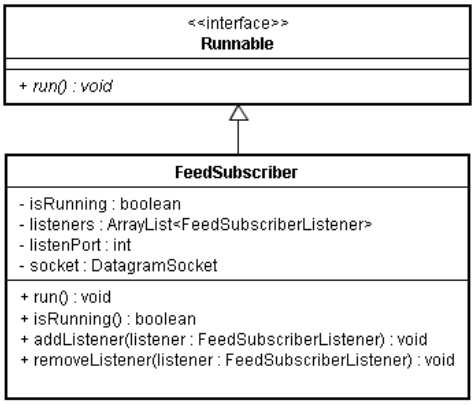

Figure 5.8: Class Diagram for FeedSubscriber

5.4 Sensor Index Server

"Sensor Index Server" manages the both geographic and network location of sensors and Distributed Process Clients. We have desigend small index server since in Sensing Cloud, the server's task is the smaller the better. The major components of the server are "Object Resolver", "Object Updater" and "Authorization Manager". A class diagram for the system is shown in Figure 5.9. Object Resolver resolves IP address of a target sensor network from user's meta-level request. Object Updater manages geographic and network location of mobile sensors. Authorization Manager manages login/out requests to SensingCloud from users.

5.4.1 Database Design

We have leverages PostGIS technology, which is a PostgreSQL database with GIS related functions. This has geometry type in nature, and can process GIS function in its query, such as getting distance between a point to a point. The database has tables as shown in Table 5.1. The relationship between tables is described in the Figure 5.10. To join SensingCloud, users

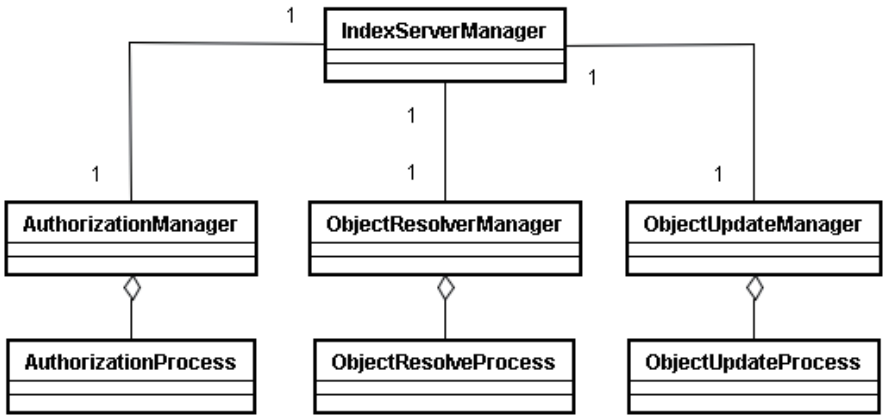

Figure 5.9: Class Diagram for Sensor Index Server

have to register their account to the database. The account information will be stored in "users" table. "network" table is separate from user table since a user can have more than one sensor networks. "ip_address" column represents the IP address of the Distributed Process Client. Every time the client goes on-line, the IP address is updated. Information of the sensors is stored in "objects" table. The uniqueness for a set of user_name and network_name is guaranteed. "location" column represents sensor's geographical location. "Geometry" is a type which represents longitude and latitude in PostGIS implementation.

Table 5.1: Table list for SensingCloud database

Name	Description
users	User account information
networks	Perticipant sensor network information
objects	Sensor and its location information
types	Registered data types
units	Registered data scale

