

**UCGE Reports
Number 20395**

Department of Geomatics Engineering

**GeoPubSubHub: A Geospatial Publish/Subscribe
Architecture for the World-Wide Sensor Web**

(URL: <http://www.geomatics.ucalgary.ca/graduatetheses>)

by

Chih-Yuan Huang

JANUARY, 2014



UNIVERSITY OF CALGARY

GeoPubSubHub: A Geospatial Publish/Subscribe Architecture for the World-Wide Sensor Web

by

Chih-Yuan Huang

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF GEOMATICS ENGINEERING

CALGARY, ALBERTA

JANUARY, 2014

© Chih-Yuan Huang 2014

Abstract

The World-Wide Sensor Web has become a useful technique for monitoring the physical world at spatial and temporal scales that were impossible in the past. With the development and deployment of sensor technologies and interoperable open standards, sensor web generates tremendous volumes of priceless data enabling scientists to observe previously-unobservable phenomena. By connecting various types of sensors located worldwide and performs observations at high-frequency, sensor web has the ability to capture time-critical events and provide up-to-date information to support efficient decision making. We argue that constructing sensor web infrastructure and applying interoperable standards are the first steps. In order to harvest the full potential of sensor web, efficiently processing sensor web data and providing timely notifications are necessary. Therefore, this research proposes *GeoPubSubHub*, a software component applying the publish/subscribe communication model to efficiently process geospatial sensor web data. In this thesis, we propose an overall system architecture to address the seven challenges identified for constructing a geospatial sensor web publish/subscribe system. In addition to the solutions that are similar to existing approaches, we propose *sensor web input adaptor*, *LOST-Tree*, *semantic layer service*, *AHS-Model*, and *sensor web browser* to address challenges that are most unique and critical in the context of a geospatial sensor web publish/subscribe system. Our evaluation results demonstrate that these proposed solutions can effectively address targeted challenges with efficient performance. As one of the first geospatial sensor web publish/subscribe systems, we believe that our proposed solutions and GeoPubSubHub architecture serve as a promising initiative to process sensor web data in a timely manner and will consequently harvest the full potential of the world-wide sensor web.

Acknowledgements

First, I would like to thank my supervisor Dr. Steve Liang for funding my research and giving me the opportunity to pursue my PhD degree in University of Calgary. Without his guidance and encouragement, this dissertation would not have been possible. I am deeply grateful to my committee of Dr. Xin Wang, Dr. Andrew Hunter, Dr. Jianxun He, and Dr. WenWen Li for their insightful comments and suggestions. I would also like to express my gratitude to CANARIE, Cybera, Alberta Innovates Technology Futures, and Microsoft Research for their financial support and opportunities to participate in many exciting projects.

I want to thank all the members in the GeoSensorWeb Lab, for helping me with my research and providing invaluable feedback. I would like to give special thanks to David Chang, who spent countless time answering my questions and giving me advice. I will always be grateful.

I want to show my greatest appreciation to Dr. Liang-Chien Chen, my previous supervisor, and all my professors in CSRSR. Their training has made it possible for me to excel in academics.

I would like to show my greatest appreciation to all the friends I met in Calgary. I am very lucky to know you guys. Thank you all for your kindest help. I would also like to thank my dear friends in Taiwan. Although we are far apart, I know you guys are always there for me.

Last but not least, I want to give a special thanks to my family, including Hui-Ling Hu and Yung-Fu Huang, my parents and my foundation. Their unconditional love and support are the reasons that I am where I am today. Words cannot express how much I love you all.

Finally, I owe my deepest gratitude to Han-Fang Tsai, the person I love and who has been extraordinarily tolerant and supportive all the time. Without you, I would not be able to achieve anything. I love you. I wish to dedicate this dissertation and all my achievements to you.

Table of Contents

| | |
|--|-----|
| Abstract | ii |
| Acknowledgements | iii |
| Table of Contents | iii |
| List of Tables | vi |
| List of Figures and Illustrations | vii |
| List of Symbols, Abbreviations and Nomenclature | ix |
| | |
| CHAPTER ONE: INTRODUCTION | 1 |
| 1.1 Big data phenomenon on the world-wide sensor web | 9 |
| 1.2 Problems | 11 |
| 1.3 Objectives | 16 |
| | |
| CHAPTER TWO: CHALLENGES | 18 |
| 2.1 Challenges and existing approaches in a general-purpose publish/subscribe architecture | 18 |
| 2.2 Identified challenges in a geospatial sensor web publish/subscribe architecture | 20 |
| | |
| CHAPTER THREE: METHODOLOGY | 26 |
| 3.1 Proposed solutions | 26 |
| 3.2 System architecture and processing steps | 31 |
| 3.3 Sensor web input adaptor | 36 |
| 3.3.1 Query aggregator | 38 |
| 3.3.2 Adaptive sensor stream feeder | 39 |
| 3.4 LOST-Tree | 42 |
| 3.4.1 Introduction | 42 |
| 3.4.2 Methodology | 47 |
| 3.4.2.1 Decompose step | 48 |
| 3.4.2.2 Filter step | 50 |
| 3.4.2.3 Update step | 52 |
| 3.4.2.4 Aggregate step | 52 |
| 3.4.2.5 Removal operation | 53 |
| 3.4.3 Contribution summary | 54 |
| 3.5 Semantic layer service | 55 |
| 3.6 AHS-Model | 59 |
| 3.6.1 Introduction | 59 |
| 3.6.2 Topological operators and DE-9IM | 61 |
| 3.6.3 Methodology | 66 |
| 3.6.3.1 Preparation stage: Generate necessary information from the geometries of subscriptions | 69 |
| 3.6.3.2 Intersection stage: Intersect with the geometry of publication | 74 |
| 3.6.3.3 Determination stage: Determine topological relationship | 76 |
| 3.6.4 Distributed AHS-Model | 80 |
| 3.6.5 Contribution summary | 86 |
| 3.7 Sensor web browser | 87 |
| 3.7.1 3D virtual-globe-based sensor web browser | 87 |

| | |
|---|-----|
| 3.7.2 2D map-based sensor web browser | 89 |
| CHAPTER FOUR: EVALUATION AND RESULTS | 91 |
| 4.1 Evaluation on the sensor web input adaptor | 91 |
| 4.2 Evaluation on the LOST-Tree..... | 95 |
| 4.2.1 Evaluation of data loading efficiency..... | 96 |
| 4.2.2 Evaluation of LOST-Tree size..... | 97 |
| 4.2.3 Evaluation of LOST-Tree performance..... | 100 |
| 4.2.3.1 Evaluation of LOST-Tree Performance on Different L_q | 103 |
| 4.2.3.2 Evaluation of LOST-Tree Performance on Different L_{gc} | 105 |
| 4.2.3.3 Evaluation of LOST-Tree performance on different combinations of L_q and L_{gc} | 107 |
| 4.2.4 Summary..... | 109 |
| 4.3 Evaluation on the AHS-Model | 110 |
| 4.3.1 AHS-Model scalability evaluation | 111 |
| 4.3.2 Evaluation of AHS-Model indexing performance | 113 |
| 4.3.3 Evaluation of AHS-Model matching performance..... | 116 |
| 4.3.4 Evaluation of AHS-Model end-to-end query performance | 119 |
| 4.3.5 Summary..... | 124 |
| CHAPTER FIVE: RELATED WORKS..... | 125 |
| 5.1 Related systems and their mechanisms..... | 125 |
| 5.1.1 Publish/Subscribe systems..... | 127 |
| 5.1.2 Data stream management systems (DSMSs)..... | 131 |
| 5.1.3 Summary..... | 136 |
| 5.2 Related approaches for the sensor web context | 136 |
| 5.2.1 Works related to sensor web input adaptor | 136 |
| 5.2.2 Works related to LOST-Tree..... | 138 |
| 5.2.3 Works related to AHS-Model..... | 140 |
| CHAPTER SIX: CONCLUSIONS AND FUTURE WORK | 143 |
| REFERENCES | 147 |

List of Tables

| | |
|--|-----|
| Table 2.1 Various URIs of the concept of wind speed. | 23 |
| Algorithm 3.1 The filter step. | 50 |
| Algorithm 3.2. The remove function. | 54 |
| Table 3.1 The topological relationships and the corresponding intersection matrices. | 65 |
| Table 3.2 The possible topological relationships between different geometry types (●: possible, ○: impossible, *: these relationships are possible if consider multi-point geometry). | 67 |
| Table 3.3 Necessary subscription indices for AHS-Model..... | 73 |
| Table 3.4 The topological relationships and the corresponding area matrices. | 79 |
| Algorithm 3.3. The worker selection algorithm..... | 82 |
| Table 4.1 Statistics of adaptive sensor stream feeder evaluation..... | 92 |
| Table 4.2 Settings of simulated scenarios for LOST-Tree evaluations. | 102 |
| Table 4.3 Query latencies with different combinations of L_q and L_{gc} (unit: millisecond)..... | 109 |
| Table 4.4 Filter efficiencies with different combinations of L_q and L_{gc} | 109 |
| Table 4.5 Number of requests with different combinations of L_q and L_{gc} | 109 |

List of Figures and Illustrations

| | |
|--|----|
| Figure 1.1 The sensor web layer stack..... | 3 |
| Figure 1.2 Examples of time-critical events: (a) 2007 Minneapolis Interstate-35W bridge collapse and (b) 2011 Japan earthquake and tsunami. | 8 |
| Figure 1.3 The high-level publish/subscribe workflow. | 13 |
| Figure 1.4 An example of query execution plan..... | 14 |
| Figure 3.1 High-level GeoPubSubHub system architecture and processing steps. | 32 |
| Figure 3.2 System architecture and workflow. | 37 |
| Figure 3.3 An example of adaptive sensor stream feeder algorithm. | 41 |
| Figure 3.4 Existing sensor data portals: (a) EarthScope (http://www.earthscope.org/); (b) SciScope (http://www.sciscope.org/); (c) National Data Buoy Centre (http://www.ndbc.noaa.gov/); (d) SensorMap (http://atom.research.microsoft.com/sensewebv3/sensormap/); (e) Sensorpedia (http://www.sensorpedia.com/). | 44 |
| Figure 3.5 LOST-Tree workflow..... | 48 |
| Figure 3.6. Quadtree-based tile system..... | 49 |
| Figure 3.7 High level architecture of the semantic layer service..... | 58 |
| Figure 3.8 Examples of AHS-Model indices with the forth level as the lowest quadtree level (interior: dark-gray, boundary: light-gray, and exterior: white). | 70 |
| Figure 3.9 The workflow of intersecting the geometry of publication. | 76 |
| Figure 3.10 High-level architecture and workflow of the distributed AHS-Model..... | 81 |
| Figure 3.11 The sequence diagram for registering a subscription in distributed AHS-Model. | 83 |
| Figure 3.12 The sequence diagram for matching a publication in distributed AHS-Model..... | 84 |
| Figure 3.13 A screenshot of the 3D virtual-globe-based sensor web browser. | 88 |
| Figure 3.14. A screenshot of the 2D map-based sensor web browser. | 90 |
| Figure 4.1 Time differences of adaptive feeder requests for service A. | 93 |
| Figure 4.2 Time differences of adaptive feeder requests for service B. | 94 |
| Figure 4.3 End-to-end latencies | 97 |

| | |
|--|-----|
| Figure 4.4 The Z-order on fourth level of quadtree. | 98 |
| Figure 4.5 The LOST-Tree size reductions on spatial aggregations. | 99 |
| Figure 4.6 The LOST-Tree size reductions on temporal aggregations. | 99 |
| Figure 4.7 The LOST-Tree size reductions on both spatial and temporal aggregations. | 100 |
| Figure 4.8 LOST-Tree performance on different L_q : (a) query latencies for different levels of q ; (b) filter efficiencies for different levels of q ; (c) number of requests for different levels of q ; (d) average query latencies; (e) average filter efficiencies; (f) average number of requests. | 104 |
| Figure 4.9 LOST-Tree performance on different L_{gc} : (a) query latencies for different levels of q ; (b) filter efficiencies for different levels of q ; (c) number of requests for different levels of q ; (d) average query latencies; (e) average filter efficiencies; (f) average number of requests. | 106 |
| Figure 4.10 LOST-Tree performance on different combinations of L_q and L_{gc} : (a) query latencies; (b) filter efficiencies; (c) number of requests. | 108 |
| Figure 4.11 The AHS-Model query latency on different number of subscriptions. | 112 |
| Figure 4.12 The indexing latency for (a) point, (b) line, and (c) polygon geometry. | 115 |
| Figure 4.13 The matching latency for (a) point, (b) line, and (c) polygon geometry. | 118 |
| Figure 4.14. The end-to-end query performance for point as subscription and (a) point, (b) line, and (c) polygon geometry as publication. | 121 |
| Figure 4.15. The end-to-end query performance for line as subscription and (a) point, (b) line, and (c) polygon geometry as publication. | 122 |
| Figure 4.16. The end-to-end query performance for polygon as subscription and (a) point, (b) line, and (c) polygon geometry as publication. | 123 |
| Figure 5.1 The basic DSMS architecture. | 133 |

List of Symbols, Abbreviations and Nomenclature

| Symbol | Definition |
|---------------|---|
| 2D | Two-Dimensional |
| 3D | Three-Dimensional |
| 3Vs | Volume, Velocity, and Variety |
| AHS-Model | Aggregated Hierarchical Spatial Model |
| CCTV | Closed-Circuit Television Cameras |
| CEP | Complex Event Processing |
| CPU | Central Processing Unit |
| CSISS | Center for Spatial Information Science and Systems |
| CTO | Chief Technology Officer |
| DBMS | Database Management System |
| DDL | Data Description Language |
| DSMS | Data Stream Management System |
| GeoCENS | Geospatial Cyberainfrastructure for Environmental Sensing |
| GEOSS | Global Earth Observation System of Systems |
| GIN | Groundwater Information Network |
| HTTP | Hypertext Transfer Protocol |
| IEC | International Electrotechnical Commission |
| IETF | Internet Engineering Task Force |
| IOOS | Integrated Ocean Observing System |
| IP | Internet Protocol |
| IT | Information Technology |
| JPL | Jet Propulsion Laboratory |
| JSON | JavaScript Object Notation |
| LOST-Tree | Loading Spatio-Temporal Indexing Tree |
| MBR | Minimum Bounding Rectangle |
| NOAA | National Oceanic and Atmospheric Administration |
| OGC | Open Geospatial Consortium |
| OWS | OGC web service |
| RGB | Red, Green, and Blue |
| RSS | RDF Site Summary |
| SES | Sensor Event Service |
| SOA | Service-Oriented Architectures |
| SOAP | Simple Object Access Protocol |
| SOS | Sensor Observation Service |
| SPS | Sensor Planning Service |
| SQL | Structured Query Language |
| SWE | Sensor Web Enablement |
| URI | Uniform Resource Identifier |
| W3C | WWW Consortium |
| WMS | Web Map Service |
| WNS | Web Notification Service |

WPS
WWW
XML
ZTD

Web Processing Service
World-Wide Web
Extensible Markup Language
Zenith Total Delay

Chapter One: **Introduction**

In recent years, large-scale sensor arrays and the vast datasets produced worldwide are being utilized, shared, and published by a rising number of researchers on an ever-increasing frequency. The global scale ARGOS network of buoys¹, the weather networks of the World Meteorological Organization, and the global GPS Zenith Total Delay (ZTD) observation network are examples of existing large-scale sensor arrays. Moreover, with the advent of the low-cost sensor networks and data loggers, it is technologically and economically feasible for individual scientists to deploy and operate small- to medium-scale sensor arrays. Individual scientists or small research groups can now easily deploy sensor arrays at strategic locations for their own research purposes. There is a spectrum of sensor networks ranging from global-scale permanent observatories to local-scale short-term sensor arrays (Liang et al. 2010).

A significant amount of effort, such as Global Earth Observation System of Systems (GEOSS) and National Oceanic and Atmospheric Administration (NOAA) Integrated Ocean Observing System (IOOS), has been put forth to web-enable these large-scale sensor networks so that the sensors and their data can be accessible through the World-Wide Web (WWW). With open standards defining data schemas and web interfaces, sensors and their data can be integrated in an interoperable manner, which is the main idea of the *World-Wide Sensor Web* (Liang et al. 2005; Botts et al. 2007; Bröring et al. 2011).

The original world-wide sensor web concept was proposed by the NASA/Jet Propulsion Laboratory (JPL) in 1997 (Delin 2005) for acquiring environmental information by integrating

¹ <http://www.argos-system.org>

massive spatially distributed consumer-market sensors. With the development of sensor technology, the sensor web concept has become broader than NASA's original definition and is more related to the concepts of web-enabling sensor networks (Liang et al. 2005). Similar to the WWW, which acts essentially as a "World-Wide Computer", the sensor web can be considered as a "World-Wide Sensor" or a "cyberinfrastructure for sensors". This World-Wide Sensor is capable of monitoring the physical world at spatial and temporal scales that were previously impossible.

To the best of our knowledge about current sensor web development, at the GeoSensorWebLab, we envision that the architecture of sensor web would be very similar to that of the WWW. For example, the WWW connects all of the web services around the world through open standard protocols, such as the Hypertext Transfer Protocol (HTTP). This solution has proven to be very scalable in terms of interchanging messages worldwide. The current sensor web development is moving in a direction that harnesses the power of WWW. We can see this trend from many sensor web projects, such as SensorWare Systems², Microsoft SensorWeb project³, and Xively.com⁴. These projects deploy sensors, collect sensor data, and host and share the data through their proprietary web services.

Similar to the WWW, the sensor web has three major layers, namely, the data layer, the web service layer, and the application layer. Our view of the sensor web layer stack is shown in the Figure 1.1 (Liang and Huang 2013). The data layer can be further divided into the physical

² <http://www.sensorwaresystems.com/>

³ <http://research.microsoft.com/en-us/projects/senseweb/>

⁴ <https://xively.com/>

layer and the sensor layer. While the data layer performs observations⁵ and transmits sensor data to the web service layer, the web service layer provides the application layer with access to the cached sensor data.

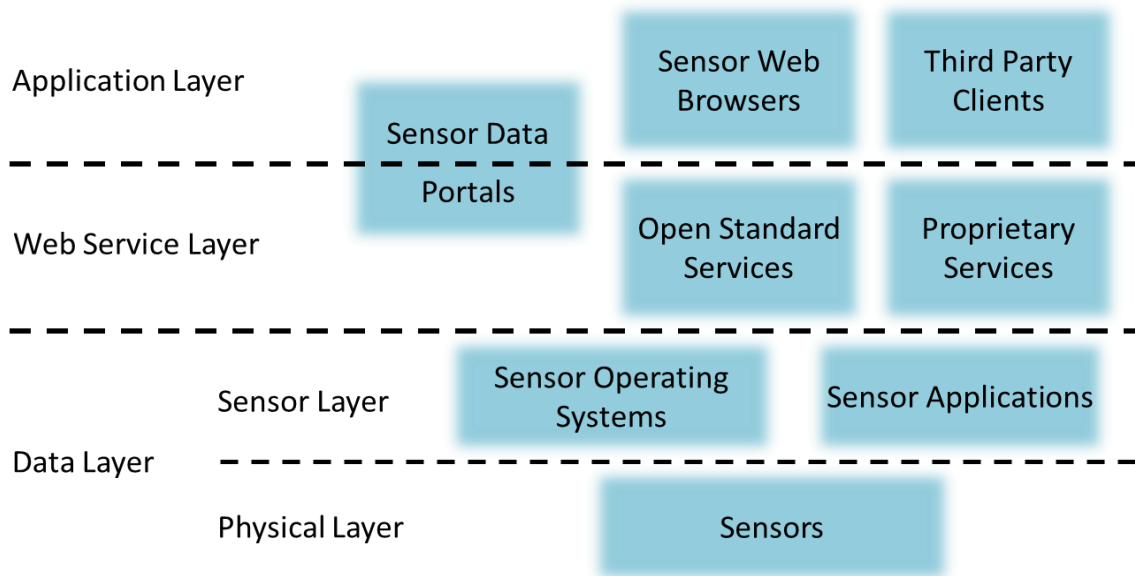


Figure 1.1 The sensor web layer stack.

Since the architecture of a sensor web and WWW are similar, the components that are essential for the WWW should be considered in the development of a sensor web. For example, open standards such as TCP, UDP, IP, HTTP, HTML, etc. are very important in the success of WWW. These open standards (and many others) developed by the Internet Engineering Task Force (IETF), the WWW Consortium (W3C), and the ISO/International Electrotechnical

⁵ Here we follow OGC SWE’s definition of an observation, which is “*an act of observing a property or phenomenon, with the goal of producing an estimate of the value of the property*”.

Commission (IEC) make sure Internet components can communicate with each other in an interoperable manner.