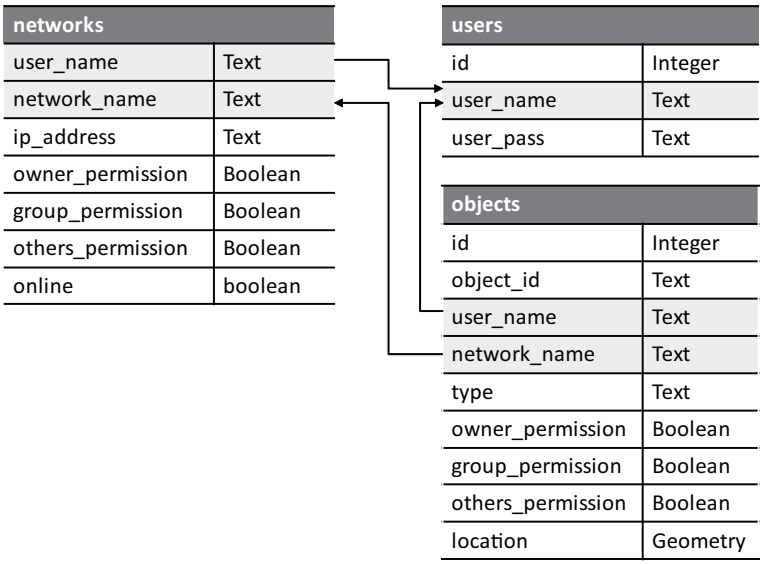

Figure 5.10: Relationship between tables

5.4.2 Sensor Resolver

A class for Object Resolver is "ObjectResolverManager" and "ObjectResolverProcess". When ObjectResolverManager receives a "Feed Request" which is a request from an user application, it creates an instance of ObjectResolverProcess and handle the request.

Basically, this process 1) finds the IP addresses of Distributed Process Clients which contain the sensors with target type located in target area, 2) choose a master client and slave clients out of them, and inform the master's IP address to the user application and 3) sends Aggregation Request for a master client(s). Following query shown in Figure 5.11 is to retrieve clients which contain target sensor nodes located within a target area for Sensing Cloud database shown in Figure 5.10. By these 3 steps, the object resolver's task is completed. The rest of the tasks to make SensingCloud work is responsible for the Distributed Process Clients. The task and communication steps after ObjectResolver is described in Figure 5.13. Sequence diagram

for the communication is shwo in Figure 5.12.

```
SELECT object_id and network_name FROM objects
WHERE ST_DWithin(ST_Transform(
                    ST_GeomFromText('POINT(139 36)',4326),32653),
                    ST_Transform(location,32653), 5000 )
AND (type= 'temperature' OR type = 'sound' );
```

Figure 5.11: PostGIS query for resolving network from meta condition

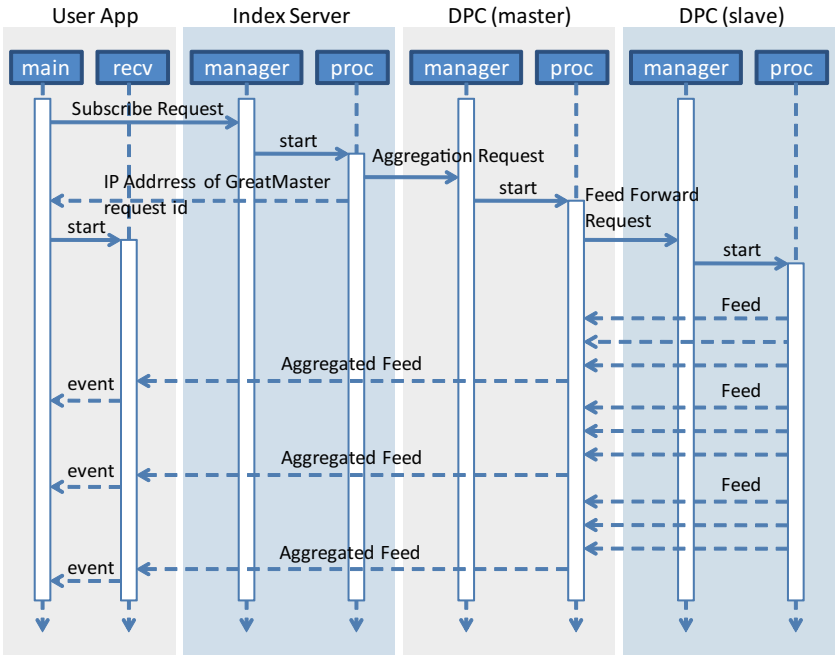

Figure 5.12: Sequence diagram for task assignation

The static number '4326' represents id for geographical coordinate system. '4326' is for WGS84, which is the most common geographical coordinate system for GPS data. Another static number, '32653', is for SRID(Spatial Referencing IDentifier). Then, the last static number, '5000', represents radius for the target area. Therefore, the query retreive network names which contains sensors with temperture and sound sensor within 5000m from the geographic coordinates(139,36).

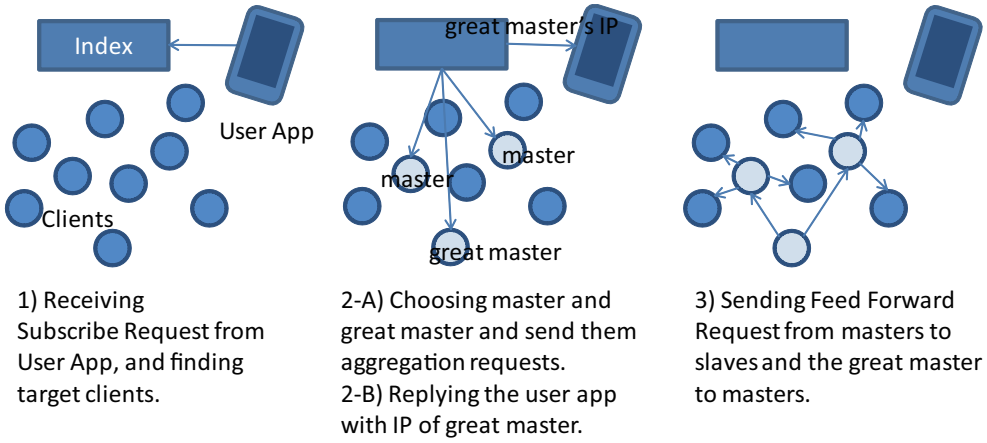

Figure 5.13: Tasks and communications after SensorResolver tasks

However, this server can execute little smarter task. It divides the requested target area and chooses master clients for each area and slave clients for each master client from Distributed Process Clients which contains target sensor nodes. The manager chooses one of the master clients to be a great master client which aggregates feeds from the other master clients and provides aggregated feeds to the user application. The IP address which is notified to the user application is the IP address of this great master client.

5.4.3 Update Manager

Sensors such as speed sensor on vehicles or accelerometer on cell-phones are changing its location time to time. The location information in database needs to be updated as they move. As described above, we are assuming that those mobile sensor nodes have the internet connectivity. Therefore, the server has a module to update the location of sensors named “Location Update Manager”. The module receives location update request directly from the mobile sensors. The request XML is shown in XML 5.4.3.

5.4.3Location Update Request XML

```
<update>
    <network>naoya's mobile</network>
    <location>
        <longitude>139.33423</longitude>
        <latitude>32.35234</latitude>
    </location>
</update>
```

5.5 Summary

In this chapter, we have described detailed system implementation and communication protocols of Distributed Process Client and Index Server. Distributed Process Client consists of "AggregationManager" and "FeedManager" which create an instance of "AggregationProcess" and "FeedProcess" when they receive their requests. On the other hand, Index Server consists of 3 components, namely "AuthorizationManager", "ObjectResolver" and "ObjectUpdateManager". In the next chapter, we will evaluate performance of SensingCloud based on this implementation.

Chapter 6

Evaluation

6.1 Introduction

In this section, we have done an evaluation to verify that the architecture and implementation of SensingCloud can perform better in performance than different type of architecture, such as high performance computer based architecture. First, we will explain detailed purpose, and environmental settings of the evaluation. Then we will show the result and analyze the result.

6.2 Purpose of Evaluation

In this research, we have adopted the architecture using P2P and distributed computing technology for improving performance of sensing cloud. In this section, we will evaluate the performance of current design and implementation of SensingCloud, To compare the performance and scalability of current design and implementation, we have developed a software which performs the same as SensingCloud with another architecture, which is centralized server-client model based on high-performance computer. We named it "SensingCenter". By comparing the architecture of SensingCloud to that of SensingCenter, we can not only verify the appropriateness of the architecture, but also find strong and weak point of SensingCloud.