To prevent “reinventing the wheel”, the current sensor web development is built on top of many existing WWW standards. However, as content on the sensor web is fundamentally different from that on the WWW, additional open standards should be defined. The Open Geospatial Consortium (OGC) Sensor Web Enablement (SWE) has defined a suite of open standards for the sensor web (Botts et al. 2007), including specifications for data models, data encodings, and web service interfaces. Although these OGC SWE standards are not as popular as the WWW standards, the development and adaption of sensor web open standards is one of the necessary steps to realize the sensor web vision.

As OGC web services (OWSs) follow W3C’s web service framework, any individual can deploy OWSs on their own machines. The availability of OWS implementations is very important for data owners to easily install and share their data in an interoperable manner. Take OGC Sensor Observation Service (SOS) (Na and Priest 2007) as an example, the public can choose between 52°North SOS⁶, MapServer SOS⁷, Deegree SOS⁸, and Geospatial Cyberinfrastructure for Environmental Sensing (GeoCENS) SOS⁹ implementations to host their sensor data. With these available OWS implementations, we envision that the number of OWSs hosted by data owners will grow rapidly.

⁶ <http://52north.org/communities/sensorweb/sos/>

⁷ http://mapserver.org/ogc/sos_server.html

⁸ <http://wiki.deegree.org/deegreeWiki/deegree3/SensorObservationService>

⁹ <http://wiki.geocens.ca/sos>

Based on these OGC SWE open standards, there are efforts tried to harness the power of sensor web. 52°North¹⁰ has been actively participating in OGC working groups and has developed and open-sourced many OGC SWE client and service implementations. The Center for Spatial Information Science and Systems (CSISS) in the George Mason University continues to participate in the OGC community over the last decade. In addition, Bröring et al. (2011) and Resch et al. (2009) suggested service-oriented architectures (SOA) and workflows that apply pure OGC solutions (including SOS, Sensor Planning Service (SPS) (Simonis and Dibner 2007), Sensor Event Service (SES) (Echterhoff and Everding 2008), Web Processing Service (WPS) (Schut 2007), and Web Notification Service (WNS) (Simonis and Echterhoff 2006), etc.) to enable the discovery, exchange, and processing of sensor observations.

However, these pure OGC approaches still have some issues. For example, Moodley and Simonis (2006), and Bai et al. (2010) utilized a Multi-Agent System approach to address identified service interaction and semantic interoperability issues. Similarly, in GeoSenosrWebLab's GeoCENS project, we not only applied OGC standards but also proposed solutions (e.g., OWS searching engine, sensor web browser, and semantic layer service) to address issues that we identified in the real-world OWS services (Liang and Huang 2013).

In addition to the OGC SWE standards, several works have attempted to propose an architecture for sensor web systems. Intel Research's IrisNet (Gibbons et al. 2003) proposed a decentralized architecture based on a hierarchical topology. IrisNet provides techniques to process queries over distributed XML documents containing sensor data. Microsoft Research's

¹⁰ <http://52north.org/>

SensorMap (Ahmad and Nath 2008) uses a centralized web portal design and tackles the scalability and performance issues with the proposed COLR-Tree. The COLR-Tree is a data structure that indexes, aggregates, and caches sensor streams in order to prevent the transfer of large volumes of sensor streams across the network. The LiveWeb project (Yang et al. 2011) proposes a sensor web service portal to represent and monitor real-time data from sensors and other information providers. LiveWeb uses a content-based publish/subscribe approach (based on keywords, category, and measurement value range) to provide real-time notifications of sensor data or events.

With the development and deployment of sensor web and sensor network technologies, the sensor web generates tremendous volumes of priceless streaming data that enable scientists to observe previously-unobservable phenomena. These technologies are increasingly attracting the interest of researchers for a wide range of applications. These include: large-scale environmental monitoring (Hart and Martinez 2006; Stasch et al. 2012; Auynirundronkool et al. 2012), civil structures (Xu 2002), roadways (Hsieh 2004; Bakillah et al. 2012), and animal habitats (Mainwaring et al. 2002; Chen et al. 2013). These applications utilize sensors ranging from video camera networks that monitor real-time traffic to matchbox-sized wireless sensor networks embedded in the environment to monitor habitats.

Among the applications applying sensor web technology, some of them are time-critical and require efficient data processing for timely decision making and notification, such as emergency response systems (Kassab et al. 2010). These time-critical applications may be used to support decision making when handling time-critical events. Figure 1.2 shows two examples

of time-critical events. Figure 1.2(a) is a photo of the 2007 Minneapolis Interstate-35W bridge collapse, which caused 13 deaths and more than 100 injuries. Figure 1.2(b) shows a photo of the 2011 Japan earthquake and tsunami, which caused 15,883 deaths, 6,145 injuries, and 2,667 people missing. As the world-wide sensor web vision is to connect various types of sensors that are located worldwide and perform observations at high-frequency, the sensor web may have the ability to capture these time-critical events and provide up-to-date information to support decision making. We believe that by efficiently converting sensor web data into information and providing timely notifications, we can lower down or even prevent the damage from these time-critical events.

Therefore, we argue that constructing sensor web infrastructure and applying open interoperable standards are only the first steps. In order to harness the full potential of the sensor web, efficiently processing sensor web data and providing timely notifications are necessary. In this case, not only the emergency response systems can be improved, the sensor web can also benefit any applications that require continuous monitoring and timely response, such as house and human security.

Therefore, from a high-level perspective, this research proposes a software component to efficiently process sensor web data and provide timely notifications. However, in order to achieve this objective, challenges from *the big data phenomenon on the world-wide sensor web* need to be addressed.



(a)



(b)

Figure 1.2 Examples of time-critical events: (a) 2007 Minneapolis Interstate-35W bridge collapse and (b) 2011 Japan earthquake and tsunami.

1.1 Big data phenomenon on the world-wide sensor web

In the context of information technology (IT), the term *big data* is used to describe datasets that are too large or too complex to manage and process with traditional database management systems (DBMS) and data processing applications. The most widely recognized model for big data is *the 3Vs model*, which was first used by Doug Laney in 2001 (Laney 2001). While the 3Vs represent the *volume*, *velocity*, and *variety*, the 3Vs model defines big data as data that are large in volume, velocity, and variety. Each “V” poses different data management challenges. With the rapid development of the sensor web, we have observed that the sensor web also encounters big data challenges. We explain the basic concept of the 3Vs model and how the sensor web fits into this model as follows.

1. *Volume*: One characteristic of big data is the large data volume, which can mean the large size or the large number of data records. While the data volume in social media is known to be enormous (e.g., facebook.com receives more than 500 terabytes of new data every day), Stephen Brobst, the CTO of Teradata, predicted¹¹ in 2010 that *“I don't think social media will be the biggest store of unstructured data for long...Within the next three to five years, I expect to see sensor data hit the crossover point...From there, the former will dominate by factors; not just by 10 to 20 percent, but by 10 to 20 times that of social media”*.

Although the GeoCENS search engine currently only discovers 2,884 Web Map Services (WMSs), which have 88,281 WMS layers, and 36 SOS services, which have 5,310 SOS

¹¹ <http://www.zdnet.com/sensor-data-is-data-analytics-future-goldmine-2062200657/>

observation offerings and 39,368 sensors/procedures (Liang and Huang 2013). It is foreseeable that with the increasing number of sensors being deployed worldwide, the sensor web will be generating more and more sensor data every day. As a result, how to store, transmit, and process sensor web data in large volumes is one of the major challenges.

2. *Velocity*: The velocity characteristic refers to the rate at which data is produced. Unlike human participants using social media, the sensors in the sensor web can produce data at very high frequencies as long as they have power. For example, Boeing jet engines produce 10 terabytes of sensor data every 30 minutes during flight (Rogers 2011). Other examples are closed-circuit television cameras (CCTV) and Internet protocol (IP) cameras for traffic monitoring or surveillance purposes that produce pictures from few to 30 frames per second. As a result, efficiently processing high-velocity sensor data streams is challenging. Especially when considering the geospatial nature of sensor data, how to finish executing time-consuming geospatial operators before new data are measured is another major challenge.
3. *Variety*: Many examples have been used to explain the variety characteristic, including non-aligned data structures, inconsistent data semantics, and incompatible data formats (Laney 2001). In general, the variety characteristic refers to differences between data records. On the sensor web, although sensor data are relatively structured in comparison to social media data, sensor data have large variety in terms of hardware (*i.e.*, different types of sensors), data types (e.g., video, image, text, and number), observed phenomena

(e.g., RGB, air temperature, wind speed), communication protocols (e.g., proprietary protocols), data encodings (e.g., XML, JSON), semantics (e.g., same term interpreted differently), and syntaxes (e.g., same concept described differently). Therefore, how to effectively integrate heterogeneous sensor data and provide a coherent view for users is one of the major challenges for the sensor web.

As big data is a critical issue for data management (Babcock et al. 2002), the big data phenomenon on the sensor web (or *big sensor web data*) also causes issues for processing sensor web data efficiently. In the next section, we further define these issues, introduce existing works, and explain the problems in these works.

1.2 Problems

The traditional *request/response communication model* and DBMS solutions are not suitable for providing timely notifications from big data as they are (1) based on point-to-point *pulling* interaction between users and data providers and (2) not designed for rapid and continuous data streams (Babcock et al. 2002). For instance, in a system following the request/response model (*i.e.*, *pull-based* system), each communication between a user and the system is a complete transaction (*i.e.*, one request and one response), in which the response is evaluated with a snapshot of the system. However, consider a use case requiring timely notifications, no user can predict when an event will happen (e.g., a start of a fire, a collapse of a bridge, or simply an observation). A communication cannot be scheduled ahead of time. By the time a user sends requests and receives responses, events may be already outdated.

Moreover, as a pull-based system may need to answer users' queries at any time, the system needs to store/cache data that happen after a users' last request. Otherwise, users would miss the data if the system does not preserve them. However, as big data are large in volume and velocity, storing a large amount of data in a traditional DBMS is challenging.

There is an alternative communication model known as the *publish/subscribe model* (or *continuous query processing model*), which allows users to register queries in a system and the system executes the queries whenever it receives new data. In this case, the publish/subscribe model can provide notifications more promptly than the request/response model.

The fundamental distinction between the request/response and publish/subscribe model is the *one-time ad-hoc* queries in the request/response model and the *continuous predefined* queries in the publish/subscribe model (Babcock et al. 2002). One-time queries are only evaluated once with a snapshot of dataset, and then answers are returned to users at the moment queries are evaluated. Continuous queries, however, are stored in the system and evaluated whenever new data arrive. The answers of continuous queries are produced over time and sent to users as notifications or streams when new data meet users' query criteria. As a result, users can receive timely notifications since their queries are examined as soon as the system receives new data. Moreover, since data are processed upon arrival, systems can assume that the data will no longer be needed. Therefore, systems following publish/subscribe model (*i.e.*, *push-based* systems) may simply discard processed data or forward them to data warehouses for permanent storage. The high-level workflow of publish/subscribe model is shown in Figure 1.3.

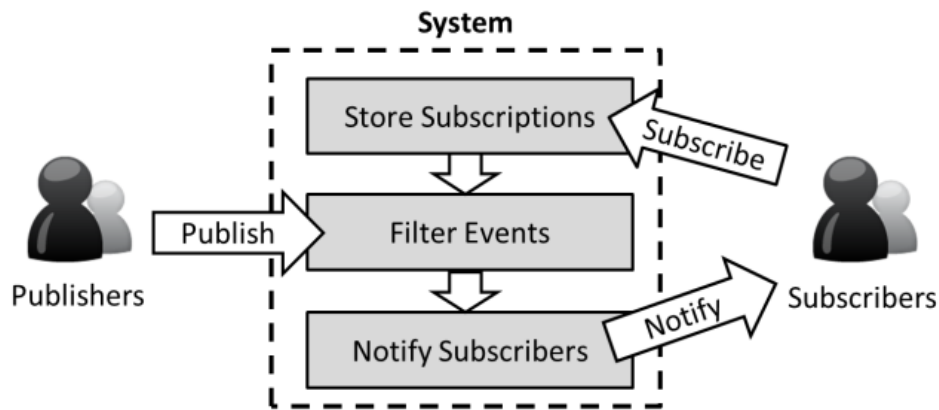


Figure 1.3 The high-level publish/subscribe workflow.

There have been different types of systems applying the publish/subscribe model to manage and process continuous data streams, such as the publish/subscribe system (Eugster et al. 2003), the simple event processing system (Michelson 2006), the data stream management system (DSMS) (Babcock 2002; Golab and Ozsu 2003; Cugola and Margara 2010), and the complex event processing system (CEP) (Luckham 2002). Although the original designs of these systems are different in terms of the targeted data type and query complexity, some of their functionality starts to overlap as they have evolved. More information about these systems and their relationships is presented in Chapter 5.

In the push-based systems, a typical approach to perform a continuous query is to first create a *query execution plan* (Babcock 2002), which consists of *operators* and *queues* (Figure 1.4), and then every new data will traverse through the query plan. There have been mechanisms proposed to optimize query plans. For example, minimizing the number of intermediate results before performing high-overhead operators (Arasu et al. 2004), sharing synopses and operators

across similar query plans (Arasu et al. 2004), and using an incremental evaluation approach to only process new and expired data (Mokbel et al. 2005). However, as most existing mechanisms are designed to optimize the overall structure of query plans, there are few works that discuss how to improve the efficiency of computational-intensive operators.

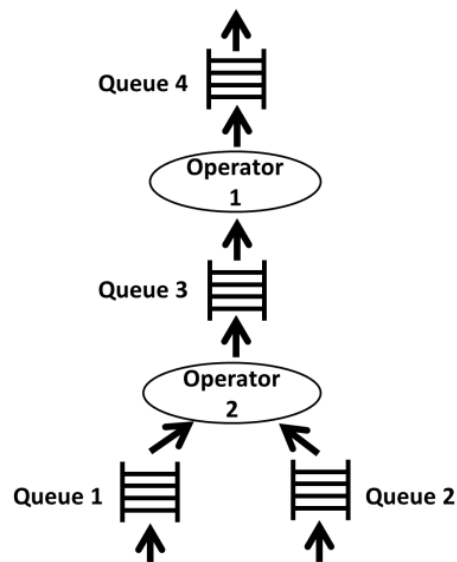


Figure 1.4 An example of query execution plan.

In the context of the world-wide sensor web, while many general-purpose operators are simple and efficient enough to be used directly in query execution plans (e.g., AVERAGE, COUNT, SUM), some geospatial operators are complex and time-consuming. In this research, we take the topological operators as an example. OGC Simple Feature Access specification defines eight geospatial relationships (Herring 2011). Topological operators are operators for determining geospatial relationships between geometries. As the sensor web data are geospatial in nature, supporting topological operators is necessary for a *sensor web publish/subscribe*

system. However, the topological operators are usually recognized as time-consuming tasks and would be a performance bottleneck on processing a large number of geometries (Clementini et al. 1994).

Although there have been some works discussing the topic of supporting geospatial operators in a publish/subscribe system, most of them simply applied spatial database join operations to prove the concept. For instance, Kassab et al. (2010) utilized the ArcGIS Engine .NET SDK to determine topological relationships. Ali et al. (2010) applied the Microsoft SQL Server Spatial Library to support spatial queries in the Microsoft StreamInsight system. Mokbel et al. (2005) encapsulated geospatial algorithms as operators (e.g., INSIDE and k-Nearest-Neighbor operators) in order to support incremental evaluation, reduce intermediate results in the query execution plan, and optimize multiple query plans. However, none of these researches discussed about improving the efficiency of geospatial algorithms for publish/subscribe systems; and we argue that geospatial algorithms can be improved based on the nature of continuous query processing model.

Besides geospatial operators, we have also observed that many of the current sensor data sources only support a request/response communication model (*i.e.*, pull-based data sources), such as OGC SOSs. While most of previous researches assume that data are automatically pushed to the systems, not much focus was put on the retrieval of data streams. As mentioned earlier, no user can predict when a new sensor observation will be available in data sources. In order to retrieve data in a timely manner, users need to frequently send requests to data sources even if many requests are unnecessary (*i.e.*, no new data in the corresponding responses).

Therefore, how to retrieve near-real-time sensor data streams from pull-based sensor web data sources is another critical challenge for a sensor web publish/subscribe system.

1.3 Objectives

The major objective of this research is to propose a comprehensive geospatial sensor web publish/subscribe system, called *GeoPubSubHub*, in order to address the big sensor web data challenges and provide timely notifications. As the scope of GeoPubSubHub is large and each component in GeoPubSubHub has its own unique challenges, this thesis tries to cover both the *breadth* and *depth* of this research by (1) identifying challenges of a geospatial sensor web publish/subscribe system (Chapter 2), (2) proposing an overall system architecture to address identified challenges (Chapter 3.1 and 3.2), and (3) investigating and proposing new solutions for selected critical components, including topological operators and near-real-time sensor web data retrieving (Chapter 3.3, 3.4, 3.5, 3.6 and 3.7).

In summary, this thesis has the following contributions.

1. We identified seven challenges for constructing a sensor web publish/subscribe architecture. While some of these challenges also happen in general-purpose publish/subscribe systems, some of them are unique because of the nature of geospatial sensor web data and data sources.
2. In order to discuss all aspects of a geospatial sensor web publish/subscribe system, we propose solutions with overall system architecture and workflow to address the identified challenges.

3. While some of the challenges have been discussed in existing literature, this thesis focuses on the software components that we believe are most unique and critical in the context of a geospatial sensor web publish/subscribe system. For example, the *sensor web input adaptor* presented in Chapter 3.3 aggregates users' spatio-temporal requests and efficiently retrieves sensor data from data sources while avoiding unnecessary requests. Chapter 3.4 proposes the *LOST-Tree* that can effectively and efficiently aggregate spatio-temporal cubes in order to avoid redundant requests. The *AHS-Model* introduced in Chapter 3.6 is able to efficiently determine topological relationships between users' spatial subscriptions and newly-produced sensor web data in a sensor web publish/subscribe system.
4. To the best of our knowledge, this thesis proposes one of the first geospatial sensor web publish/subscribe system architectures, which could serve as a promising initiative to address the unique big sensor web data challenges and consequently allow us to harvest the full potential of the world-wide sensor web.

The remainder of this thesis is organized as follows. Chapter 2 presents the challenges and existing approaches in general-purpose publish/subscribe architectures, and also identifies the challenges in a sensor web publish/subscribe architecture. In Chapter 3, we present the proposed solutions, system architecture, and detailed algorithms of selected components. This is followed by a performance evaluation of proposed algorithms in Chapter 4. Please kindly note that detailed discussions of related works are presented in Chapter 5. Finally, Chapter 6 includes conclusion and future work.

Chapter Two: **Challenges**

In order to provide an overview of this research, this chapter presents the identified challenges for constructing a geospatial publish/subscribe system for a sensor web context. In the Chapter 2.1, we summarize the challenges and existing approaches in a general-purpose publish/subscribe architecture. And in the Chapter 2.2, we discuss the identified challenges in a sensor web publish/subscribe architecture.

2.1 Challenges and existing approaches in a general-purpose publish/subscribe architecture

In order to design a comprehensive architecture for sensor web publish/subscribe systems, we first review the challenges and existing solutions in general-purpose push-based systems. As mentioned earlier, these systems including publish/subscribe systems, simple event processing systems, DSMSs, and CEP systems follow the same concept of continuous query processing model, *i.e.*, users can register queries and receive notifications as new data match their queries. In order to provide an overview for general-purpose publish/subscribe systems, we summarize and categorize the challenges and existing solutions that are common in these systems.

There have been many papers summarizing these general-purpose push-based systems such as Babcock et al. (2002), Eugster et al. (2003), Golab and Ozsu (2003), Cugola and Margara (2010). Readers interested in further details of this section are referred to these papers. In order to provide a high-level overview of challenges these systems discussed, we observe that the challenges can be categorized into two major issues, namely *memory* and *query efficiency*.

Since these systems aim on handling continuous data streams, which are large in size and generated at a high rate, processing these continuous data streams could easily exhaust available memory (*i.e.*, memory issue) or result in poor query performance (*i.e.*, query efficiency issue). In order to address these two issues, previous works proposed several approaches. Here we categorize these approaches into three types, namely *approximation*, *query optimization*, and *adaptivity approaches*. We further explain these three categories as follows:

1. *Approximation approaches*: Due to limited memory, providing exact answers for data stream queries is not always possible. In this case, a high-quality approximated answer becomes the second choice. Approaches belonging to this category are sliding windows, batch processing, replacing blocking operators by non-blocking operators, load shedding, synopsis construction, and the k-constraint in the STREAM system (Arasu et al. 2004). While the major objective of approximation approaches is to reduce the memory usage, some of these approaches indirectly speed up the query processing.
2. *Query optimization approaches*: A typical approach to perform steam query processing is to first create a query execution plan (Figure 1.4), and then each new data will traverse through the operators and queues in the query plan. There have been some approaches to optimize queries by modifying the structure of query plans. For example, minimizing the number of intermediate results (based on the *selectivity* of operators) before performing high-overhead operators (Arasu et al. 2004), sharing synopses and operators across similar query plans (Arasu et al. 2004), using an incremental evaluation approach to only process new and expired data, and distributing query plans to multiple processing nodes

(Mokbel et al. 2005). These approaches are mainly for improving the query efficiency and would not degrade the quality of answers.