6.2.1 Environment Settings

Hardware Settings

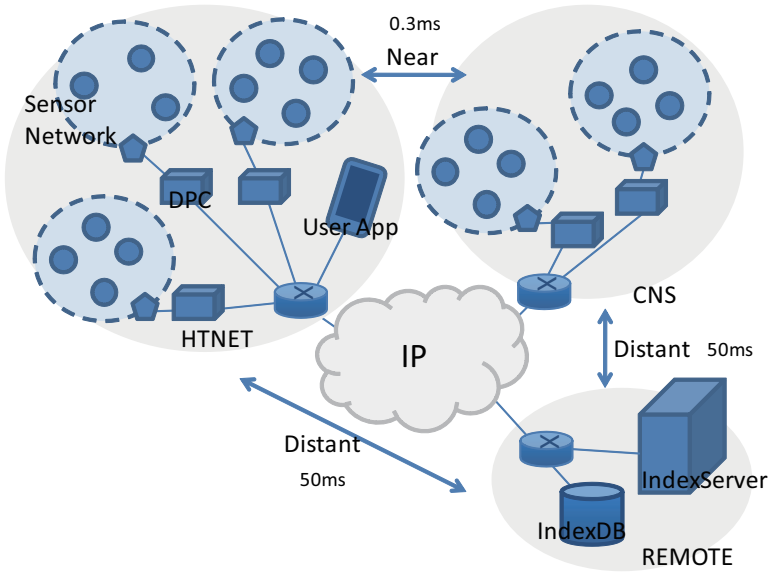

Figure 6.1: Evaluation Environment Settings

Figure 6.1 shows hardware environment of this evaluation. We have virtually emulated 3 different network environments, namely HTNET, CNS and REMOTE. All three networks actually exist and we emulated the network environments with RTT between these networks by using DummyNet. HTNET and CNS are networks placed in Keio University, SFC. Therefore, these two network are geographically located near by. Referring to results of RTT from a node in actual HTNET to a node in actual CNS, which are approximately 0.6 ms, we have set 0.3 ms delay between these two. On the other hand, network named REMOTE is assumed to be located at remote place. To communicate between the node placed in REMOTE network and HTNET or CNS, we have set 50ms latency. We placed 3 Distributed Process Clients in HTNET, 2 in CNS. Sensor Networks connected to these Nodes are virtually emulated. Each sensor network has 100 sensor nodes and each node is sending one temperature feed per 1 second. User application which

queries sensor feeds is located in HTNET.

Hardware specification for each node is shown in Figure. Basically, Distributed Process Clients are low-mid price computers such as Mac mini, MacBook air, or MacBook Pro. On the other hand, Index Server is high performance machine, such as MacPro. Figure is the hardwares used for this evaluation. 6.2

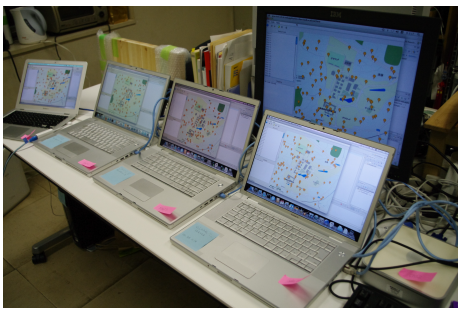

Figure 6.2: Evaluation Settings

Table 6.1: Specification of Hardware

Name	CPU	Memory	OS	Description
MacBook Air	Core2Duo 1.86GHz	2GB	MacOSX 10.5.8	DPC (HTNET)
MacBook Pro	Core2Duo 2.4GHz	4GB	MacOSX 10.5.8	DPC (HTNET)
MacBook Pro	Core2Duo 2.33GHz	3GB	MacOSX 10.6	DPC (HTNET)
Mac Mini	PowerPC G4 1.42GHz	1GB	MacOSX 10.5.8	DPC (CNS)
MacBook Pro	Core2Duo 2.6GHz	4GB	MacOSX 10.5.2	DPC (CNS)
MacPro	Quad-Core Xeon 2x2.8GHz	4GB	MacOSX 10.5.6	IndexServer
ThinkPad t43p	Pentium M 2.13GHz	2GB	Linux Ubuntu 9	Database
MacBook Pro	Core2Duo 2.93GHz	8GB	MacOSX 10.5.8	User App

Software Settings

We run SensingCloud on this hardware environment. SensingCenter, the competitor software which is designed with centralized server-client architecture, is also runs on this architecture. In SensingCenter, Distributed Process Clients do not take aggregation process; only forwards sensor feeds

of their underlying sensor network. Index Server runs all aggregation process instead. SensingCenter has "CentralManager" and "CentralProcess". CentralManager receives FeedSubscribeRequest and create CentralProcess. The instance of CentralProcess, which runs as a thread, 1) receives FeedRequest from the user application, 2) resolves target Distributed Process Client to collect data from, 3) sends FeedForwardRequest to them 4) aggregates feed and 5) sends the aggregated feeds to the user application. The sequence diagram is shown in Figure 6.3.

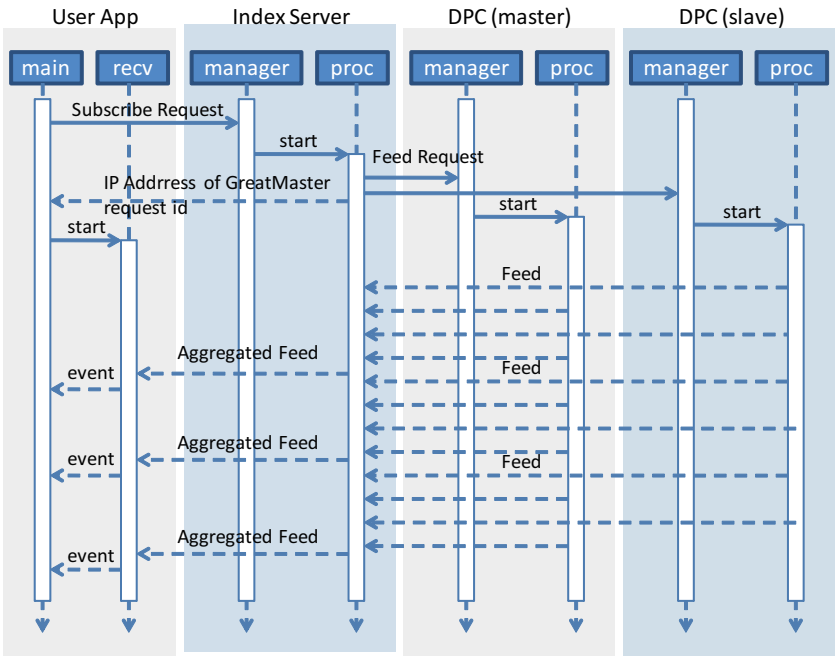

Figure 6.3: Sequence Diagram for SensingCenter

6.2.2 Procedure

The evaluation procedure is as follows.

1. Sends a subscribe request to SensingCloud/SensingCenter.
2. Receive the first sensor feed.
3. Time the interval between step.1 and step.2.

4. Repeat the step.1 to step.3 until it gets 100 times.

The subscribe request we used is shown in XML 6.2.2. According to the request, the first sensor feed will be arrived at the user client in 1000 milliseconds + latency. Therefore, the latency of SensingCloud/SensingCenter can be calculated by (Interval between 1 and 2) - 1000. We do the steps described above for 5 times and average the time consumption.

6.2.2Feed Request XML

```
<request>
    <client>
        <ip>133.27.170.224</ip>
        <port>23456</port>
    </client>
    <geography>
        <area type="radius">
            <value type="m">1000<value>
        </area>
        <longitude>139.427326</longitude>
        <latitude>35.38827</latitude>
    </geography>
    <feedConditions>
        <feedCondition name="temp">
            <type unit="celcius">temperature</type>
            <condition type="*">0</condition>
        </feedCondition>
    </feedConditions>
    <feedAggregations>
        <feedAggregation name="temp">
            <interval>1000</interval>
            <operation>IDW</operation>
            <args>139.427324,35.38829:139.427324,35.38823:....</args>
        </feedAggregation>
    </feedAggregations>
</request>
```

The requested operation is IDE, which is an established interpolation method utilized for geocentric sensing. IDE interpolates the value of an

arbitrary points based on the observed value around the points utilizing each distance to a point as a calculation weight. This request can burden the server because this operation requires much iteration. This xml requests for the system to interpolate arbitrary 20 points with IDE algorithm.

6.3 Performance Evaluation

To verify the appropriateness of the distributed architecture in performance, we have executed the procedure for both SensingCloud and SensingCenter. After measuring each performance, we plotted the average performance for each trial on Figure 6.4.

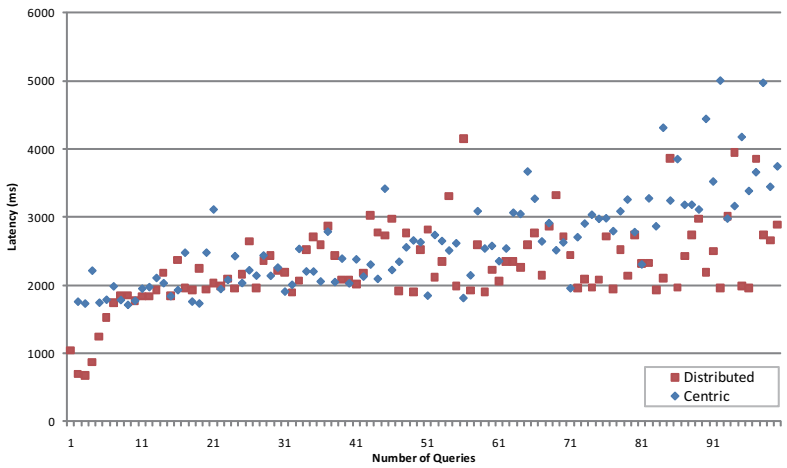

Figure 6.4: Result on Performance Evaluation

Blue diamond-shaped plots represent the results of SensingCenter, and red square-shaped plots represent the results of SensingCloud. For the beginning, both results are very similar each other. However, after around 70th query, the performance of SensingCenter gradually goes up while the performance of SensingCloud keeps its performance. By this evaluation, we found that distributed architecture which SensingCloud has adopted can perform better in this sale of global sensor network.