3. *Adaptivity approaches*: The throughput of query plans vary with the dynamic nature of data streams and queries. For example, the arrival rate of data streams, the number of intermediate results, and the available CPU and memory resources all change over time. In order to ensure the query plans are the most efficient for the current system condition, approaches are proposed to optimize the system performance on-the-fly. For example, the Eddies approach (Avnur and Hellerstein 2000) first routes data through query operators to discover fast and selective operators and then dynamically re-organize query plans. The StreaMon module in STREAM employs three components (*i.e.*, Profiler, Reoptimizer, and Executor) to adaptively adjust the k values of k -constraint and the structure of query plans (Arasu et al. 2004).

This review of the challenges and existing solutions in general-purpose publish/subscribe systems provides a foundation for designing GeoPubSubHub. While some of the aforementioned solutions could be directly applied to GeoPubSubHub, GeoPubSubHub is different from other systems in that it is specifically designed to handle sensor web data streams. Since the sensor web data streams and data sources have their own unique characteristics, GeoPubSubHub would face some unique challenges. Therefore, we identify these challenges in the next section.

2.2 Identified challenges in a geospatial sensor web publish/subscribe architecture

In this section, we try to provide a comprehensive understanding about the challenges for constructing a geospatial publish/subscribe system in the sensor web context. While some of the

challenges may be common with other general-purpose systems, others are unique because of the nature of sensor web data and data sources.

As we explained earlier about the sensor web layer stack (Figure 1.1), the data layer performs observations and transmits sensor data to the web service layer, and the web service layer provides the application layer with access to the cached sensor data. Since GeoPubSubHub is an application that filters sensor web data streams with predefined queries, GeoPubSubHub needs to consider the challenges of communicating with data sources in the web service layer. Moreover, as sensor web data is geospatial in nature, GeoPubSubHub would have unique challenges in handling geospatial queries or visualizing geospatial data.

We identify and present the major challenges for building a geospatial sensor web publish/subscribe system in the following list. In general, the first four challenges are related to the data sources, the fifth and sixth challenges are mainly about query processing, and the seventh challenge locates in the client side. In addition, to provide a different view for the challenges, the first and fifth challenges arise from the nature of continuous data streams, which is more similar to the challenges in general-purpose systems, while others are more related to the sensor web.

1. *Large amount of continuous data streams*: Although large amount of data streams allows users to observe events that are previously unobservable, traditional DBMS approaches are not designed for the rapid and continuous arrival characteristic of data streams (Babcock et al. 2002; Golab and Ozsu 2003). This challenge happens in general-purpose publish/subscribe systems as well.

2. *Pull-based data sources*: We have observed that many of the sensor data sources only support a request/response communication model, e.g., OGC SOSs. Users need to proactively send requests in the first place. However, as mentioned earlier, no user can know when new sensor observations will be available in data sources. In order to retrieve near-real-time data from pull-based data sources, a naïve approach is to send requests to data sources very frequently. However, many of these requests would be unnecessary (*i.e.*, no new data in the corresponding responses) and cause extra burden on both clients and servers. Therefore, how to retrieve near-real-time sensor data from pull-based sensor web data sources with an acceptable overhead is one of the critical challenges.
3. *Large number of data sources*: Follow on the previous challenge, if a data source only supports pull-based communication model, at least one Internet connection is needed to frequently retrieve data from a data source. As we argue that the number of sensor web services would grow rapidly, handling a large number of connections becomes an important challenge for a client with limited resource.
4. *Heterogeneous sensor web data*: We observed that the sensor web is highly heterogeneous in terms of communication protocols, data encodings, semantics, syntactic, etc. (Knoechel et al. 2011). Some of these heterogeneities can be addressed by applying open standard protocols, such as communication protocols and data encodings. However, even after applying open standards, there are still some interoperability issues due to the lack of standardized naming, such as semantic heterogeneity and syntactic heterogeneity. As an example for semantic heterogeneity, consider the two strings

“precipitation” and “rainfall”. Since rainfall is a type of precipitation, a user interested in precipitation data would likely be interested in rainfall. Although these concepts are intuitively related to human, to any computer these are simply different sequences of characters.

The syntactic heterogeneity usually comes from the various labels used to represent the same concept, as different data providers may label their data differently. For example, Table 2.1 shows an example of the various URIs used in SOSs to represent the concept of wind speed. This syntactic heterogeneity causes difficulties for a system to integrate all sensor data about wind speed.

As a result, how to integrate heterogeneous sensor web data and provide users with a coherent view of sensor web data is one of the unique challenges for GeoPubSubHub.

Table 2.1 Various URIs of the concept of wind speed.

| | |
|----|---|
| 1 | urn:x-ogc:def:property:OGC::WindSpeed |
| 2 | urn:ogc:def:property:universityofsaskatchewan:ip3:windspeed |
| 3 | urn:ogc:def:phenomenon:OGC:1.0.30:windspeed |
| 4 | urn:ogc:def:phenomenon:OGC:1.0.30:WindSpeeds |
| 5 | urn:ogc:def:phenomenon:OGC:windspeed |
| 6 | urn:ogc:def:property:geocens:geocensv01:windspeed |
| 7 | urn:ogc:def:property:noaa:ndbc:Wind Speed |
| 8 | urn:ogc:def:property:OGC::WindSpeed |
| 9 | urn:ogc:def:property:ucberkeley:odm:Wind Speed Avg MS |
| 10 | urn:ogc:def:property:ucberkeley:odm:Wind Speed Max MS |
| 11 | http://marinemetadata.org/cf#wind speed |
| 12 | http://mmisw.org/ont/cf/parameter/winds |

5. *Large number of queries*: There have been a number of applications applying world-wide sensor web on large-scale monitoring (Xu, 2002). It is foreseeable that numerous domain

specialists or the general public will generate various types of queries to receive timely notifications from the sensor web. Therefore, how to maintain query efficiency while handling a large number of queries becomes one of the major challenges. While this challenge also happens in general-purpose publish/subscribe systems, the concepts of existing solutions would benefit the design of GeoPubSubHub.

6. *Geospatial data and queries*: One of the major differences between GeoPubSubHub and general-purpose publish/subscribe systems is the geospatial nature of sensor web data. While many general-purpose operators are simple and efficient enough to be applied directly in query execution plans, some geospatial operators are complex and time-consuming, such as topological operators (Clementini et al. 1994). While aiming on providing timely notifications, these geospatial operators would become performance bottlenecks. As a result, how to efficiently process geospatial operators in a publish/subscribe system becomes a critical research question for GeoPubSubHub.
7. *Sensor web data visualization*: Due to the geospatial nature of sensor web, the data streams, queries, and notifications are inherently geospatial. How to enable users to visualize sensor web data, create queries, and display notifications in a geospatial manner is another unique challenge for GeoPubSubHub.

To sum up, in this chapter, we identified the challenges for constructing a geospatial publish/subscribe system in the sensor web context. We believe that each of these challenges is an interesting and important research question that is worth further investigated. In this research, we present an overall system architecture to address these challenges (Chapter 3.1 and 3.2).

Then, among the proposed solutions, we selectively choose some to explain in detail (from Chapter 3.3 to 3.7).

Chapter Three: **Methodology**

In this chapter, we present the design of GeoPubSubHub including proposed solutions for addressing the identified challenges and overall system architecture. While some of the proposed solutions are similar to existing solutions, we put our focus on the modules that we believe are most unique and critical in the context of a geospatial sensor web publish/subscribe system. Hence, the proposed solutions, including *sensor web input adaptor*, *LOST-Tree*, *semantic layer service*, *AHS-Model*, and *sensor web browser*, are presented in detail in this chapter.

3.1 Proposed solutions

In order to construct an architecture for efficiently examining sensor data streams and providing timely notifications, we propose solutions to address the seven challenges presented in the Chapter 2.2. While some of the proposed solutions are inspired by existing solutions in general-purpose publish/subscribe systems, some solutions are newly proposed to address the unique challenges from the nature of sensor web data and data sources. To be more specific, the sixth challenge is about processing geospatial data streams and subscriptions; and the second, third, and fourth challenges are even more domain-specific for the sensor web. The proposed solutions are as follows:

1. *Publish/Subscribe communication model*: We designed GeoPubSubHub based on the publish/subscribe communication model for processing continuous data streams. GeoPubSubHub allows users to register continuous queries. And these queries are evaluated whenever new data arrive. The answers from the continuous queries are

produced over time and sent to users as notifications when a new data meets query criteria.

As mentioned earlier, the publish/subscribe model has been used in literatures to process continuous data streams. In this research, we apply the same model on GeoPubSubHub to address the *large amount of continuous data streams* challenge mentioned in Chapter 2.2.

2. *Adaptive sensor stream feeder*: As mentioned in Chapter 2.2, many of the sensor web data sources pull-based. Since no user can know when new sensor observations will be available in data sources, a naïve approach is to send requests to data sources very frequently. However, many of these requests would be unnecessary and cause extra burden on both client and server. Therefore, in order to retrieve data streams from pull-based sensor data sources in a timely manner, GeoPubSubHub develops a new component called *adaptive sensor stream feeder*. The adaptive sensor stream feeder detects the sampling period of each data source, and adaptively adjusts the frequency of retrieving data from sources.

The idea of adaptive sensor stream feeder is based on two assumptions, namely (1) the sampling period of sensor web data is regular, and (2) new sensor data become available in data sources as soon as they are measured (*i.e.*, the time difference between *sampling time* and *valid time* is small). As long as the data updating behaviors of data sources match the assumptions, the adaptive sensor stream feeder can retrieve new sensor data in a timely manner and consequently address the challenge of *pull-based data sources*

mentioned in Chapter 2.2. The detail of adaptive sensor stream feeder is presented in Chapter 3.3.

3. *Semantic layer service*: As mentioned in the Chapter 2.2, existing sensor data sources are heterogeneous in terms of communication protocol, semantic, and syntactic. The protocol heterogeneity could be addressed by supporting different protocols for various types of data sources. However, even within the same type of data source (e.g., OGC SOS), semantic and syntactic heterogeneities still exist. Therefore, in order to provide a coherent conceptual framework for users to query the sensor web, GeoPubSubHub establishes a consultant service called *semantic layer service*.

The semantic layer service retrieves metadata from sensor data sources to classify their data into a predefined *phenomenon taxonomy* (i.e., a taxonomy for terms representing physical properties such as air temperature, pressure, and water level). In this case, users can register subscriptions with the terms in the phenomenon taxonomy, while each term could essentially link to data instances (measuring the same phenomenon) in multiple data sources. As a result, the semantic layer service integrates the heterogeneous sensor web data and provides users a coherent view of sensor web data, which addresses the challenge of *heterogeneous sensor web data* mentioned in Chapter 2.2.

As the semantic layer service is a cooperative contribution, the detailed methodology can be found in Knoechel et al. (2013), a high-level introduction is presented in Chapter 3.5.

4. *Subscription aggregation*: One of the existing solutions for optimizing continuous queries is to aggregate and share operators across multiple query plans (Arasu et al. 2004). That

means if one similar operator is required by multiple queries (*i.e.*, subscriptions), a system will aggregate these queries and only execute the operator once to answer the queries. In this case, compared with the approach of processing these queries independently, the system could reduce memory usage and improve performance.

GeoPubSubHub applies a similar concept on two different modules. One is for retrieving data from data sources and the other one is for processing a large number of subscriptions. As sensor web data is geospatial in nature, sensor web data sources like OGC SOS allow users to retrieve data within spatial and temporal extents (e.g., bounding boxes and time periods, or *spatio-temporal cubes*). This functionality allows GeoPubSubHub to only retrieve data in the spatio-temporal cubes that users are interested in, and reduces the data transmission size.

However, users' spatio-temporal cubes could overlap. If treating each subscription independently, sensor data in the overlapped spatio-temporal cubes may be retrieved repetitively, which would be redundant and cause extra burden on both clients and servers. Therefore, we apply a new indexing structure called *Loading Spatio-Temporal Indexing Tree (LOST-Tree)* as a data loading component to aggregate spatio-temporal cubes from multiple subscriptions and avoid redundant data transmission (Huang et al. 2011). The details of the LOST-Tree are presented in Chapter 3.4.

Similar to existing solutions, GeoPubSubHub also aggregates subscriptions during query processing. Since sensor web data are geospatial in nature, query aggregation in GeoPubSubHub would be different from that in non-geospatial publish/subscribe

systems. There were existing works that group queries based on spatial operators in a DBMS table and are then joined with spatial data (Mokbel et al. 2005). However, we argue that this kind of approach simply uses the spatial database join operations to prove the concept. Therefore, in this research, we propose new geospatial operators that can aggregate spatial queries to address the challenge of *a large number of queries/subscriptions* mentioned in Chapter 2.2.

5. *Geospatial operators*: As mentioned in the previous point, GeoPubSubHub is different from other publish/subscribe systems as it handles geospatial sensor web data and queries. In order to aggregate queries to improve query performance, we propose the *Aggregated Hierarchical Spatial Model (AHS-Model)* to process topological queries based on the nature of continuous query processing.

The AHS-Model uses two key ideas to efficiently determine topological relationships in a publish/subscribe system. First, as the queries are predefined and continuous, AHS-Model *pre-generates* necessary indices for geometries of subscriptions and *re-uses* them every time new data arrives. Second, by indexing geometries of subscriptions with the same indexing structure, we can *aggregate* together the subscription indices. In this case, we can not only save storage space, but also intersect new data with all subscriptions in a single process. As a result, the AHS-Model addresses both the *geospatial data and queries* and *a large number of queries/subscriptions* challenges mentioned in Chapter 2.2. The details of the AHS-Model are presented in Chapter 3.6.

6. *Distributed computing*: As many challenges in publish/subscribe systems are caused by the limited memory and CPU resources, GeoPubSubHub tries to address these challenges not only by improving the algorithms and architectures, but also by applying a different infrastructure. Many of the aforementioned solutions are designed to be suitable for distributed computing, such as feeding data from multiple data sources and executing query operators in parallel. With the advance of distributed computing (Dean and Ghemawat 2008) and cloud computing techniques (e.g., Amazon Elastic Compute Cloud), GeoPubSubHub is designed to scale horizontally to address challenges that remain unsolved, such as the *large number of data sources* mentioned in Chapter 2.2.
7. *Sensor web browser*: As we explained about the *sensor web data visualization* challenge in Chapter 2.2, sensor web data are geospatial in nature and how to visualize and query sensor data in a coherent environment is a unique challenge in GeoPubSubHub. In this research, we propose the *sensor web browser*, a map-based online platform, as a coherent frontend. Details about the sensor web browser are presented in Chapter 3.7.

Other than these proposed solutions, since GeoPubSubHub is similar to other general-purpose publish/subscribe systems, it is also flexible enough to leverage existing solutions from previous works, such as sliding window, incremental evaluation, minimizing intermediate results, etc.

3.2 System architecture and processing steps

In this section, we present the design of overall system architecture and the end-to-end workflow. GeoPubSubHub contains seven modules, namely (1) *query repository*, (2) *input*

adaptor, (3) continuous query engine, (4) output adaptor, (5) sensor data cache, (6) semantic layer service, and (7) sensor web browser. The system architecture and processing steps are shown in Figure 3.1.

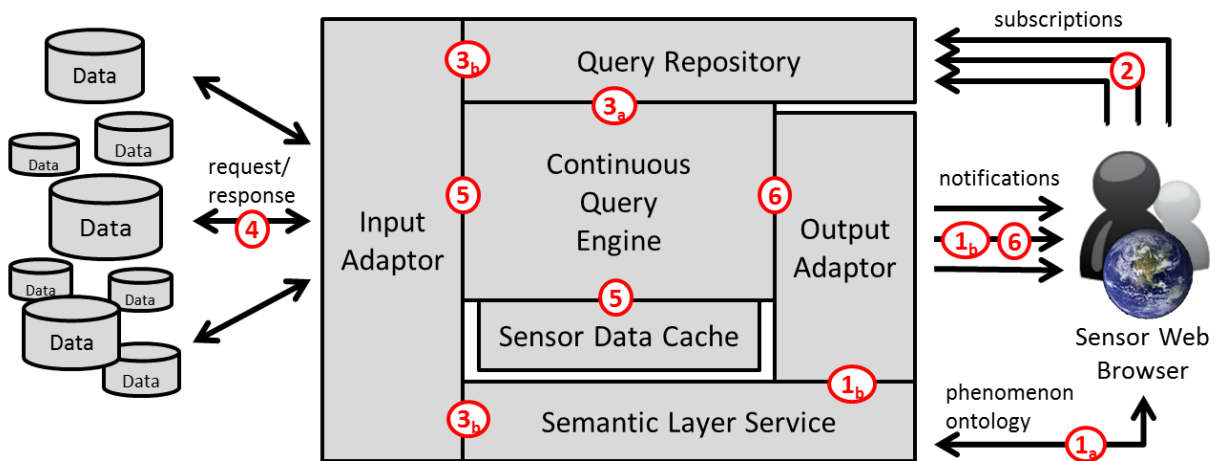


Figure 3.1 High-level GeoPubSubHub system architecture and processing steps.

The system has six major steps starting from sending subscriptions to receiving notifications. Each step is explained as follows:

- 1_a. *Get phenomenon taxonomy*: In order to subscribe to a *sensor web topic* (i.e., a phenomenon), users can get available topics from the phenomenon taxonomy in the semantic layer service and then choose the phenomenon that they are interested in. Users can also get necessary information (e.g., data instance locations) to preview historical sensor data on the sensor web browser.
- 1_b. *Advertise a new sensor web topic or data resource*: Other than requesting for available sensor web topics proactively, GeoPubSubHub can also *advertise* new sensor web topics

or data resources (as Eugster et al. (2003) described). Whenever the semantic layer service receives a new data source or classifies a new phenomenon, the semantic layer service will task the output adaptor to send notifications to users. Users can then preview the data with the sensor web browser.

2. *Register subscription*: A subscription in GeoPubSubHub (*i.e.*, *SUB*) mainly consists of four parts, namely sensor web topic *SUB_{TOPIC}*, spatial predicates *SUB_{SP}*, temporal predicates *SUB_{TP}*, and attribute predicates *SUB_{AP}*. The spatial predicate has two parameters: a base geometry (*i.e.*, point, line, and polygon) and a geospatial operator (which will be further discussed in Chapter 3.6). The temporal predicate could be one of the three different kinds of temporal windows: *fixed window* (*i.e.*, two fixed endpoints in the time axis), *sliding window* (*i.e.*, two sliding endpoints), and *landmark window* (*i.e.*, one fixed endpoint and one sliding endpoint).

The attribute predicate expresses the criterion of measurement value, which could contain four parameters: an aggregation operator (e.g., AVG, SUM, MAX, MIN, and COUNT), a comparison operator (e.g., =, >, <, ≥, ≤ for numerical values, and EQUAL and LIKE for text values), a comparison value, and a unit of measurement. For example, “AVG(value) > 30 km/h” examines whether the average of value is larger than 30 km/h. After using these types of predicates to create subscriptions, users can send subscriptions to the query repository and get a subscription identifier (*i.e.*, *SUB_{ID}*).

3_a. *Prepare continuous query engine*: After the query repository receives subscriptions from users, it forwards the subscriptions to the continuous query engine. The continuous query

engine parses the subscription criteria (*i.e.*, SUB_{TOPIC} , SUB_{SP} , SUB_{TP} , and SUB_{AP}) and prepares query operators, such as pre-generating necessary indices for AHS-Model.

3_b. *Generate requests for data resources*: The query repository also forwards subscriptions to the input adaptor. The input adaptor parses and aggregates criteria (where only SUB_{TOPIC} , SUB_{SP} , and SUB_{TP} are needed here) from subscriptions. The aggregated criteria will be the union of subscriptions for each SUB_{TOPIC} , that is $Aggregation(Topic) = (SUB_{TOPIC}, SUB_{SP_AGG}, SUB_{TP_AGG}) \mid SUB_i \in SUB, SUB_{i_TOPIC} = Topic, SUB_{SP_AGG} = Union(SUB_{i_SP}), SUB_{TP_AGG} = Union(SUB_{i_TP})$, where SUB_{i_TOPIC} , SUB_{i_SP} , and SUB_{i_TP} are the topic, spatial predicate, and temporal predicate of SUB_i . In addition, the *Union* function is performed by LOST-Tree. After aggregation, the input adaptor generates requests to retrieve data in SUB_{SP_AGG} and SUB_{TP_AGG} from data sources that contain data of SUB_{TOPIC} .