6.4 Scalability Evaluation

To verify the distributed architecture enables autonomous scalability, we focused only on the performance of SensingCloud and executed the procedure for network with 3 Distributed Process Client and 5 Distributed Process Client. The result is shown in Figure 6.5.

Blue diamond-shaped plots are the performance for the network with 3 clients and Red square-shaped plots represents the performance for the network with 5 clients. We can observe the network with 5 clients performs the similar performance transition to that of the network with 3 clients does. SensingCloud with 5 Distributed Process Clients performs better; even the number of sensor feeds to process is larger. We can observe that SensingCloud scales with the scale of the network.

In this evaluation case, the performance turned to be better since increments of the processing power was larger than the processing power required to sustain performance with the increments of sensor feeds.

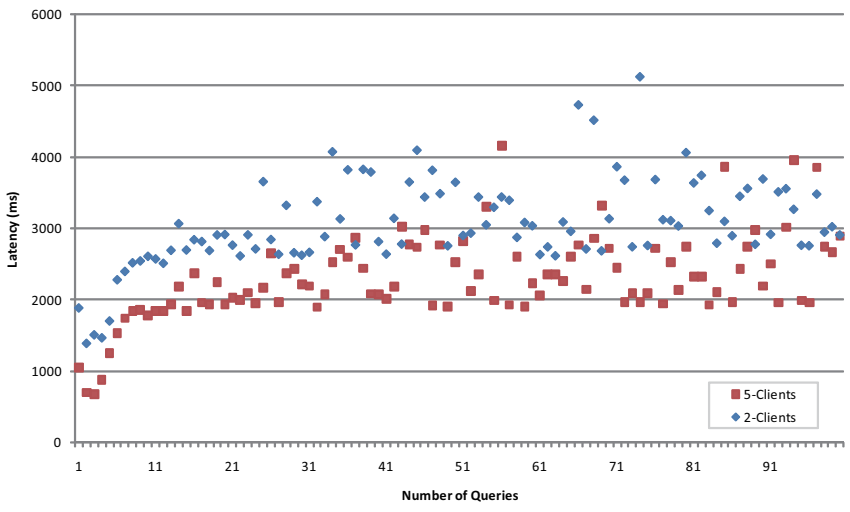

Figure 6.5: Result on Scalability Evaluation

6.5 Summary

In this chapter, we evaluate and compare the performance of two softwares which act the same for user application. One software is SensingCloud, which leverages P2P and distributed computing methods. The other software is what we named SensingCenter, which does same thing as SensingCloud but it adopts centric server-client approach. We could find out that SensingCloud can keep this performance with much larger setting athough SensingCenter is better architecture in this small evaluation setting.

Chapter 7

Conclusion and Future Works

In this chapter, we conclude this thesis by summarizing overall contribution of this research and discussing the future work of SensingCloud.

7.1 Conclusion

In this thesis, we have organized am architecture of current sensor network system and pointed out a problem that all sensor networks are closed to the others. We focused on the problem, which prevents future expansion of the sensor network systems, and proposed an architecture named SensingCloud to share sensor data streams of sensor networks distributed all over the world with automatic scalability. The contribution of this research is as follows.

- **Modeling Current Sensor Network System**
  We have organized current sensor network systems into 3 layers, which are, from the bottom, Sensor Network Layer, Data Processing Layer and Application Layer. Sensor Network Layer includes hardware, network protocols and etc. Data Processing Layer contains three sublayers, which are Raw Data Layer, Characteristic Data Layer and Context Data Layer. A task such as unit translation is done in Raw

Data Layer. Basic and well-known numerical calculations, from averaging to interpolation, are done in the Data Processing Layer. Context Layer is to process characteristic data and convert it into an abstract form such as situation, emotion or atmosphere. At last, the instances of Application Layer are user applications. Currently, a developer of sensor network system needs to implement all instances of the three layers. This situation not only decreases development efficiency but also prevents user applications to subscribe to data streams of the sensor networks owned by others.

- **Modeling Future Sensor Network Architecture**
  In the future, sensor networks distributed all around the world should share its sensor data stream and allow user applications to acquire the global awareness. To realize this, a common sensor network platform is needed. Referring to the sensor network architecture, the common platform should include Sensor Network Layer and a part of Data Processing Layer, which are Raw Data and Characteristic Data Layer. Since Context Layer has an infinite possibility of instance, it is too heavy for the common platform to include it. Also, the user application would not be able to handle an infinite possibility of answers. Therefore, Middleware or Application should include Context Layer.