4. *Get data from data resources*: After the input adaptor aggregates subscriptions and generates requests, it then sends out requests with the protocols supported by the data sources, such as OGC SOS. When receiving responses from data sources, the input adaptor parses the responses and calculates the sampling periods with the data that have been previously retrieved and stored in sensor data cache. Then in order to achieve the objective of getting data in a timely manner, the input adaptor uses the adaptive sensor stream feeder to adaptively schedule the next retrieving tasks with the calculated sampling periods.

Here we define a sensor observation as PUB , which consists of the sensor identifier PUB_{SID} , the sensor location PUB_{SL} , the phenomenon this observation measures PUB_{PHE} , the geometry this sensor observes PUB_{GEO} , the sampling time PUB_T , the measurement value PUB_{VALUE} , and the unit of measurement PUB_{UNIT} .

5. *Process continuous query plans*: After the input adaptor parses data from responses, it forwards the data to the continuous query engine. The continuous query engine then uses the query operators created in step 3_a to match the new data with subscription criteria. During this process, if operators require calculations of aggregated answers (e.g., AVG, SUM, MAX, MIN, COUNT), old sensor data can be retrieved from the sensor data cache. After the query process, the continuous query engine stores the new data into the sensor data cache and removes the *expired data* (i.e., data that are no longer needed by any subscription).
6. *Send notifications*: During the continuous query processing, if the data meet the criteria of a subscription, the continuous query engine will task the output adaptor to send a notification to the subscriber. The notification contains the SUB_{ID} , the data that meet criteria, and the timestamp of sending out the notification. If users are not online at the time of sending notifications, the notifications will be temporarily stored in the output adaptor. When users are back online, they can use the SUB_{ID} to retrieve those notifications.

From an implementation point of view, the distributed computing technique could benefit all the modules in GeoPubSubHub to address potential scalability issues from the number of data

sources, the number of subscriptions, and the amount of sensor data. With the advance of distributed computing and cloud computing techniques, GeoPubSubHub is designed to scale horizontally to address these scalability issues. For instance, GeoPubSubHub can use multiple machines in the input adaptor module to share the load of data retrieving tasks; and the continuous query engine can distribute query operators to multiple machines then combine the processed results, which is essentially the concept of MapReduce (Dean and Ghemawat 2008).

As the scope of the entire GeoPubSubHub is large, we try to cover all aspects of a geospatial sensor web publish/subscribe system by identifying challenges in Chapter 2, proposing possible solutions to address these challenges in Chapter 3.1, and presenting the design of overall system architecture and workflow in Chapter 3.2. We argue that the challenges identified in Chapter 2.2 are worth further investigated. In this research, we put our focus on the modules that we believe are most unique and critical in the context of a geospatial sensor web publish/subscribe system. The details of these solutions are presented in the following sections, including the sensor web input adapter, LOST-Tree, semantic layer service, AHS-Model, and sensor web browser.

3.3 Sensor web input adaptor

The input adaptors in existing general-purpose publish/subscribe systems are usually used for receiving and parsing streaming data from different types of data sources. The proposed sensor web input adaptor in GeoPubSubHub is different from the existing input adaptors because of the two following reasons. First, as many major sensor web data sources are pull-based, the sensor web input adaptor needs to proactively retrieve sensor data from these data sources in a

timely manner and also try to avoid unnecessary requests. Second, as sensor data are located in the spatio-temporal domain and users' spatio-temporal cubes may overlap with each other, the sensor web input adaptor needs to aggregate these cubes to avoid sensor data being retrieved redundantly.

The proposed sensor web input adaptor has two major components, namely (1) *query aggregator*, and (2) *adaptive sensor stream feeder*. With the queries/subscriptions users submit, the query aggregator first aggregates queries to avoid redundant requests. Then the adaptive feeder tries to get new data with the aggregated queries in a timely manner. After receiving new data from data sources, the new data are forwarded to other modules in GeoPubSubHub and finally stored in the sensor data cache. The workflow and architecture of the sensor web input adaptor is shown in Figure 3.2.

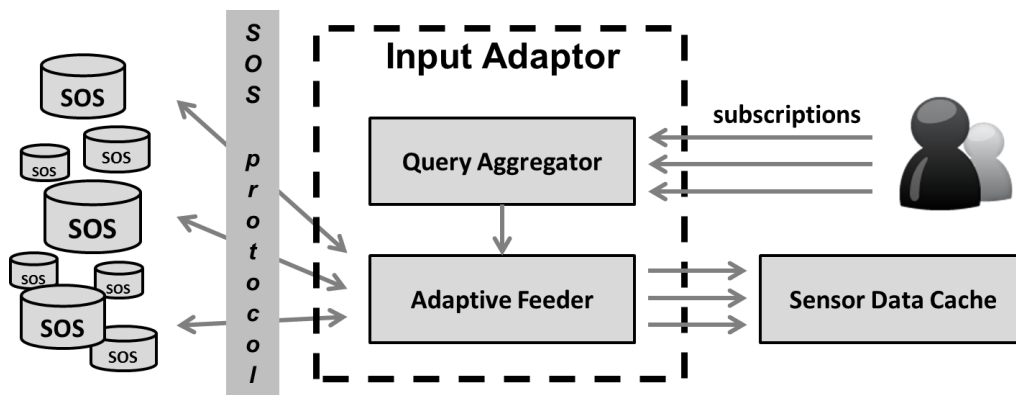


Figure 3.2 System architecture and workflow.

In this research, we use OGC SOS as sensor data sources since it is one of the most popular open standards for hosting sensor data online. In addition, OGC SOS only supports pull-based communication model, which confirms one of the major issues we mentioned earlier. The

details of the query aggregator and the adaptive sensor stream feeder are presented in Chapter 3.3.1 and Chapter 3.3.2, respectively.

3.3.1 Query aggregator

The major objective of the query aggregator is to aggregate sensor data retrieving requests from multiple subscriptions in order to avoid redundant sensor data transmission. Before presenting the details of the query aggregator, we need to introduce the request for sensor web data services. As we are using OGC SOS as the sensor web data sources in this research, a data retrieving request in SOS (*i.e.*, the GetObservation request) mainly contains the service location on the Internet (*i.e.*, service URL), an observation offering ID (*i.e.*, the identifier of a collection of related sensor observations), a observed property URI (*i.e.*, the identifier for the phenomenon), a geographical coverage (*i.e.*, a bounding box), and a temporal coverage (*i.e.*, a time period).

Based on users' subscription SUB (defined in Chapter 3.2), the service URL, observation offering ID, and observed property URI can be obtained from the semantic layer service with the SUB_{TOPIC} . The geographical and temporal coverage (*i.e.*, spatio-temporal cube) are stored in the spatial and temporal predicates (*i.e.*, SUB_{SP} and SUB_{TP}).

Since subscriptions from users could have different but overlapping geographical and temporal coverage, if we treat these subscriptions independently and retrieve data for each subscription, the sensor data in the overlapped spatio-temporal cubes will be transmitted redundantly. These redundant transmissions could cause a large and unnecessary burden on both clients and servers due to the big sensor web data phenomenon.

Therefore, in the query aggregator, we utilize the proposed LOST-Tree (Huang and Liang 2013) to efficiently aggregate spatio-temporal cubes and avoid redundant data transmission. LOST-Tree uses two key ideas to aggregate requests and filters out the loaded portions. First, LOST-Tree applies predefined hierarchical spatial and temporal frameworks, so that both the spatial and temporal extents of requests can be indexed for loading management. Since the frameworks are predefined, LOST-Tree can simply compare spatial and temporal indices between requests to filter out redundant transmission. Also, because the frameworks are hierarchical, LOST-Tree can aggregate several indices to attain a smaller tree size, which consequently results in a smaller memory footprint and query latency. The second idea is that LOST-Tree uses only the spatio-temporal extent of requests to specify the loaded portions. In this case, LOST-Tree does not grow with the sensor data volume, which also allows LOST-Tree to attain a small memory footprint and query latency.

LOST-Tree itself is an independent solution proposed for managing spatio-temporal requests and it does not need to be coupled with the input adaptor. For example, we also apply LOST-Tree in the sensor web browser to manage local cache and sensor data loading requests. Therefore, we present the detail of LOST-Tree in a separate section, Chapter 3.4.

3.3.2 Adaptive sensor stream feeder

After the query aggregator aggregates users' subscriptions, the aggregated requests are forwarded to the adaptive sensor stream feeder. The major problem of retrieving data from a pull-based data source is that we do not know when new data will be available in the service. A naïve solution is to frequently and periodically send requests to services. However, this naïve

solution could generate many unnecessary requests with *empty-hit* response (*i.e.*, no new data in the response).

In order to address this issue, the adaptive feeder attempts to predict when new data will be available in data services. The main idea is to detect the sensor sampling period (*i.e.*, the time difference between observations) and schedule the next request accordingly. Although the sampling time (*i.e.*, the time that the data was measured) and valid time (*i.e.*, the time that the data is available online) could be different in reality, a client can only speculate the valid time from the sampling time as the valid time is usually unknown to the client.

Therefore, the idea of adaptive sensor stream feeder is based on two assumptions, namely (1) the sampling period of sensor web data is regular, and (2) new sensor data are made available in data sources as soon as they are measured (*i.e.*, the time difference between sampling time and valid time is small).

To more formally define the adaptive feeder algorithm, assuming a sensor data instance I has a collection of measurements $I_{Measurements}$ in a descending order of time. And to determine the time of sending next request, we need to first calculate the maximum sampling period SP by calculating the time difference between each successive pair of measurements, that is $SP(I) = \text{Max}(\{ \text{Time}(M_n) - \text{Time}(M_{n+1}) \mid M_i \in I_{Measurements}, 0 < n \leq |I_{Measurements}| \})$, where the M_i represents the i^{th} measurement in $I_{Measurements}$, the Max function calculates the maximum value in a set, and the Time function outputs the sampling time of a measurement. Then we predict that a new measurement of a sensor data instance I will happen at $SP(I)$ after the last measurement, which is $SP(I) + \text{Time}(M_0)$. Finally, in order to accommodate possible delays (e.g., network

delays), a small buffer time $Buffer$ is added to the predicted time. Therefore, $SP(I)+Time(M_0)+Buffer$ will be the time that the adaptive feeder sends request to retrieve the predicted measurement. We also provide a figure to express the algorithm (Figure 3.3).

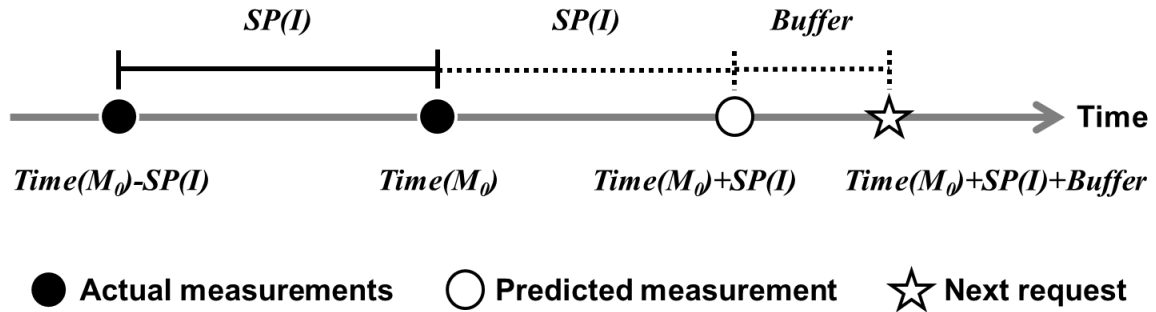


Figure 3.3 An example of adaptive sensor stream feeder algorithm.

To further clarify the definition of a sensor data instance, a sensor data instance can be a sensor or a collection of sensor depending on the type of data service and the functionality supported by the data service. In principle, the adaptive feeder algorithm is ideal for predicting new measurements of each individual sensor as different sensors may have different sampling periods. However, that also means one Internet connection would be required for each sensor, which is impractical considering the big sensor web data phenomenon. Therefore, in order to prove the concept, this research assumes that sensors in the same SOS observation offering and observed property update new measurements at similar time. Thus, we treat each observed property (which contains a collection of sensors) as a sensor data instance in the proposed adaptive sensor stream feeder.

The adaptive feeder will be able to retrieve the data in a timely manner when the aforementioned assumptions are close to reality. Even in the cases that the valid time is very different from predictions, the adaptive feeder can still retrieve data no later than the sampling period after the data becomes available in services. The evaluation results of the proposed adaptive sensor stream feeder are presented in Chapter 4.1.

3.4 LOST-Tree

As mentioned earlier, LOST-Tree is a new solution proposed for managing spatio-temporal requests. In GeoPubSubHub, we apply LOST-Tree in both the input adaptor and the sensor web browser to avoid redundant sensor data transmission. In the input adaptor, LOST-Tree is used to aggregate the spatio-temporal cubes from users' subscriptions. In the sensor web browser, we apply LOST-Tree to manage sensor data loading from local cache and data services.

As LOST-Tree was originally proposed for a sensor web browser use case, this section is written from this point of view. In Chapter 3.4.1, we introduce the background, define the sensor web browser, and explain the needs of LOST-Tree. Chapter 3.4.2 presents the detail algorithms of LOST-Tree including four processing steps and one operation. Finally, we sum up in Chapter 3.4.3.

3.4.1 Introduction

In the same way that the WWW needs a web browser to load and display web pages from web servers, the world-wide sensor web needs a coherent front end to access distributed and heterogeneous sensor networks. However, sensor data are geospatial in nature, and the number of

sensor observations can be extremely large. Efficiently transmitting large amounts of sensor data over the WWW is known to be a major challenge (Nath et al. 2006).

There have been some server-side optimization approaches that provide mashups of base maps and sensor locations. Since this type of application serves as an intermediary between users and the sensor data they host, we term them *sensor data portals*. Figure 3.4 shows screen captures of some existing sensor data portals.

Since these portals have full knowledge about the data they host (e.g., sensor locations and sampling times), they can pre-generate indices or utilize spatio-temporal distributions of sensor data to optimize Internet transmission. For example, Ahmad and Nath (2008) proposed COLR-Tree to aggregate and sample sensor data to reduce data size before transmission. Some sensor data portals, such as the Groundwater Information Network (GIN)¹², present a map of sensor locations at small scale and actual sensor observations at large scale to limit the number of sensor observations being transmitted in each request. However, a critical drawback of these sensor data portals is that they can only present the data for which they have prior knowledge. As a result, these portals are data-specific static maps of sensors.

We argue that these server-side optimization approaches are very difficult to scale up, considering that it is very challenging for a single portal to index every sensor in the world. Therefore, instead of a server-side approach, we propose a pure client-side approach to load sensor data efficiently without prior knowledge of sensor data. As a result, our approach enables a client-side application to load sensor data independently; and we have termed this kind of

¹² <http://analysis.gw-info.net/gin/public.aspx>

application *sensor web browser*. One of the major use cases is that users can use a sensor web browser to access any sensor web services (e.g., OGC SOSs) that are technically interoperable with the sensor web browser.

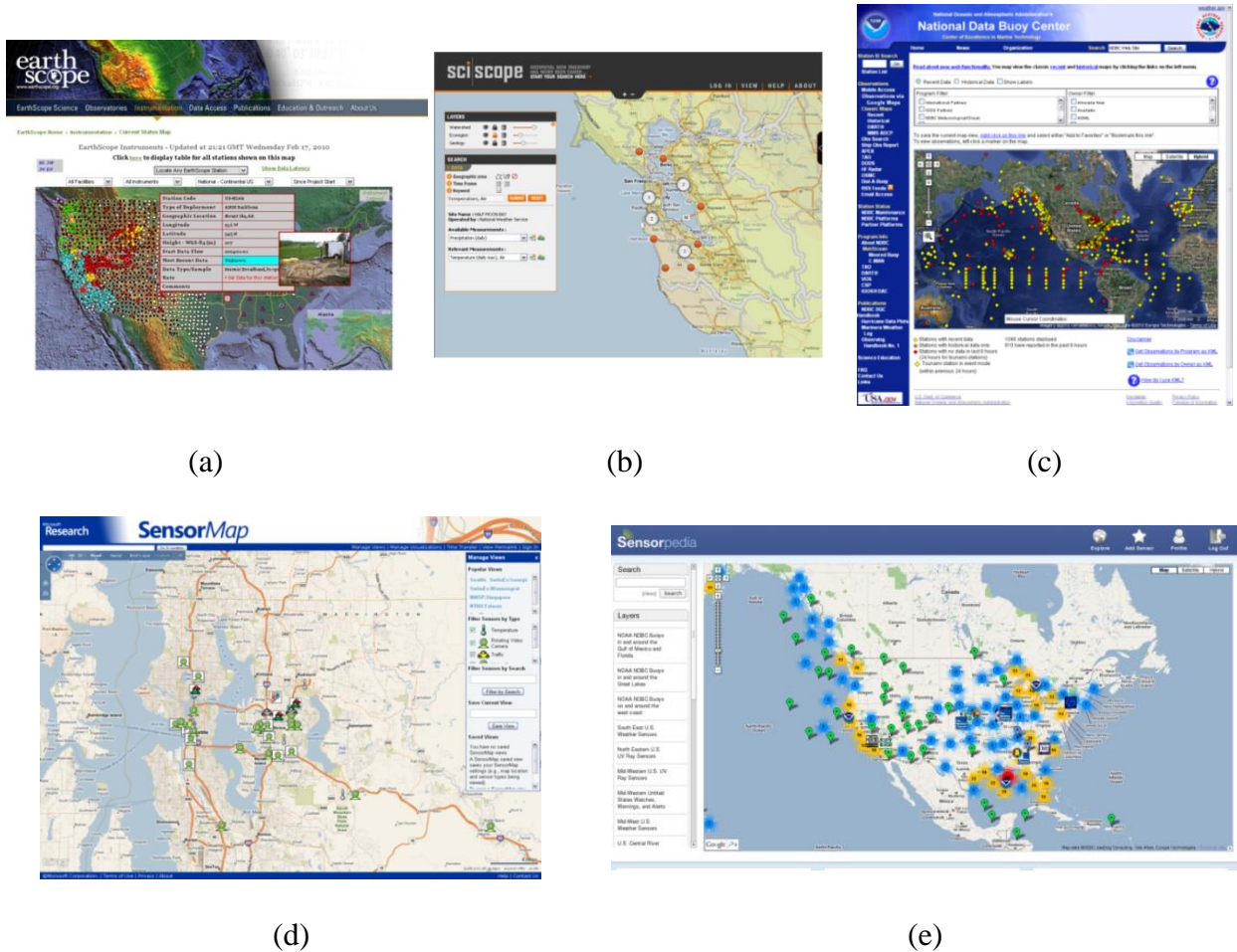


Figure 3.4 Existing sensor data portals: (a) EarthScope (<http://www.earthscope.org/>); (b) SciScope (<http://www.sciscope.org/>); (c) National Data Buoys Center (<http://www.ndbc.noaa.gov/>); (d) SensorMap (<http://atom.research.microsoft.com/sensewebv3/sensormap/>); (e) Sensorpedia (<http://www.sensorpedia.com/>).

However, unlike sensor data portals, sensor web browsers only have knowledge about the sensor data they have already retrieved but not the “empty space” between the sensor data in the spatio-temporal domain. A sensor web browser that only records the retrieved data cannot recognize whether the “empty space” between data is truly empty (*i.e.*, the space was loaded before but has no data) or not-yet-loaded (*i.e.*, the space may or may not contain data). The only way for that sensor web browser to make sure is to send request asking for data in the “empty space”, which may consequently become an endless loop for space that is truly empty and generate redundant transmissions between clients and servers.

Unlike server-side approaches that reduce data size in transmission, our approach applies a client-side cache to avoid redundant transmissions and improve Internet bandwidth utilization. For example, today’s earth browser systems (Craglia et al. 2008) (e.g., Google Earth) use a quadtree-based tiling scheme to index and manage the cached image tiles at different levels of detail. Before sending requests to servers, these systems check the local cache first. In the case of a *cache hit*, no request needs to be sent. Otherwise, in the case of a *cache miss*, requests are sent to servers; and the returned image tiles are then inserted into the local cache for future use.

However, the same tiling and caching method cannot be directly applied to a sensor web browser for two major reasons. First, sensor data is spatio-temporal in nature; compared to static map images, as such there is an additional temporal dimension to be considered. Second, sensor data may be distributed sparsely in space and even more sparsely in time (e.g., transient sensors or sensors with different sampling frequencies). Spatio-temporal requests may receive responses without any sensor data (*i.e.*, empty-hits). In order to prevent redundant empty-hits, not only the

responses (*i.e.*, sensor data) need to be stored and managed in a cache as a *data management component*, the requests also need to be stored and managed in a separate cache as a *data loading component*.

The spatio-temporal data management component has long been investigated to efficiently query spatio-temporal data in databases (Mokbel et al. 2003). However, there are only a few studies that focus on the data loading component for spatio-temporal requests. We argue the major reason is that an independent client-side application retrieving spatio-temporal data was not common. Most client-side applications were highly coupled with the server-side, such as sensor data portals. However, as we are witnessing a technological shift from highly coupled and proprietary web applications to interoperable web APIs, an independent and interoperable client-side application like a sensor web browser becomes necessary to access spatio-temporal data. As a result, this work focuses on developing a data loading component for a sensor web browser in order to avoid unnecessary transmissions and consequently attain efficient sensor data loading with a local cache.

This research proposes LOST-Tree, which stands for loading spatio-temporal tree. LOST-Tree manages sensor web browser requests and acts as a data loading layer between a sensor web browser and servers. LOST-Tree uses two key ideas to solve the aforementioned challenges. First, LOST-Tree applies predefined hierarchical spatial and temporal frameworks, so that both the spatial and temporal extents of requests can be indexed for loading management. Since the frameworks are predefined, LOST-Tree can simply compare spatial and temporal indices between requests to filter out redundant transmission. Also, because the frameworks are

hierarchical, LOST-Tree can aggregate several indices to attain a smaller tree size, which consequently results in a smaller memory footprint and query latency.

The second idea is that LOST-Tree only uses the spatio-temporal extent of requests to determine a cache hit or miss. By separating the data loading and data management components, any data management method can be applied while LOST-Tree handles the data loading. For instance, this work simply uses R-Tree (Guttman 1984) and B-Tree (Bayer, 1972) to manage sensor data in the local cache. Moreover, since LOST-Tree only manages the spatio-temporal extents of requests, LOST-Tree does not grow with the spatio-temporal density of sensor data, which also allows LOST-Tree to have a small memory footprint and query latency.

Besides the GeoPubSubHub, LOST-Tree has been utilized as a data loading component in the sensor web browser of the GeoCENS project, which has been publicly available¹³ since 2010.

3.4.2 Methodology

LOST-Tree manages spatio-temporal requests in a sensor web browser. A typical request, R , from a sensor web browser to a sensor web server can be defined by three parameters: R_{bbox} , which is the minimum bounding box of the request's spatial extent; R_{t_period} , which is the request's temporal extent defined by a start time, t_1 , and an end time, t_2 ; and finally R_{obs} , which is the observed phenomenon of interest (e.g., air temperature). In fact, we can further define the combination of R_{bbox} and R_{t_period} as a spatio-temporal cube, R_{STCube} . Thus, request R and its

¹³ GeoCENS sensor web browser (desktop client): <http://dev.geocens.ca/gswclient>

corresponding server response can be defined as follows: $R (R_{STCube}, R_{obs}): \{o_1, o_2, \dots, o_i\}$, where o_i is an observation collected by a sensor that fulfills request R .

LOST-Tree is a data loading layer between a sensor web browser and servers. The major objective is to prevent sending unnecessary requests to sensor web servers. When a sensor web browser sends request R through LOST-Tree, LOST-Tree has four steps: (1) decompose, (2) filter, (3) update, and (4) aggregate (Figure 3.5).

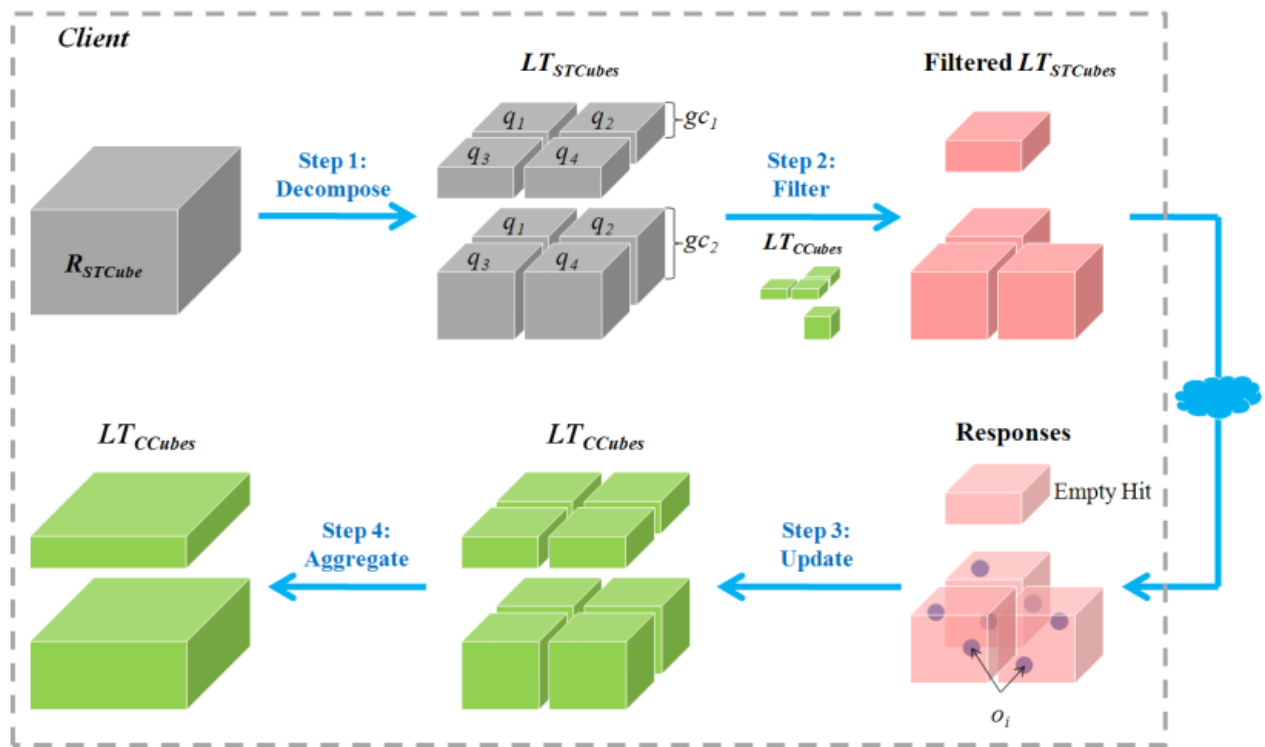


Figure 3.5 LOST-Tree workflow.

3.4.2.1 Decompose step

The purpose of the decompose step is the conversion of an ad-hoc R_{STCube} into one or many non-overlapping LOST-Tree-based requests: $LT_{STCubes}$ (i.e., LOST-Tree spatio-temporal

cubes). The decomposition is based on two predefined hierarchical spatial and temporal frameworks. In this research, we implemented LOST-Tree with a quadtree-based tile system (Figure 3.6) (Gaede and Gunther 1998) as the spatial framework and the Gregorian calendar as the temporal framework.

We used an LT_{STCube} key, which is the combination of a quadkey q , and a calendar string gc (e.g., in formats of YYYY, YYYYMM, YYYYMMDD or YYYYMMDDHHMMSS), to represent an LT_{STCube} , where q represents a bounding box and gc represents a time period (e.g., gc ‘20100930’ represents the entire day on September 30th 2010). One important characteristic in both q and gc is that they are hierarchical in nature, which means that the lengths of q and gc represent their levels of detail. This also allows us to simply apply a prefix matching method to identify whether $LT_{STCube_A} \subseteq LT_{STCube_B}$. For example, given $LT_{STCube_A} (q_A, gc_A)$ and $LT_{STCube_B} (q_B, gc_B)$, $LT_{STCube_A} \subseteq LT_{STCube_B}$ if and only if q_A starts with q_B and gc_A starts with gc_B . As a result, we can easily manage $LT_{STCubes}$ by manipulating the LT_{STCube} keys.

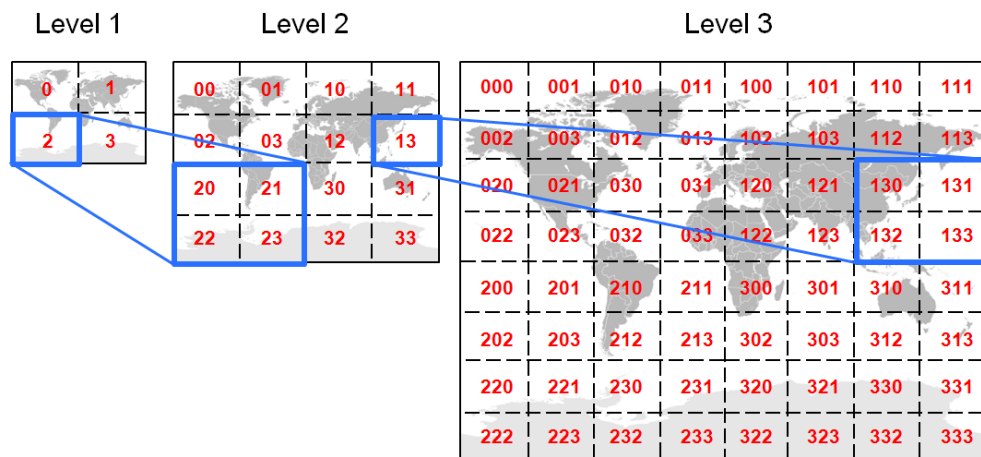


Figure 3.6. Quadtree-based tile system.

3.4.2.2 Filter step

The objective of the filter step is to filter out the requests that corresponding responses have previously loaded. We use LT_{CCubes} (*i.e.*, LOST-Tree cached cubes) to represent loaded $LT_{STCubes}$, and Algorithm 3.1 describes the filtering process. The determination of the containing relationship between $LT_{STCubes}$ and LT_{CCubes} (lines 5 and 8) is done by prefix matching of the LT_{STCube} keys (*i.e.*, q and gc). In line 9 of Algorithm 3.1, if LT_{STCube} covers LT_{CCube} , LT_{STCube} “decomposes” itself and removes the portion covered by LT_{CCube} . For instance, if q of LT_{STCube} is ‘01’ and q of LT_{CCube} is ‘0110’ (assuming LT_{STCube} and LT_{CCube} have the same gc), the $FilteredLT_{STCubes}$ have q equal to ‘010’, ‘0111’, ‘0112’, ‘0113’, ‘012’, and ‘013’. In this way, LOST-Tree can filter out redundant requests.

Algorithm 3.1 The filter step.

Function $Filter(LT_{STCubes}, LT_{CCubes}): FilteredLT_{STCubes}$

```

1:  $FilteredLT_{STCubes} \leftarrow \{\}$ 
2: FOREACH  $LT_{STCube} \in LT_{STCubes}$ 
3:    $previously\_loaded \leftarrow false$ 
4:   FOREACH  $LT_{CCube} \in LT_{CCubes}$ 
5:     IF  $LT_{STCube}$  is contained by  $LT_{CCube}$  THEN
6:        $previously\_loaded \leftarrow true$ 
7:       BREAK
8:     ELSE IF  $LT_{STCube}$  contains  $LT_{CCube}$  THEN
9:        $LT_{STCube} \leftarrow LT_{STCube} - LT_{CCubes}$ 
10:    END IF
11:  END FOREACH
12:  IF NOT  $previously\_loaded$  THEN
13:     $FilteredLT_{STCubes} \leftarrow FilteredLT_{STCubes} \cup LT_{STCube}$ 
14:  END IF
15: END FOREACH
16: RETURN  $FilteredLT_{STCubes}$ 

```

In reality, however, there is a trade-off between reducing redundant transmissions and keeping the number of requests small. This trade-off comes from the limitation of communication protocol or data service implementation. Take OGC SOS as an example, a GetObservation request can only have one bounding box as the spatial extent, so that the number of requests is the number of unique q in $FilteredLT_{STCubes}$ (assuming that these quadkeys are not connected). For instance, the example in Figure 3.5 will need three requests, if q_1 , q_3 , and q_4 are not connected. Although OGC SOS specification allows multiple time periods as the temporal extent in one request, some SOS implementations only support one time period per request. For these SOS implementations, the number of requests becomes the number of unconnected spatio-temporal cubes, e.g., the example in Figure 3.5 will need four requests assuming those $FilteredLT_{STCubes}$ are not connected to each other and cannot be aggregated. As a result, filtering out redundant transmissions may generate a large number of requests, which would consequently reduce loading efficiency and causes issues in handling a large amount of connections.

In order to address this issue, we provide a mechanism for LOST-Tree to control the trade-off between the number of requests and redundant transmissions. As mentioned, in order to filter out LT_{CCube} from LT_{STCube} , LT_{STCube} is decomposed to the same q and gc levels of LT_{CCube} . Usually, the larger difference between the level of LT_{STCube} and the level of LT_{CCube} , the more $FilteredLT_{STCubes}$ are generated, which consequently increases the number of requests.

Therefore, we allow users to configure two variables, L_q and L_{gc} , to specify the lowest q and gc levels to which LT_{STCube} can be decomposed. For example, if L_q is 5, the lowest q level of $FilteredLT_{STCubes}$ is equal to or smaller than 5. If L_{gc} is an hour, the lowest gc level of

$FilteredLT_{STCubes}$ is equal to or larger than hour (*i.e.*, year, month, day, and hour). Even if the levels of q and gc in LT_{CCube} are larger than L_q and L_{gc} , LOST-Tree stops the decomposition after reaches the L_q and L_{gc} . Although this approach may result in some redundant transmissions (*i.e.*, LT_{CCube} whose levels of q and gc are larger than L_q and L_{gc}), the number of requests can be dramatically decreased. We show how this mechanism affects LOST-Tree performance in the evaluation section.

3.4.2.3 Update step

This step is for updating LT_{CCubes} to reflect the request history. Once the client receives responses from servers, LOST-Tree inserts the corresponding $LT_{STCubes}$ into LT_{CCubes} . This operation is very important in the context of a sensor web. Since sensor observations may distribute sparsely in space and time, server responses may not contain any sensor data (*i.e.*, empty-hits). While only caching responses (like today's earth browsers) cannot avoid empty-hit requests, LOST-Tree is unique as it can avoid these redundant transmissions by remembering the spatio-temporal extents of successful requests (whether they are empty-hits or not). Therefore, LOST-Tree inserts the corresponding $LT_{STCubes}$ into LT_{CCubes} for the complete history of requests (*i.e.*, $LT_{STCubes}$) to avoid redundant requests.

3.4.2.4 Aggregate step

This step is for minimizing the memory footprint of LT_{CCubes} with the hierarchical characteristic of the spatial and temporal frameworks. When all $sub-LT_{CCubes}$ (e.g., the eight small green cubes in Figure 3.5) of an LT_{CCube} (e.g., the two large green cubes in Figure 3.5) are loaded, we can replace all the $sub-LT_{CCubes}$ with just one LT_{CCube} . Therefore, after the update step

inserting the loaded $LT_{STCubes}$ into LT_{CCubes} , LOST-Tree first identifies those “aggregatable” LT_{CCubes} and aggregates them into one LT_{CCube} . For example, quadkeys ‘00’, ‘01’, ‘02’, and ‘03’ can be replaced by ‘0’.

In this case, LOST-Tree can have a smaller number of LT_{CCubes} , which consequently allows better query performance in the filtering step. In addition, since LOST-Tree only maintains a quadkey (*i.e.*, q) and calendar string (*i.e.*, gc) for each LT_{CCube} , the tree size is small enough to fit into memory for efficient processing.

3.4.2.5 Removal operation

In addition to the above four steps, LOST-Tree also has an operation to remove LT_{CCubes} . Some sensor data providers need time to collect and calibrate data for quality assurance. In this case, data would be outdated when they become available online. However, before these historical data become available online, clients who request for $LT_{STCubes}$ covering these data may get empty-hit responses and mark these $LT_{STCubes}$ as loaded in LOST-Tree. In order to reload these $LT_{STCubes}$, clients need an operation to remove spatio-temporal cubes from LT_{CCubes} .

The removal operation is shown in Algorithm 3.2, where LT_{RCubes} represents the spatio-temporal cubes to be removed. Similar to line 9 in Algorithm 3.1, the “ $(LT_{CCube} - LT_{RCubes})$ ” in line 5 of Algorithm 3.2 means that the LT_{CCube} decomposes itself and removes the portion covered by LT_{RCube} . Although Algorithms 3.1 and 3.2 are similar, the meanings of their outputs are different. Algorithm 3.1 returns the spatio-temporal cubes to be requested, while Algorithm 3.2 returns the spatio-temporal cubes that have been loaded.

Algorithm 3.2. The remove function.

Function $Remove(LT_{RCubes}, LT_{CCubes}): LT_{CCubes}$

```
1:  FOREACH  $LT_{RCube} \in LT_{RCubes}$ 
2:    FOREACH  $LT_{CCube} \in LT_{CCubes}$ 
3:      IF  $LT_{RCube}$  is contained by  $LT_{CCube}$  THEN
4:         $LT_{CCubes} \leftarrow LT_{CCubes} - LT_{CCube}$ 
5:         $LT_{CCubes} \leftarrow LT_{CCubes} + (LT_{CCube} - LT_{RCube})$ 
6:        BREAK
7:      ELSE IF  $LT_{RCube}$  contains  $LT_{CCube}$  THEN
8:         $LT_{CCubes} \leftarrow LT_{CCubes} - LT_{CCube}$ 
9:      END IF
10:    END FOREACH
11:  END FOREACH
12:  RETURN  $LT_{CCubes}$ 
```

3.4.3 Contribution summary

We evaluated LOST-Tree implementation with a real OGC SOS service. Our evaluation results demonstrated that, with LOST-Tree and the local cache, we can attain sensor data loading of at least 100 times faster, up to a 100% reduction of unnecessary transmissions, a small tree size (less than 164 Kbytes during our evaluation), and a small latency when determining a cache hit/miss. The detail of this evaluation is presented in Chapter 4.2. In summary, the proposed LOST-Tree makes the following contributions.

1. We present LOST-Tree, a data loading component that determines whether or not a spatio-temporal request has been sent previously. LOST-Tree can significantly improve Internet bandwidth usage by filtering out redundant requests, and enable a client-side application to load sensor data efficiently.
2. LOST-Tree applies predefined hierarchical spatial and temporal frameworks to index and manage spatio-temporal requests. We demonstrate how to determine a cache hit/miss

with spatial and temporal indices and how to aggregate them for storage and computation efficiency.

3. To address the issue of empty-hits, we manage spatio-temporal requests in LOST-Tree. By decoupling data loading and management, we show that LOST-Tree is scalable in terms of the sensor data volume. This also allows LOST-Tree to work with any data management method.

3.5 Semantic layer service

As mentioned earlier, sensor web data have semantic and syntactic heterogeneities even in the same ecosystem such as the OGC SOS. The semantic layer service is proposed to integrate the heterogeneous sensor web data and provides users with a coherent view of the sensor web data. As the semantic layer service is a cooperative contribution, a detailed methodology can be found in Knoechel et al. (2013), and this section provides a high-level introduction about the semantic layer service.

In this research, we focus on the OGC SOS as a sensor web data source. Here we follow the terminology defined in OGC SOS. An *observation* is the “act of measuring a real world phenomenon”. A *phenomenon* is a “characteristic of one or more feature types, such as wind speed”. An *observed property* uses a unique identifier (*i.e.*, URI) to represent the phenomenon. An *observation offering* is a “logical grouping of observations that are similar in some way”. The GetObservation operation, one of three core operations in SOS, is used to get observation data from SOS services. In a GetObservation request, one observation offering and at least one observed property are required to specify observation data measuring one phenomenon. Here we

define the combination of a service location (*i.e.*, service URL), an observation offering, and an observed property as a *property layer*. Besides the aforementioned mandatory parameters, users can optionally assign a bounding box and time period(s) to specify the observation data in the spatio-temporal cube(s) that they are interested in.

Although SOS achieves the goal of sharing sensor data online, we observed that the real-world SOS services are heterogeneous because of two major issues, namely (1) a large variety of observed property URIs used to represent the same phenomenon, and (2) missing relationships between observed properties. To give an example for the first issue, Table 2.1 shows the URIs for wind speed in real-world SOS services. The large variety of observed property URIs causes challenges for phenomenon-based searches. For example, a scientist looking for wind speed data would need to know all observed property URIs for wind speed used in every service to collect all relevant data. The URIs in Table 2.1 show how the same semantic concept “wind speed” is encoded with different syntactic schemes. This raises the issue of syntactic differences between URIs.

For the second issue, we observed problems when semantic information between observed properties is missing. An example would be the relationship between “precipitation” and “rainfall”. A user searching for precipitation would be interested in property layers with rainfall data. However, since observed property URIs are defined by data providers, there is no semantic relationships between observed property URIs without following a commonly-agreed phenomenon taxonomy. Although there has been effort to propose ontologies that can be used to

address this issue such as the NASA’s SWEET ontology¹⁴, it is the data providers’ responsibility to follow these ontologies when sharing their data. Therefore, we argue that this kind of *top-down approach* is not a viable solution as data providers may not be aware of these ontologies or they do not want to expend the effort.