- **Proposing an Architecture for Open and Global Sensor Network**
  To create the common platform, we organized the requirements of the platform and proposed architectural approach. The requirements are 1) transparent connectivity to distributed sensor network, 2) meta-level accessibility to arbitrary data and 3) automatic scalability for growing number of sensors. We propose SensingCloud, which is the common sensor network platform with the required functions. SensingCloud leverages P2P and distributed computing method and enables transparent access with meta-level query.

7.2 FutureWork

On developing SensingCloud, we have found several technical problems. For future SensingCloud development, the following future works are necessary.

7.2.1 Serverless Architecture

SensingCloud leverages Hybrid P2P system architecture. In the architecture, a peer finds a target peer with centric index server, and starts peer to peer communication. Since the centric server is a single point of failure, the robustness decreases. Also, the time consumption for searching target sensors from the database can be bottleneck. Especially in our case, this is critical since we used geocentric queries, which needs more computational resource. To solve this problem, we would like to leverage Distributed Hash Table (DHT) [27, 28, 29] in the future implementation of SensingCloud. However, DHT only allows finding contents with the name completely match to the request. In our case, complete match is nearly impossible since contents name includes geographical coordinates. To develop a mechanism to allow DHT to find sensors based on geographical coordinates is also future work.

7.2.2 Security and Restriction

In the implementation of SensingCloud, the security issue is out of focus. However to use this platform in practice, security issue is imperative. For example, current implementation of SensingCloud is vulnerable for DOS attacks. Since hardware which executing aggregation process are low-middle performance computers, CPU utilization can easily gets up to 100% with DOS attacks even if the node is randomly chosen from several candidates. This can happen with not only malicious attempts but also with normal innocent utilization. Access control based on CPU utilization or network throughput is needed. Also, fake sensors that provide random value or tampered value with some purpose can exist in SensingCloud. Filtering

method against data from fake sensors is needed for SensingCloud to be Trustworthy sensor network platform.

Acknowledgments

First of all, I would like to thank my advisor, Professor Hideyuki Tokuda. Professor Tokuda always took time out of his extremely busy schedule to support, guide, and encouraged me.

I would also like to thank Professor Kazunori Takashio, Dr. Jin Nakazawa, Dr. Masayuki Iwai, Dr. Junichi Yura, and Dr. Masaki Ito for their daily supports and valuable advices.

And I am especially grateful to Yoshiyuki Tokuda, Jumpei Yamamoto, Yuki Yonezawa, Ken Mochizuki, Khohei Saijo and all other members of "Keio Media Space Family" and "i208" for providing great support while I was writing this thesis.

I am deeply grateful to Katsuya Hashizume, Kyohei Kawada, Tomotaka Ito for the most exciting, adventurous, hilarious and productive 5 years in my life. We have had the most intellectual discussions in the world as well as the stupidest conversation in the whole world.

Huge thanks to all the members of Hide Tokuda & Kaz Takashio Laboratory for making my life at the laboratory filled with fun and excitement.

Last and most importantly, I would like to thank my family and Erika Shizume for encouraging and supporting me all the time.

February 15, 2010

Naoya Namatame

Bibliography

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38:393–422, 2002.

[2] Mohammad Ilyas and Imad Mahgoub. *Smart Dust: Sensor Network Applications, Architecture and Design*. CRC, 1 edition, 2006.

[3] J. M. Kahn K. S. J. Pister and B. E. Boser. Smart dust: Wireless networks of millimeter-scale sensor nodes. *Highlight Article in 1999 Electronics Research Laboratory Research Summary*, 1999.

[4] Ito Masaki, Yukiko Katagiri, Mikiko Ishikawa, and Hideyuki Tokuda. Airy notes: An experiment of microclimate monitoring in shinjuku gyoen garden. In *Proceedings of INSS '07, Fourth International Conference on Networked Sensing Systems, 2007*, pages 260–266, 2007.

[5] M. Marin-Perianu and P.J.M. Havinga. D-fler - a distributed fuzzy logic engine for rule-based wireless sensor networks. In *Ubiquitous Computing Systems*, pages 86–101, Germany, November 2007. Springer Verlag.

[6] Jonathan Lester, Tanzeem Choudhury, and Gaetano Borriello. A practical approach to recognizing physical activities. In *Pervasive '06: Proceedings of 4th International Conference on Pervasive Computing*, pages 1–16, 2006.