Therefore, we propose the semantic layer service using a *bottom-up approach* to address these two issues. The approach is to provide a well-defined taxonomy of phenomenon, where each element maps to a list of property layers that contain sensor data of this phenomenon. In this prove-of-concept implementation, a data processing team in the GeoSensorWebLab first creates phenomenon elements and organises them in a hierarchical structure. Then the next step is to map each property layer to the phenomenon element. The idea is to first extract the information about phenomenon from the observed property URIs as they represent the phenomenon each property layer measures, and then compare the information with the phenomenon elements.

To be more specific, we first perform normalization and tokenization text processing on the observed property URI of each property layer. Normalization is the process of canonicalizing strings such that superficial differences between strings are removed (e.g., “windspeeds” becomes “windspeed”). Tokenization is the process of converting text into distinct tokens (e.g., “windspeed” becomes “wind” and “speed”).

Then we map property layers with phenomenon elements by calculating the similarity between the processed observed property URIs with the names of phenomenon elements. Three

¹⁴ <http://sweet.jpl.nasa.gov/ontology/>

similarity functions are used and compared for this work, a length-adjusted Levenshtein similarity function (which is a modification of the Levenshtein distance (Levenshtein 1966)), the Jaccard similarity function, and a semantic similarity function. The semantic similarity function is based on the Jaccard similarity function and uses WordNet¹⁵ as a lexical database to calculate word pair semantic similarity scores. The calculated similarity between two objects is a numerical measure of the degree to which the two objects are alike. By ranking the similarities, we can map a property layer to the most similar phenomenon element. Figure 3.7 illustrates the high-level architecture of the semantic layer service.

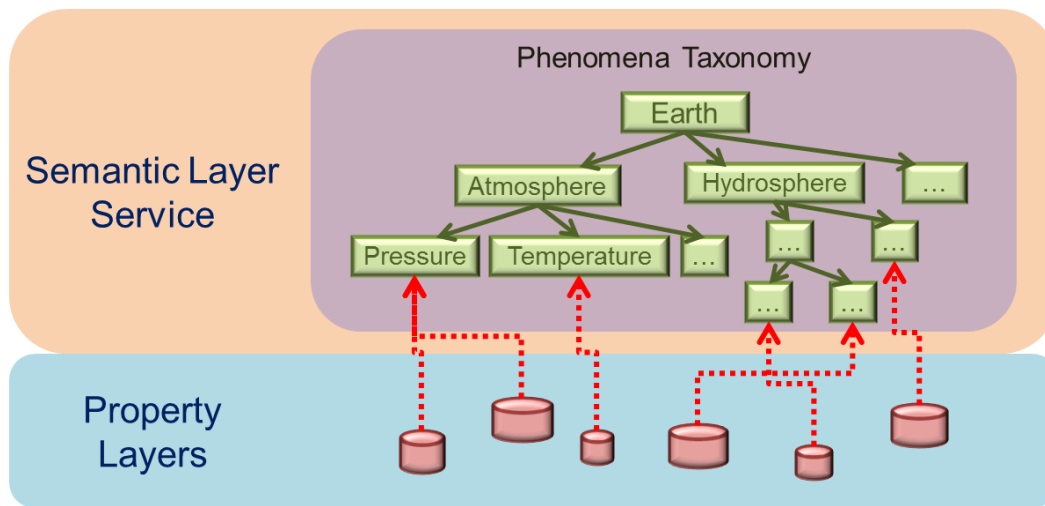


Figure 3.7 High level architecture of the semantic layer service.

In addition, as the phenomenon elements are organized in a meaningful hierarchical structure, elements that are close to each other in the taxonomy are more similar than distant

¹⁵ <http://wordnet.princeton.edu/>

elements. Therefore, with the taxonomy structure, the semantic layer service is able to tackle the semantic heterogeneity issue.

Since the semantic layer service is a cooperative work, this thesis only includes the high-level introduction. Detail algorithms and evaluation can be found in Knoechel et al. (2013).

3.6 AHS-Model

As mentioned earlier, GeoPubSubHub is different from other publish/subscribe system as it handles geospatial sensor web data and queries. While there have been works discussing geospatial operators in a continuous query processing, we argue that time-consuming geospatial operators such as topological operators can be improved by designing algorithms based on the nature of continuous query processing. Therefore, this research proposes the *Aggregated Hierarchical Spatial Model (AHS-Model)* to efficiently determine topological relationships between new data and a large number of predefined queries.

In Chapter 3.6.1, we introduce the background and objectives of AHS-Model. Chapter 3.6.2 defines topological operators and presents DE-9IM, a typical approach for determining topological relationships. The detailed algorithms of AHS-Model are introduced in Chapter 3.6.3. Chapter 3.6.4 presents the AHS-Model architecture applying distributed computing approaches. Finally, we sum up the AHS-Model in Chapter 3.6.5.

3.6.1 Introduction

As mentioned in Chapter 2.1, there have been approaches proposed to optimize query execution plans. However, as most existing approaches are designed for optimizing the structure of query plan(s), there are few works that discuss how to improve the efficiency of

computational-intensive operators. While many general-purpose operators are simple and efficient enough to be used directly in query plans, many geospatial operators are complex and time-consuming. In this research, we choose the topological operators (Herring 2011) as an example.

The topological operators are recognized as time-consuming tasks and would be a performance bottleneck when processing a large number of geometries (Clementini et al. 1994). As the sensor web data are geospatial in nature, supporting topological operators is necessary for a sensor web publish/subscribe system. Therefore, in order to improve the query efficiency of topological operators in a publish/subscribe system, we propose a new topological relationship determination model called *AHS-Model*.

AHS-Model uses two key ideas to efficiently determine topological relationships in a publish/subscribe system. First, as the queries are predefined and continuous in publish/subscribe systems, we can *pre-generate* necessary indices for geometries of subscriptions and *re-use* them every time when needed. In this case, we can avoid generating any redundant index. Although this approach may require more disk space, it is a necessary trade-off to determine topological relationships efficiently. Second, by indexing geometries of subscriptions with the same indexing structure, we can *aggregate* together the indices into one single object. Thus, we can not only reduce the space needed to store the indices, but also intersect the geometries of a new publication and all subscriptions in a single process.

In general, the AHS-Model is inspired by the idea of sharing operators across multiple query plans (Arasu et al. 2004), as described in Chapter 2.1. As the AHS-Model aggregates the

pre-generated indices, the matching between new data and all the subscriptions can be done in a single process. This idea is critical as it allows the AHS-Model to be scalable in terms of *the large number of queries/subscriptions* challenge mentioned in Chapter 2.2.

3.6.2 Topological operators and DE-9IM

Here we define the topological operators and introduce the typical approach to determine topological relationships. The topological operators are functions that determine the topological relationships between two geometries (*i.e.*, point, polyline, polygon, multi-point, multi-polyline, and multi-polygon). The OGC Simple Feature Access Specification (Herring 2011) defines eight topological relationships: *EQUALS*, *DISJOINT*, *INTERSECTS*, *TOUCHES*, *OVERLAPS*, *CROSSES*, *WITHIN*, and *CONTAINS*. This specification has been widely adopted in many spatial databases, such as PostGIS, Oracle, and Microsoft SQL Server.

The typical approach to determine topological relationships is the Dimensionally Extend 9 Intersection Model (DE-9IM) (Clementini et al. 1993). DE-9IM has three steps. First, DE-9IM generates the *interior*, *boundary*, and *exterior* regions of two geometries. The boundary of geometry λ is denoted by $B(\lambda)$ and is defined for each of the geometry types. The boundary of a point geometry is always empty; the boundary of a line geometry is the set of the two separate end-points; and the boundary of a polygon geometry is a circular line surrounding the polygon. As with the definition of boundary, the interior of a geometry λ is denoted by $I(\lambda)$ and is defined as the points that are left when the boundary are removed, that is $I(\lambda) = \lambda - B(\lambda)$. The exterior of a geometry λ is denoted by $E(\lambda)$ and is defined as all the points in the space that are not in the

interior and boundary, that is $E(\lambda) = \mathbb{R}^2 - I(\lambda) - B(\lambda) = \mathbb{R}^2 - \lambda$, where \mathbb{R}^2 is the Euclidean planes.

The second step of DE-9IM is to intersect these interior, boundary, and exterior regions of two geometries and constructs a three-by-three intersection matrix (Equation 3.1). Finally, if the intersection matrix matches the predefined matrices (*i.e.*, Table 3.1), the topological relationships between the two geometries can be determined accordingly. Table 3.1 shows the topological relationships, the definition of relationships, and the corresponding intersection matrices, where the wildcard symbol (*) means “any value would work” (Herring 2011).

$$DE-9IM(a, b) = \begin{bmatrix} \dim(I(a) \cap I(b)) & \dim(I(a) \cap B(b)) & \dim(I(a) \cap E(b)) \\ \dim(B(a) \cap I(b)) & \dim(B(a) \cap B(b)) & \dim(B(a) \cap E(b)) \\ \dim(E(a) \cap I(b)) & \dim(E(a) \cap B(b)) & \dim(E(a) \cap E(b)) \end{bmatrix}, \quad (3.1)$$

where *dim* function returns the maximum dimension (*i.e.*, 0 for points, 1 for lines, and 2 for polygons) of the intersection (\cap) of interior (*I*), boundary (*B*), and exterior (*E*) of geometries *a* and *b*. The geometry *a* and *b* are called *primary* and *secondary* geometry, respectively. If an intersection is an empty set (\emptyset), *dim* function returns -1. Otherwise, if an intersection is not an empty set, *dim* function returns 0, 1, or 2. One way to simplify the matrix is to store only True (if *dim* function returns 0, 1, or 2) and False (if *dim* function returns -1) in the matrix, which is also the representation in Table 3.1.

Please note that for the *CROSSES* relationship, the OGC definition shown in Table 3.1 may not be the most commonly used definition. Instead, the *CROSSES* relationship is usually defined as “ $(\dim(I(a) \cap I(b)) < \max(\dim(I(a)), \dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap$

$b \neq b$)". Hence, the DE-9IM matrix is defined as $dim(I(a) \cap I(b)) = 0$ for line/line relationship. In this research, AHS-Model follows this common definition as well.

However, DE-9IM is a time-consuming process (Clementini et al. 1994) and could be a performance bottleneck when processing a large number of geometries. In order to address this issue, a common solution is to reduce the number of unnecessary DE-9IM processes. For example, a typical approach consists of two stages: *filter* and *refinement* (Clementini et al. 1994). The filter stage finds candidate geometries with *approximated rectangles* of geometries (e.g., minimum bounding rectangles (MBRs)); then the refinement stage performs the actual DE-9IM process on the candidates found in the filter stage. While this approach has been widely applied in many DBMS systems (e.g., PostGIS), we argue that this approach can be further improved for a publish/subscribe system to handle a larger number of geometries.

Although there have been some works that discussed the addition of spatial operators in a publish/subscribe system, most of them simply applied the same filter-and-refinement spatial join approach to prove the concept. For instance, Kassab et al. (2010) utilized the ArcGIS Engine .NET SDK¹⁶ to determine topological relationships. Ali et al. (2010) applied the Microsoft SQL Server Spatial Library to support spatial queries in their Microsoft StreamInsight system. Mokbel et al. (2005) encapsulated geospatial algorithms as operators (e.g., INSIDE and k-Nearest-Neighbor operators) in order to support incremental evaluation and optimize multiple query plans. However, none of these researches discussed how to improve the efficiency of geospatial

¹⁶ ArcGIS Engine: <http://www.esri.com/software/arcgis/arcgisengine/>

algorithms for publish/subscribe systems. We argue that the geospatial algorithms can be improved based on the nature of continuous queries. Therefore, as an example, we propose AHS-Model as a new determination model to improve the efficiency of topological operators for GeoPubSubHub.

Table 3.1 The topological relationships and the corresponding intersection matrices.

| Relationship | OGC Definition (P: point, L: line, A: polygon) | DE-9IM Intersection Matrix (T: True, F: False) |
|--------------------|--|--|
| a EQUALS b | $a \subseteq b \wedge b \subseteq a$ | $\begin{bmatrix} T & * & F \\ * & * & F \\ F & F & * \end{bmatrix}$ |
| a DISJOINTS b | $a \cap b = \emptyset$ | $\begin{bmatrix} F & F & * \\ F & F & * \\ * & * & * \end{bmatrix}$ |
| a INTERSECTS b | $a \cap b \neq \emptyset$ | $\begin{bmatrix} T & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} * & T & * \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} * & * & * \\ T & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} * & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$ |
| a TOUCHES b | $(I(a) \cap I(b) = \emptyset) \wedge (a \cap b \neq \emptyset)$. Applies to P/L, P/P, L/L, L/A, and A/A situations. | $\begin{bmatrix} F & T & * \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} F & * & * \\ * & T & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} F & * & * \\ T & * & * \\ * & * & * \end{bmatrix}$ |
| a OVERLAPS b | $(dim(I(a)) = dim(I(b)) = dim(I(a) \cap I(b))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$. Applies to P/P, L/L, and A/A situations. | $\begin{bmatrix} T & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$ or $\begin{bmatrix} 1 & * & T \\ * & * & * \\ T & * & * \end{bmatrix}$ (for line/line relationship) |
| a CROSSES b | $(I(a) \cap I(b) \neq \emptyset) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$. Applies to P/L, P/A, L/L and L/A situations. | $\begin{bmatrix} T & * & T \\ * & * & * \\ * & * & * \end{bmatrix}$ or $\begin{bmatrix} 0 & * & * \\ * & * & * \\ * & * & * \end{bmatrix}$ (for line/line relationship) |
| a WITHIN b | $(a \cap b = a) \wedge (I(a) \cap E(b) = \emptyset)$ | $\begin{bmatrix} T & * & F \\ * & * & F \\ * & * & * \end{bmatrix}$ |
| a CONTAINS b | $(a \cap b = b) \wedge (I(b) \cap E(a) = \emptyset)$ | $\begin{bmatrix} T & * & * \\ * & * & * \\ F & F & * \end{bmatrix}$ |

3.6.3 Methodology

We propose AHS-Model to improve the query efficiency and scalability of topological operators in a sensor web publish/subscribe system. The AHS-Model follows the definition of the eight topological relationships in OGC Simple Feature Access specification. Hence, AHS-Model follows the same conceptual framework used in traditional approaches such as DE-9IM.

The only difference between AHS-Model and traditional approaches is that the current AHS-Model does not take multi-point, multi-line, and multi-polygon into consideration. Extending the algorithm to support these types of geometries would not change the main idea of AHS-Model, but is one of the future directions that this work will pursue.

Before we introduce the key ideas and algorithms of AHS-Model, we first define the subscriptions and publications in a sensor web publish/subscribe system. In general, subscriptions are continuous queries registered by users; and publications are the sensor data produced by sensors. As sensor data are geospatial in nature, subscriptions and publications both have geospatial components. A subscription (SUB) can have different predicates as query criteria/filters; among which, the spatial predicate in a subscription (SUB_{SP}) has two parameters: a base geometry (SUB_{SP_GEO}) and a topological operator (SUB_{SP_OPER}). These two parameters are set by users to select publications whose geometry (PUB_{GEO}) matches the topological relationship (*i.e.*, SUB_{SP_OPER}) with SUB_{SP_GEO} . For example, in the context of the sensor web a PUB_{GEO} could be a sensor's location or the geometry of a feature the sensor observed (e.g., the coverage of a river or a road intersection).

The relationship between PUB_{GEO} , SUB_{SP_OPER} , and SUB_{SP_GEO} follows the *subject-verb-object* structure, in which PUB_{GEO} , SUB_{SP_OPER} , and SUB_{SP_GEO} are the subject, verb, and object, respectively. For example, if $point_1$ is PUB_{GEO} , $WITHIN$ is SUB_{SP_OPER} , and $polygon_1$ is SUB_{SP_GEO} , the spatial predicate (*i.e.*, SUB_{SP}) evaluates whether the relationship “ $point_1$ $WITHIN$ $polygon_1$ ” is true or not. To be more specific, in this example “send me notification if any water level sensor within 1 km radius of my house reports a reading larger than 50 centimeter”, the SUB_{SP_GEO} is the “1 km radius of my house”, the SUB_{SP_OPER} is the “within”, and the PUB_{GEO} is the location of any water level sensors. Based on this definition and the definitions of topological relationships (Table 3.1), we can derive all the possible topological relationships between different geometry types as shown in Table 3.2.

Table 3.2 The possible topological relationships between different geometry types (●: possible, ○: impossible, *: these relationships are possible if consider multi-point geometry).

| PUB_{GEO} | Point | Line | Polygon | Point | Line | Polygon | Point | Line | Polygon |
|-----------------|-------|-------|---------|-------|------|---------|---------|---------|---------|
| SUB_{SP_GEO} | Point | Point | Point | Line | Line | Line | Polygon | Polygon | Polygon |
| EQUALS | ● | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● |
| DISJOINT | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| INTERESTS | ● | ● | ● | ● | ● | ● | ● | ● | ● |
| TOUCHES | ○ | ● | ● | ● | ● | ● | ● | ● | ● |
| OVERLAPS | ○* | ○ | ○ | ○ | ● | ○ | ○ | ○ | ● |
| CROSSES | ○ | ○* | ○* | ○* | ● | ● | ○* | ● | ○ |
| WITHIN | ● | ○ | ○ | ● | ● | ○ | ● | ● | ● |
| CONTAINS | ● | ● | ● | ○ | ● | ● | ○ | ○ | ● |

Since the targeted objectives of traditional DBMS and publish/subscribe systems are essentially different, algorithms optimized for DBMS may not be suitable for publish/subscribe systems. For example, since the queries in DBMS are atomic and independent, the algorithms are optimized for each individual query. However, since the queries are continuous and pre-defined in publish/subscribe systems, algorithms should consider the aggregation of multiple queries/subscriptions.

Moreover, with the nature of continuous queries, we argue that it is acceptable spending more effort (e.g., create indices) on the start-up preparation stage to execute continuous queries more efficiently. Although this approach may cause some delay in the beginning, it can generate a larger throughput considering the long running nature of continuous query. Therefore, based on these concepts, there are two key ideas in AHS-Model. First, AHS-Model *pre-generates* necessary indices from the geometries of subscriptions and *re-uses* the indices when needed in the continuous queries. Second, by indexing the geometries of subscriptions with the same indexing structure, we can *aggregate* together the indices of all subscriptions to not only save the storage space but also intersect PUB_{GEO} with all SUB_{SP_GEO} in a single process.

AHS-Model consists of the three major stages: (1) Preparation Stage: generate necessary information from the geometries of subscriptions, (2) Intersection Stage: intersect with the geometry of publication, and (3) Determination Stage: determine geospatial relationship. We introduce the details of these three stages in the following sections.

3.6.3.1 Preparation stage: Generate necessary information from the geometries of subscriptions

Similar to the DE-9IM, AHS-Model also determines topological relationships by intersecting the interior, boundary, and exterior regions of two geometries. However, instead of the typical two-step (*i.e.*, filter and refinement) approach, AHS-Model performs candidate-finding and intersections at the same time. This is doable in a sensor web publish/subscribe system since queries are predefined and PUB_{GEO} is usually small (e.g., a sensor's location, a road intersection, or a football field). We first create the regions of the geometries in subscriptions (*i.e.*, SUB_{SP_GEO}) and reuse them until SUB_{SP_GEO} is changed. In this case, we can avoid generating redundant indices, which can consequently speed up the query processing.

However, generating a set of points for interiors, boundaries, and exteriors could create a storage issue. In order to address this, the AHS-Model applies a hierarchical indexing structure, which is similar to the idea proposed by Zimbrao and Souza (1998). In this case, multiple nodes in a lower level can be aggregated as a node in a higher level. In AHS-Model, we use a quadtree tile system (Figure 3.6, Gaede and Gunther 1998) as the hierarchical structure. By defining the lowest level of the quadtree as the granularity, any geometry can be indexed into (or say approximated as) a list of quadtree nodes (*i.e.*, quadkeys). Examples can be seen in Figure 3.8, where the maximum quadtree level is 4 and interiors, boundaries, and exteriors are represented in dark-gray, light-gray, and white respectively. Since lower level indices can be aggregated into higher level indices, the number of indices can be reduced if a large geometry covers multiple quadkeys. Use the upper two polygons in Figure 3.8 as an example. The upper-left polygon is about four times larger than the upper-right polygon. By using hierarchical structure, we can

aggregate upper-left polygon's interior indices from 72 fourth-level indices to 6 indices, which is the same number as the upper-right polygon's interior indices.

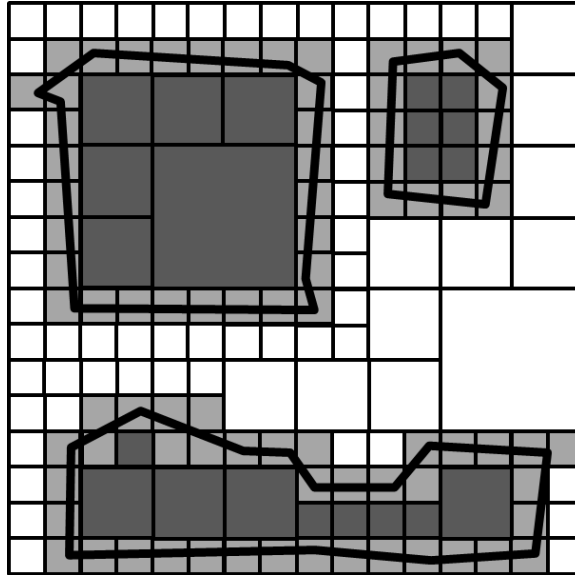


Figure 3.8 Examples of AHS-Model indices with the fourth level as the lowest quadtree level (interior: dark-gray, boundary: light-gray, and exterior: white).

In addition, the determination of a topological relationship does not need all three interior, boundary, and exterior of SUB_{SP_GEO} . Instead, the AHS-Model only needs to generate the necessary information according to the spatial predicate SUB_{SP} . In this case, we can further reduce the number of indices and speed up query processing. Based on the definition of topological relationships in OGC Simple Feature Access Specification (Table 3.1) and the possible topological relationships between different geometry types (Table 3.2), we can analyze and propose the necessary information for determining each geospatial relationship.

However, during a preliminary test of a first version of AHS-Model, we found that the indexing and query performance using exterior indices were poorer than that for interior and




























boundary indices. This is because the number of exterior indices is usually larger than the number of interior and boundary indices. Therefore, we re-designed the AHS-Model to avoid exterior indices. Our final analyses are listed below and the necessary indices are presented in Table 3.3:

- *EQUALS*: Since we can simply compare the interior and boundary of two geometries to determine the *EQUALS* relationship, only the interior and boundary are needed.
- *DISJOINT*: Since the *DISJOINT* relationship can be seen as “no intersection between the interiors and boundaries”, only the interior and boundary are needed to determine *DISJOINT* relationship.
- *INTERSECTS*: Since the *INTERSECTS* relationship means that the two geometries have at least one interior or boundary point in common, only the interior and boundary are needed for this relationship.
- *TOUCHES*: The *TOUCHES* relationship means two geometries are *INTERSECTS* but their interiors do not intersect with each other. Therefore, similar to the *INTERSECTS* relationship, only the interior and boundary are required for the *TOUCHES* relationship.
- *OVERLAPS*: The *OVERLAPS* relationship means that the interior of both geometries intersects the interior and exterior of the other. And if both geometries are line geometries, the intersection of interiors needs to be a line for *OVERLAPS* relationship. In order to avoid the processing of exterior indices, we follow the idea that “if the intersection of interiors and boundaries are not equal to neither geometries, each geometry intersects with the exterior of the other”, that is $(a \cap E(b) \neq \emptyset) \wedge (E(a) \cap b \neq \emptyset)$.

\emptyset) *iff* $(a \cap b \neq a) \wedge (a \cap b \neq b)$. Therefore, only the interior and exterior are required for this relationship. In addition, since multi-point is not in the scope, a single point does not *OVERLAPS* with another point.

- *CROSSES*: The *CROSSES* relationship means that the interior of the primary geometry *a* intersects the interior and exterior of secondary geometry *b*. Similar to the *OVERLAPS* relationship, we replace the requirement of exterior with the intersections of interiors and boundaries. In addition, for line geometries, they are *CROSSES* if and only if their interiors intersect at a point. Since we only consider single point geometries, no geometry can cross a single point based on the OGC definition. As a result, the determination of the *CROSSES* relationship only needs the interior for line geometries and requires both interior and boundary for polygon geometries.
- *WITHIN*: “*a WITHIN b*” means that the interior and boundary of *a* fully locate in the interior and boundary of *b*. That means only the interior and boundary are required to determine *WITHIN* relationship.
- *CONTAINS*: The *CONTAINS* relationship is the inverse of the *WITHIN* relationship, which means “*a CONTAINS b*” and “*b WITHIN a*” are equal. Therefore, the interior and boundary are also the necessary information for *CONTAINS*.

Table 3.3 Necessary subscription indices for AHS-Model.

| Geometry | Point | Line | Polygon |
|-------------------------|---|---|---|
| Geospatial relationship |  |  |  |
| <i>EQUALS</i> |  |  |  |
| <i>DISJOINTS</i> |  |  |  |
| <i>INTERSECTS</i> |  |  |  |
| <i>TOUCHES</i> |  |  |  |
| <i>OVERLAPS</i> |  |  |  |
| <i>CROSSES</i> |  |  |  |
| <i>WITHIN</i> |  |  |  |
| <i>CONTAINS</i> |  |  |  |

After generating the necessary information (Table 3.3) from SUB_{SP} , AHS-Model aggregates the necessary indices from all subscriptions into a single data structure. In the data structure, each quadkey (which is at least one subscription) maintains a list of subscription identifier (SUB_{ID}) and the corresponding property type ($TYPE_{SUB}$) (*i.e.*, interior or boundary). In this case, the AHS-Model can directly match quadkeys of new data with quadkeys in the data

structure to intersect new data with all subscriptions in a single process. Therefore, the worst case scenario is that no quadkey is shared by any other subscriptions (*i.e.*, each quadkey only matches to one SUB_{ID}), which means that the aggregation benefits neither the storage nor the query processing. On the other hand, as long as there is more than one subscription sharing the same quadkey, the aggregation can reduce both the storage size and the query latency. For the remainder of this paper, we refer to this aggregated quadtree structure as AHS_{SUB} for clarity.

A more critical contribution is that by aggregating quadkeys from all SUB_{SP_GEO} into AHS_{SUB} , the AHS-Model can simply match the PUB_{GEO} with the quadkeys in the AHS_{SUB} and link to a set of SUB_{ID} . This means that the AHS-Model decouples quadkeys and SUB_{ID} and allows itself to be more scalable in terms of the number of subscriptions.

3.6.3.2 Intersection stage: Intersect with the geometry of publication

There are three steps for intersecting the geometry of publication (*i.e.*, PUB_{GEO}) with AHS_{SUB} : (1) index, (2) match, and (3) create matrices. This workflow is shown in Figure 3.9. In order to efficiently intersect the PUB_{GEO} with the AHS_{SUB} , AHS-Model first indexes the interior and boundary of PUB_{GEO} with the same hierarchical structure (*i.e.*, quadtree tile system). We call the outcome of this indexing as AHS_{PUB} for clarity.

As mentioned in Chapter 3.6.3.1, AHS-Model modifies the determination algorithm to avoid exterior indices. Therefore, this stage only needs to generate the interior and boundary of the publication geometry.

The matching step benefits from using the same indexing structure. Since both AHS_{PUB} and AHS_{SUB} are indexed by the same quadtree tile system, we can match the prefixes of quadkeys

to find the intersection. To be more specific, every quadkey q has a corresponding geospatial bounding box $bbox$. Given two quadkeys q_A and q_B , $bbox_A \subseteq bbox_B$ if and only if q_A starts with q_B . For example, the bounding box of quadkey ‘0’ contains all the bounding boxes of quadkeys whose first digit is ‘0’. If a quadkey in AHS_{PUB} intersects with a quadkey in AHS_{SUB} , we refer to this intersection as a *match*.

Each match contains the following five attributes: the intersected quadkey from AHS_{PUB} , the intersected quadkey from AHS_{SUB} , subscription identifier (SUB_{ID}), property type of AHS_{PUB} ($TYPE_{PUB}$; *i.e.*, interior or boundary), and property type of AHS_{SUB} ($TYPE_{SUB}$; *i.e.*, interior or boundary). After finding all the matches, we group them by SUB_{ID} and create a matrix for each group. We refer to this matrix as the *area matrix* because it records the size of the intersected area. Similar to three-by-three intersection matrices of the DE-9IM, the area matrices are for determining the topological relationship between geometries, which is the third stage of AHS-Model. The area matrix is a two-by-two matrix, and its form is shown in Equation 3.2.

$$\text{Area matrix}(\text{matches}) = \begin{bmatrix} \text{Area}(II(\text{matches})) & \text{Area}(IB(\text{matches})) \\ \text{Area}(BI(\text{matches})) & \text{Area}(BB(\text{matches})) \end{bmatrix}, \quad (3.2)$$

where the *Area* function returns the sum of the number of *area unit* (here we define that every lowest level quadkey has one area unit and the quadkey on level n has $4^{(\text{lowest level}-n)}$ area units), $II(\text{matches})$ returns intersected quadkeys whose $TYPE_{PUB}$ and $TYPE_{SUB}$ are both interior, $IB(\text{matches})$ returns intersected quadkeys whose $TYPE_{PUB}$ is interior and $TYPE_{SUB}$ is boundary, $BI(\text{matches})$ returns intersected quadkeys whose $TYPE_{PUB}$ is boundary and $TYPE_{SUB}$ is interior,

and $BB(matches)$ returns intersected quadkeys whose $TYPE_{PUB}$ and $TYPE_{SUB}$ are both boundary.

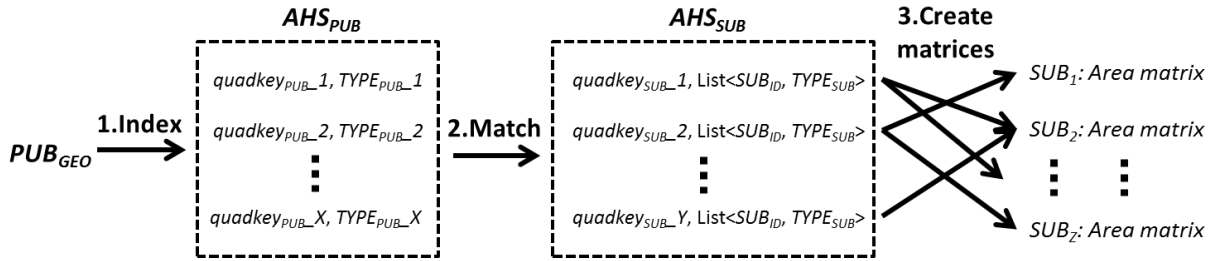


Figure 3.9 The workflow of intersecting the geometry of publication.

3.6.3.3 Determination stage: Determine topological relationship

With the area matrices generated in the previous stage, we can determine the topological relationships between PUB_{GEO} and SUB_{SP_GEO} . Similar to the DE-9IM, each relationship has a specific matrix pattern. Table 3.4 lists the topological relationships and the corresponding area matrices ($AMatrix$), where $Area$ function returns the number of area unit while I and B functions return the interior and boundary of SUB_{SP_GEO} or PUB_{GEO} , respectively. In addition, the $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ represent the cell (0, 0), (0, 1), (1, 0), and (1, 1) in an area matrix, respectively. Finally, $Sum(AMatrix)$ equals to $AMatrix_{II} + AMatrix_{IB} + AMatrix_{BI} + AMatrix_{BB}$.

Please also note that most of the information used during the determination process can be calculated in advance. For instance, $Area(SUB_{SP_GEO})$, $Area(I(SUB_{SP_GEO}))$, $Area(B(SUB_{SP_GEO}))$, $Area(PUB_{GEO})$, $Area(I(PUB_{GEO}))$, and $Area(B(PUB_{GEO}))$ can be generated at the time subscriptions and publications enter the system.

Here we explain the concept of each determination:

- *EQUALS*: If PUB_{GEO} *EQUALS* SUB_{SP_GEO} , their interiors and boundaries should be the same, which means that their interiors are completely intersected with each other and so are their boundaries. Consequently, $AMatrix_{II}$ equals to the area of both interiors, and $AMatrix_{BB}$ equals to the area of both boundaries.
- *DISJOINTS*: If PUB_{GEO} *DISJOINTS* SUB_{SP_GEO} , both the interior and boundary of PUB_{GEO} locate completely in the exterior of SUB_{SP_GEO} . There is no intersection between the interiors and boundaries of PUB_{GEO} and SUB_{SP_GEO} . Therefore, $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ are all zeros.
- *INTERSECTS*: PUB_{GEO} *INTERSECTS* SUB_{SP_GEO} if any interiors or boundaries intersect, which means that one of the cells in $AMatrix$ is not zero.
- *TOUCHES*: If PUB_{GEO} *INTERSECTS* SUB_{SP_GEO} and their interiors do not intersect (*i.e.*, $AMatrix_{II}$ equals to zero), PUB_{GEO} *TOUCHES* SUB_{SP_GEO} .
- *OVERLAPS*: As mentioned in Chapter 3.6.3.1, the processing of exterior indices, is replaced with the intersection of interiors and boundaries as “if the intersection of interiors and boundaries are not equal to both geometries, each geometry interests with the exterior of the other”. Therefore, for *non-line/line* relationships, the determination algorithm becomes “if $AMatrix_{II}$ is not zero and $Sum(AMatrix)$ equals to neither $Area(PUB_{GEO})$ nor $Area(SUB_{SP_GEO})$, PUB_{GEO} *OVERLAPS* SUB_{SP_GEO} ”.

In addition, as the OGC specification defines, the *OVERLAPS* relationship requires the dimension of intersection region to be equal to the dimensions of both geometries. The

intersection of a “line *OVERLAPS* line” relationship should be a line instead of a point. Therefore, for line/line relationship, AHS-Model also checks if $AMatrix_{II}$ is larger than *one* area unit in order to confirm that PUB_{GEO} does not intersect with SUB_{SP_GEO} at a point.

- *CROSSES*: For non-line/line relationships, the determination algorithm for PUB_{GEO} *CROSSES* SUB_{SP_GEO} relationship is the same as that for the *OVERLAPS* relationship. For line/line relationships, as AHS-Model follows the common definition mentioned in Chapter 3.6.2, PUB_{GEO} *CROSSES* SUB_{SP_GEO} if the interiors intersect at a point (*i.e.*, *one* area unit).
- *WITHIN*: The relationship of PUB_{GEO} *WITHIN* SUB_{SP_GEO} means that $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ contain the entire area of PUB_{GEO} (*i.e.*, interior and boundary of PUB_{GEO}).
- *CONTAINS*: PUB_{GEO} *CONTAINS* SUB_{SP_GEO} if $AMatrix_{II}$, $AMatrix_{IB}$, $AMatrix_{BI}$, and $AMatrix_{BB}$ contain the entire area of SUB_{SP_GEO} (*i.e.*, interior and boundary of SUB_{SP_GEO}).

Table 3.4 The topological relationships and the corresponding area matrices.

| Topological Relationship | Area Matrix | |
|---|--|---|
| $PUB_{GEO} \text{ EQUALS } SUB_{SP_GEO}$ | $\begin{bmatrix} Area(I(SUB_{SP_GEO})) & 0 \\ 0 & Area(B(SUB_{SP_GEO})) \end{bmatrix}$ and $\begin{bmatrix} Area(I(PUB_{GEO})) & 0 \\ 0 & Area(B(PUB_{GEO})) \end{bmatrix}$ | |
| $PUB_{GEO} \text{ DISJOINTS } SUB_{SP_GEO}$ | $AMatrix_{II} = AMatrix_{IB} = AMatrix_{BI} = AMatrix_{BB} = 0$ | |
| $PUB_{GEO} \text{ INTERSECTS } SUB_{SP_GEO}$ | $\begin{bmatrix} \neq 0 & * \\ * & * \end{bmatrix}$ or $\begin{bmatrix} * & \neq 0 \\ * & * \end{bmatrix}$ or $\begin{bmatrix} * & * \\ \neq 0 & * \end{bmatrix}$ or $\begin{bmatrix} * & * \\ * & \neq 0 \end{bmatrix}$ | |
| $PUB_{GEO} \text{ TOUCHES } SUB_{SP_GEO}$ | $\begin{bmatrix} = 0 & \neq 0 \\ * & * \end{bmatrix}$ or $\begin{bmatrix} = 0 & * \\ * & \neq 0 \end{bmatrix}$ or $\begin{bmatrix} = 0 & * \\ \neq 0 & * \end{bmatrix}$ | |
| $PUB_{GEO} \text{ OVERLAPS } SUB_{SP_GEO}$ | $AMatrix_{II} \neq 0$ and $Sum(AMatrix) \neq Area(PUB_{GEO})$ and $Sum(AMatrix) \neq Area(SUB_{SP_GEO})$ | For line/line relationship, $AMatrix_{II} > 1$ and $Sum(AMatrix) \neq Area(PUB_{GEO})$ and $Sum(AMatrix) \neq Area(SUB_{SP_GEO})$ |
| $PUB_{GEO} \text{ CROSSES } SUB_{SP_GEO}$ | $AMatrix_{II} \neq 0$ and $Sum(AMatrix) \neq Area(PUB_{GEO})$ and $Sum(AMatrix) \neq Area(SUB_{SP_GEO})$ | For line/line relationship, $AMatrix_{II} = 1$ |
| $PUB_{GEO} \text{ WITHIN } SUB_{SP_GEO}$ | $Sum(AMatrix) = Area(PUB_{GEO})$ | |
| $PUB_{GEO} \text{ CONTAINS } SUB_{SP_GEO}$ | $Sum(AMatrix) = Area(SUB_{SP_GEO})$ | |

3.6.4 Distributed AHS-Model

As mentioned in Chapter 3.1, GeoPubSubHub tries to use distributed computing and cloud computing techniques to provide more memory and CPU resources. The AHS-Model is one of the modules that could benefit from distributed computing techniques. For example, as the number of pre-generated AHS_{SUB} is potentially large, splitting AHS_{SUB} into pieces and storing them in different machines could effectively address the storage issue. In addition, distributing computing techniques could also improve query processing performance especially on indexing and matching stages. Therefore, with these potential benefits, here we slightly modify the AHS-Model architecture based on distributed computing concepts. For the remainder of this paper, we refer to this modified AHS-Model as *distributed AHS-Model* for clarity.

Figure 3.10 shows the high-level architecture and workflow of the proposed distributed AHS-Model. There are four stages in the distributed AHS-Model, namely (1) index, (2) match, (3) aggregate area matrices, and (4) determine relationships. As the processes in the (1) index, (2) match, and (4) determine relationships stages are the same as the processes introduced in Chapter 3.6.3, the (3) aggregate area matrices stage mainly groups and integrates area matrices based on SUB_{ID} .

As shown in Figure 3.10, distributed AHS-Model has a *master* node and a set of *worker* nodes. The master is responsible of receiving subscriptions and publications as well as forwarding subscriptions and publications to appropriate workers. Each worker is in charge of creating and matching indices according to the set of quadkeys assigned to it. That means if a worker is in charge of a quadkey q_A , all the indices that have q_A as a prefix are created and maintained by this worker. The master node has a lookup table storing the set of quadkeys that each worker is responsible of. For the remainder of this thesis, we refer to this set of quadkeys as

$WorkerQ$ and the lookup table as LUT for clarity. Hence, a worker $Worker_i$ has $WorkerQ_i$ as the set of quadkeys it is responsible for.

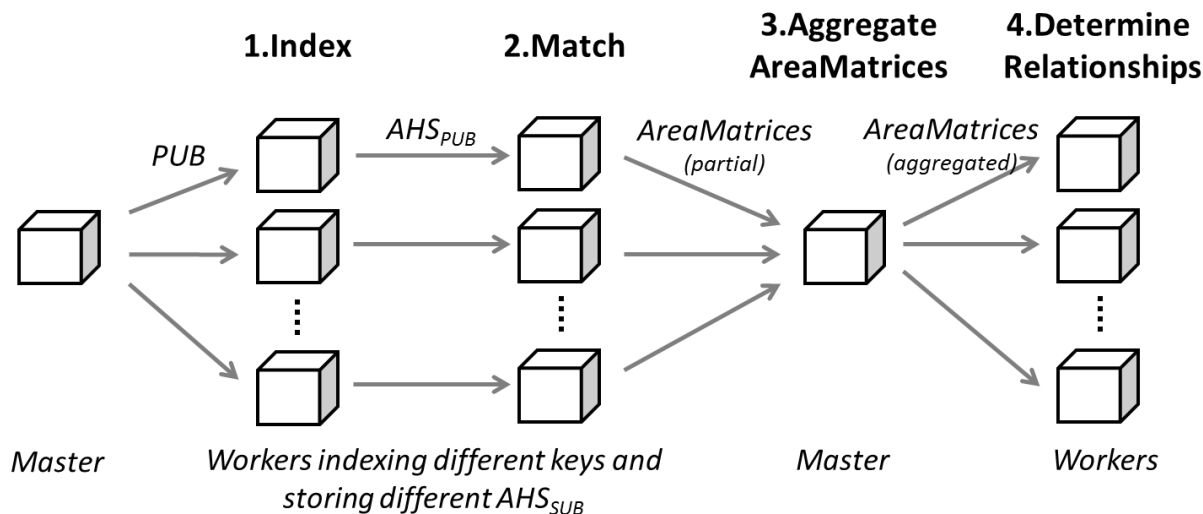


Figure 3.10 High-level architecture and workflow of the distributed AHS-Model.