[7] Ryan Aipperspach, Elliot Cohen, and John F. Canny. Modeling human behavior from simple sensors in the home. In *Pervasive '06: Proceedings*

of 4th International Conference on Pervasive Computing, pages 337–348, 2006.

[8] Stephen S. Intille, Kent Larson, Emmanuel Munguia Tapia, Jennifer Beaudin, Pallavi Kaushik, Jason Nawyn, and Randy Rockinson. Using a live-in laboratory for ubiquitous computing research. In *Pervasive '06: Proceedings of 4th International Conference on Pervasive Computing*, pages 349–365, 2006.

[9] Toshio Hori, Yoshifumi Nishida, Hiroshi Aizawa, Shinichi Murakami, and Hiroshi Mizoguchi. Distributed sensor network for a home for the aged. In *Proceedings of 2004 IEEE International Conference on Systems, Man and Cybernetics*, pages 1577–1582, 2004.

[10] Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *Int. J. Ad Hoc Ubiquitous Comput.*, 2(4):263–277, 2007.

[11] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *WMCSA '94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.

[12] Jun'ichi Yura, Hideaki Ogawa, Taizo Zushi, Jin Nakazawa, and Hideyuki Tokuda. objsampler: A ubiquitous logging tool for recording encounters with real world objects. In *RTCSA '06: Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 36–41, Washington, DC, USA, 2006. IEEE Computer Society.

[13] Liqian Luo, Aman Kansal, Suman Nath, and Feng Zhao. Sharing and exploring sensor streams over geocentric interfaces. In *GIS '08: Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, pages 1–10, New York, NY, USA, 2008. ACM.

[14] Aman Kansal, Suman Nath, Jie Liu, and Feng Zhao. Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia*, 14(4):8–13, 2007.

[15] Anthony Rowe, Berges Mario, Gaurav Bhatia, Ethan Goldman, Ragunathan Rajkumar, Lucio Soibelman, Garrett James, and Jośe Moura. Sensor andrew: Large-scale campus-wide sensing and actuation. Technical report, Carnegie Mellon University, 2008.

[16] Anthony Rowe, Mario Berges, Gaurav Bhatia, Ethan Goldman, Ragunathan Rajkumar, and Lucio Soibelman. Demo abstract: The sensor andrew infrastructure for large-scale campus-wide sensing and actuation. In *IPSN '09: Proceedings of the 2009 International Conference on Information Processing in Sensor Networks*, pages 415–416, Washington, DC, USA, 2009. IEEE Computer Society.

[17] P. Saint-Andre. Extensible messaging and presence protocol (xmpp): Core. *Network Working Group RFC 3920*, November 2009. `http://tools.ietf.org/html/draft-ietf-xmpp-3920bis-04`.

[18] Mohammad Mehedi Hassan, Biao Song, and Eui-Nam Huh. A framework of sensor-cloud integration opportunities and challenges. In *ICUIMC '09: Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication*, pages 618–626, New York, NY, USA, 2009. ACM.

[19] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. High Perform. Comput. Appl.*, 15(3):200–222, 2001.

[20] Karl Aberer, Manfred Hauswirth, and Ali Salehi. Global sensor network. Technical report, LSIR, Ecole Polytechnique Fédérale de Lausanne, 2008.

[21] J.L. Hill and D.E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE micro*, 22(6):12–24, 2002.

[22] J. Beutel, O. Kasten, and M. Ringwald. Poster abstract: Btnodes–a distributed platform for sensor nodes. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 292–293. ACM New York, NY, USA, 2003.

[23] Yoh Shiraishi, Niwat Thepvilojanapong, Yosuke Tamura, Tatsuro Endo, Koichi Yamada, Nayuta Ishii, Hiroki Ishizuka, Keisuke Kanai, and Yoshito Tobe. Tomudb: multi-resolution queries in heterogeneous sensor networks through overlay network. In *SenSys*, pages 419–420, 2007.

[24] Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM national conference*, pages 517–524, New York, NY, USA, 1968. ACM.

[25] Jason Campbell, Phillip B. Gibbons, Suman Nath, Padmanabhan Pillai, Srinivasan Seshan, and Rahul Sukthankar. Irisnet: an internet-scale architecture for multimedia sensors. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 81–88, New York, NY, USA, 2005. ACM.

[26] Ian Foster. The physiology of the grid: An open grid services architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.

[27] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, 2001. ACM.

[28] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Comput. Commun. Rev.*, 31(4):149–160, 2001.

[29] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Lecture Notes in Computer Science*, pages 329–350, 2001.