Algorithm 3.3 shows the worker selection algorithm. When a master node receives a subscription SUB or a publication PUB , the master uses the LUT and SUB_{SP_GEO} or PUB_{GEO} to determine the workers that are responsible of processing the SUB or PUB . Workers are returned if their $WorkerQ$ overlaps with SUB_{SP_GEO} or PUB_{GEO} . In order to reduce the computation load on the master node, we used a coarse estimation on SUB_{SP_GEO} and PUB_{GEO} . That is we first find the lowest level of quadkeys (*i.e.*, $LowestLevel$) in the LUT with the $GetLowestQuadkeyLevel$ function (line 2 of Algorithm 3.3), and generate quadkeys of SUB_{SP_GEO} or PUB_{GEO} (*i.e.*, qs) on the $LowestLevel$ (line 3 of Algorithm 3.3). Then the containing relationship on line 6 of Algorithm 3.3 is determined with the prefix matching of quadkeys from qs and $WorkerQ_i$, which is the same as the prefix matching approach in LOST-Tree. Finally, if quadkeys from qs and $WorkerQ_i$ overlap with each other, the algorithm returns the $Worker_i$.

Algorithm 3.3. The worker selection algorithm.

Function *SelectWorkers(LUT, Geo): Workers*

```

1:  Workers ← {}
2:  LowestLevel ← GetLowestQuadkeyLevel(LUT)
3:  qs ← GetQuadkeysByLevel(Geo, LowestLevel)
4:  FOREACH WorkerQi ∈ LUT
5:      FOREACH q ∈ qs
6:          IF q is contained by WorkerQi OR q contains WorkerQi THEN
7:              IF Workers does not contain Workeri
8:                  Workers ← Workers + Workeri
9:              BREAK
10:         END IF
11:     END FOREACH
12: END FOREACH
13: RETURN Workers

```

Figure 3.11 shows the sequence diagram for registering a subscription. To simplify the diagram, only two workers (*i.e.*, worker 1 and worker 2) are shown. When the master node receives a subscription *SUB*, the master first uses Algorithm 3.3 to select one or more workers, and forwards *SUB* to the selected workers. When a worker receives *SUB*, the worker first creates indices with the SUB_{SP_GEO} based on the quadkeys it is in charge of, and then stores the created indices into the local AHS_{SUB} .

Figure 3.12 shows the sequence diagram for matching a publication. When the master node receives a publication *PUB*, the master first uses Algorithm 3.3 to select one or more workers, and forwards *PUB* to the selected workers. When a worker receives *PUB*, the worker creates indices with the PUB_{GEO} based on the quadkeys it is in charge of, matches AHS_{PUB} with the local AHS_{SUB} , creates area matrices based on the local matches, and finally returns the created area matrices to the master. Since each worker only has a portion of AHS_{SUB} (based on the quadkeys it is in charge of), the created area matrices only represent a portion of the complete area matrices. Therefore, after the workers send the partial area matrices to the master, the master

groups and aggregates them into complete area matrices based on the SUB_{ID} . Finally, the master node equally distributes the aggregated area matrices to workers to determine topological relationships in parallel.

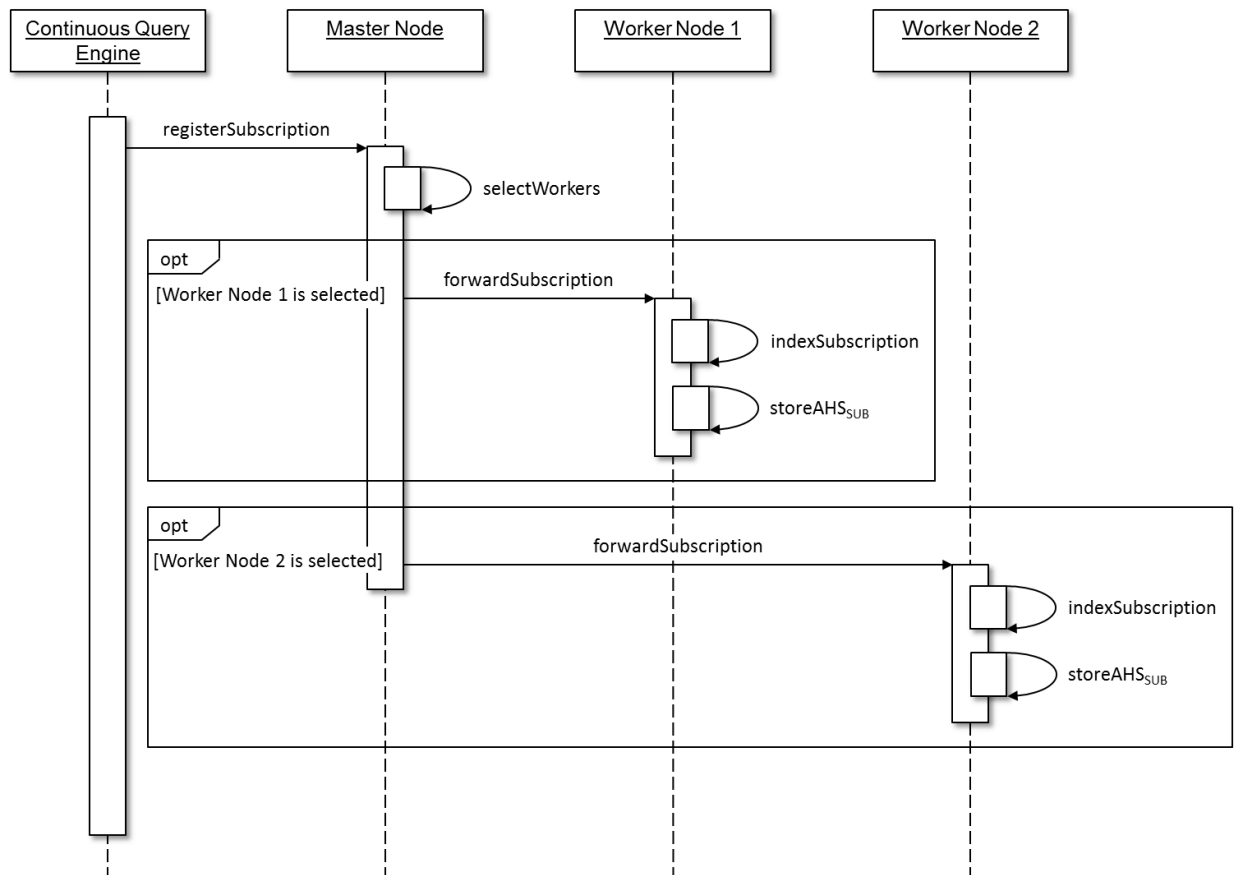


Figure 3.11 The sequence diagram for registering a subscription in distributed AHS-Model.

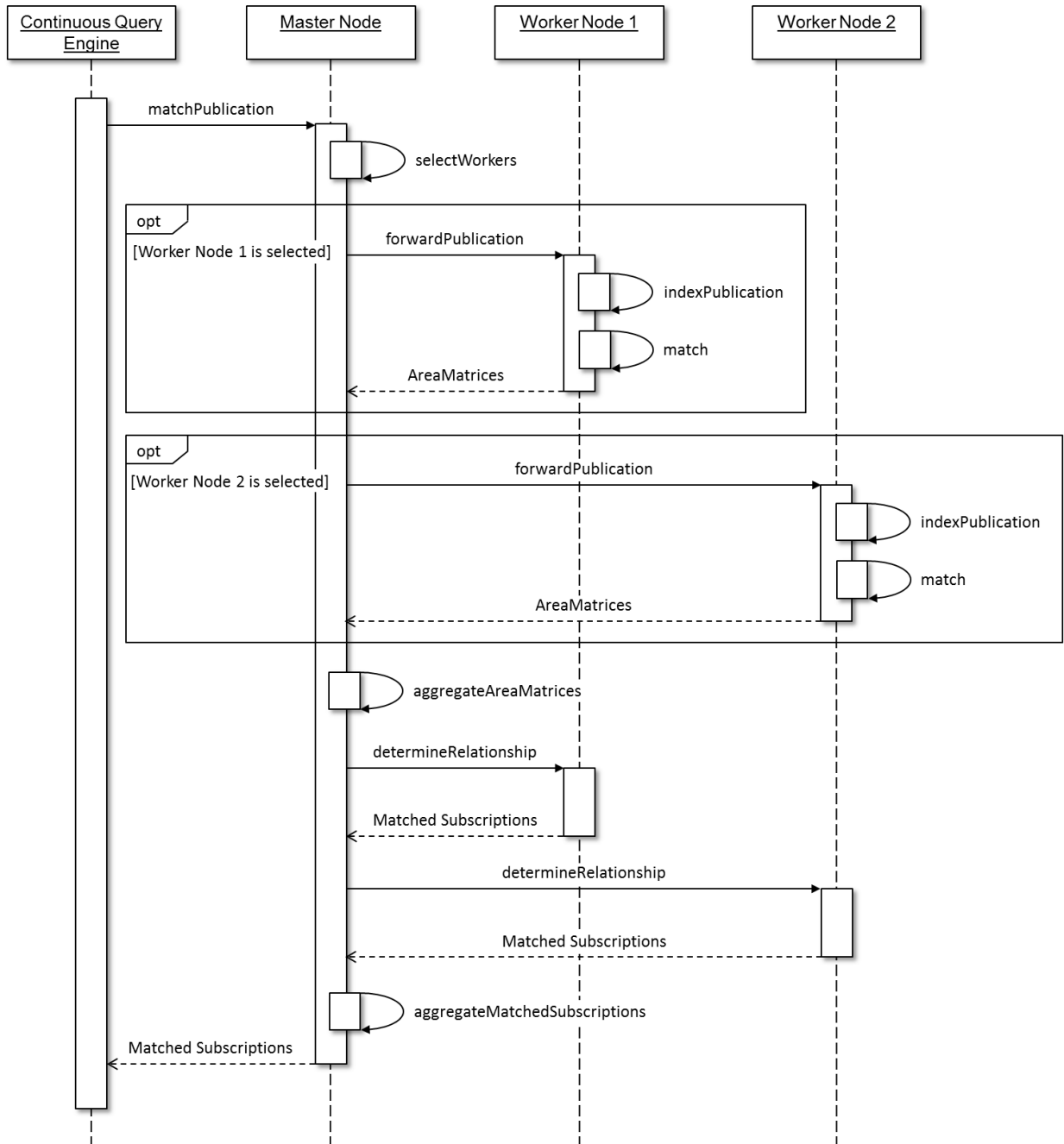


Figure 3.12 The sequence diagram for matching a publication in distributed AHS-Model.

Regarding the load balancing in the distributed AHS-Model, the number of quadkeys in AHS_{SUB} determines the overall performance. That is because the number of quadkeys in AHS_{SUB}

determines the required storage and CPU resources in a worker node. A worker needs more storage and CPU resources to store and process a larger number of quadkeys. Therefore, a simple load balance approach is to assign a threshold θ on the number of quadkeys each worker handles. After each process of registering a subscription, if the number of quadkeys in a worker's AHS_{SUB} is larger than the threshold θ , the worker splits the original AHS_{SUB} into multiple AHS_{SUB} based on the quadkeys this worker is in charge of. These split AHS_{SUB} s are then assigned to other existing or newly created workers; and finally the master updates its lookup table accordingly.

To sum up this section, we further improve the AHS-Model with distributed computing concepts. The proposed distributed AHS-Model is able to harness the storage and CPU resources from multiple machines to address potential storage issues and improve indexing and matching performance. The distributed AHS-Model assigns quadkeys to workers (*i.e.*, $WorkerQ$) to distribute the processing load. This approach allows the distributed AHS-Model to retain the ability to match a publication with all subscriptions in a single process. This is because the quadkeys shared by multiple subscriptions remain aggregated in the same AHS_{SUB} . For example, assuming the case where a quadkey '01' is required by all subscriptions, as only one worker is in charge of quadkey '01', this worker preserves a SUB_{ID} list of all subscriptions that require quadkey '01'. If the worker receives a publication that intersects with the quadkey '01', the worker knows that this match is for all the subscriptions.

In addition, the load balance approach is a simple naïve solution. There are other factors that can be considered in the future. For example, in order to improve service availability and avoid the potential issue of machine failure, the distributed AHS-Model can assign multiple workers to handle the same quadkeys (*i.e.*, replicas). In addition, as the current load balancing approach only considers the geospatial distribution of subscriptions, monitoring the geometries

of publications may allow the adaptive distribution of the processing loads. One of the future directions is to further investigate more sophisticated load balancing approaches for distributed AHS-Model.

3.6.5 Contribution summary

We have presented the AHS-Model, a model that efficiently determines the topological relationship between the geometries in a sensor web publish/subscribe system. Because of the potentially for very large amounts of sensor web data, the continuous query processing model is increasingly attracting interest for many time-critical applications. However, we argue that time-consuming geospatial operators are not suitable for applications require timely processing and notification. The AHS-Model is one example showing that traditional topological operators can be re-designed as efficient continuous query operators in the context of publish/subscribe systems.

The proposed AHS-Model applies two key ideas. First, it pre-generates necessary quadkeys for geometries of subscriptions and re-uses these quadkeys whenever needed. Second, the AHS-Model aggregates together the quadkeys of different subscriptions to save storage space and intersect the geometries between a publication and all subscriptions in a single matching process. Furthermore, the AHS-Model is a complete model in comparison to existing works (Kassab et al. 2010; Mokbel et al. 2005), because it is able to determine all the topological relationships that are defined in the OGC Simple Feature Access specification.

However, since the AHS-Model pre-generates quadkeys, a large number of quadkeys could cause a storage issue even after applying the quadtree tile system to reduce the number of quadkeys. To address this issue, we further propose a distributed AHS-Model to harness storage

and CPU resources from multiple machines. In this case, we can not only address the storage issue, but also improve query efficiency.

The evaluation of the AHS-Model includes (1) scalability analysis in terms of the number of subscriptions, (2) evaluation of indexing latency when handling geometries in various sizes, (3) evaluation of matching latency when handling geometries in various sizes, and (4) end-to-end performance analysis with simulated city-level subscriptions and publications. Our evaluation shows that the AHS-Model is more scalable than PostGIS, can efficiently index and match large geometries with multiple worker nodes, and can efficiently determine topological relationships with an acceptable overhead. The detail of this evaluation is presented in Chapter 4.3.

3.7 Sensor web browser

As mentioned in Chapter 3.4.1, a sensor web browser is a client-side component that loads sensor data from sensor web data sources, and renders these data in a coherent and unified geographical environment. With the sensor web browsers we developed, users can browse, discover, visualize, and access heterogeneous sensing resources from both OGC SOS and WMS. During my Ph.D. study, I have developed two types of sensor web browsers. One is a 3D virtual-globe-based sensor web browser, and the other one is a light-weight 2D map-based sensor web browser. Both of them are developed for the GeoCENS project and have been publicly available¹⁷ since 2010.

3.7.1 3D virtual-globe-based sensor web browser

The 3D virtual-globe-based sensor web browser was developed on top of the open source WorldWind virtual globe system¹⁸. To the best of our knowledge, it is the world's first OGC-

¹⁷ <http://dev.geocens.ca/>

¹⁸ <http://worldwind.arc.nasa.gov/>

based sensor web 3D browser. The GeoCENS browser has two unique components. First, in order to interoperate with existing sensor web servers, an OGC SWE communication module was developed to communicate with OGC SWE-compatible servers. Second, in order to prevent transferring large volume of sensor data across the network repeatedly, we designed and developed the LOST-Tree (introduced in Chapter 3.4) to control sensor data loading from local cache or data sources.

LOST-Tree is the most critical component for this 3D sensor web browser. As efficiently transmitting large amounts of sensor data over the WWW is known to be a major challenge (Nath et al. 2006), LOST-Tree is able to determine whether or not a spatio-temporal request has been sent previously in order to filter out redundant requests. As a result, with the help of a local cache, LOST-Tree allows the sensor web browser to load sensor data efficiently. A screenshot of the 3D virtual-globe-based sensor web browser is shown in Figure 3.13.



Figure 3.13 A screenshot of the 3D virtual-globe-based sensor web browser.

3.7.2 2D map-based sensor web browser

In addition to the 3D virtual-globe-based sensor web browser, we also develop a light-weight 2D map-based sensor web browser. The 2D sensor web browser is developed based on the Microsoft Bing Maps AJAX Control¹⁹. However, unlike the 3D sensor web browser which is a pure client-side component, the 2D sensor web browser retrieves cached sensor data from a mediator called as the *translation engine*. This design was adopted because of the heavy communication loads of OGC sensor web services (e.g., SOAP and/or XML). A sensor web browser that wants to communicate with OGC services independently becomes heavy as well, which is the case for the 3D sensor web browser. In the 3D sensor web browser, the libraries used to handle communication with OGC services weights 2.7 Mbytes.

Therefore, with the help of a translation engine, the 2D sensor web browser is developed as a lighter choice. While the translation engine handles the heavy communication load with OGC sensor web services, the 2D sensor web browser can retrieve the cached sensor data from the translation engine in a light-weight and efficient manner. In this case, the 2D map-based sensor web browser is mobile-friendly, and users can use it on modern smart mobile devices (*i.e.*, smartphones, tablets). The details of the translation engine are published in Knoechel et al. (2011). A screenshot of the 2D map-based sensor web browser is shown in the Figure 3.14.

In addition, to update the cached data in a timely manner, the translation engine utilizes the adaptive sensor stream feeder mentioned in Chapter 3.3.2. The adaptive feeder detects the data sampling period and fetches the latest sensor data from the services by scheduling requests adaptively. In this case, the cached sensor data in the translation engine can always be up-to-date.

¹⁹ <http://msdn.microsoft.com/en-us/library/gg427610.aspx>



Figure 3.14. A screenshot of the 2D map-based sensor web browser.

Chapter Four: **Evaluation and Results**

As mentioned in Chapter 1.3, this thesis tries to cover all aspects of a geospatial sensor web publish/subscribe system by identifying challenges (in Chapter 2), proposing possible solutions (in Chapter 3.1), and designing an overall system architecture and workflow (in Chapter 3.2). One of our future directions is to further investigate the challenges presented in Chapter 2.2.

As the scope of GeoPubSubHub is large, this thesis focuses on the modules that we believe are most unique and critical in the context of a geospatial sensor web publish/subscribe system. Therefore, we proposed the sensor web input adaptor (in Chapter 3.3), LOST-Tree (in Chapter 3.4), and AHS-Model (in Chapter 3.6) as three new solutions in a sensor web publish/subscribe system. In this chapter, we present the evaluations on these solutions.

4.1 Evaluation on the sensor web input adaptor

As mentioned in Chapter 3.3, the proposed sensor web input adaptor has two major components, namely the query aggregator and the adaptive sensor stream feeder. While the query aggregator applies the proposed LOST-Tree to aggregate overlapped spatio-temporal cubes, the evaluation of LOST-Tree is presented in Chapter 4.2. Therefore, this section mainly presents the experimental results of the adaptive sensor stream feeder.

We tested the proposed adaptive sensor stream feeder with two real-world OGC SOSs (here we name them as *service A* and *service B*). While the sensors in both services have similar sampling periods (around 15 minutes), these two services have different data update behaviors. Service A makes the sensor data available as soon as their sensors perform observations, which matches our assumptions mentioned in Chapter 3.3.2. Service B buffers sensor data before making them available online, which results in large differences between the sampling time and

valid time. To be more specific, service B releases their data 60 to 150 minutes after the data was collected by sensors. Therefore, these two services are suitable to examine the performance of adaptive feeder as one service matches with our assumptions and the other one does not.

In this evaluation, we set the buffer time (*i.e.*, the b in Figure 3.3) to 30 seconds to accommodate possible delay. In this case, retrieving requests are sent 30 seconds later than the actual predicted time points. In reality, the best buffer time setting would be the largest possible delay.

In order to evaluate how “real-time” the data that the proposed adaptive feeder retrieves, we calculate the time difference between the time point at which we receive new data and the time point that the latest reading was measured. Table 1 shows the statistics of experimental results including the testing duration, average time difference, number of empty-hit requests (*i.e.*, requests that did not retrieve any new data), and the total number of requests performed during the experiments. In addition, Figure 4.1 and Figure 4.2 show the time differences of each non-empty-hit request for service A and service B, respectively.

Table 4.1 Statistics of adaptive sensor stream feeder evaluation

| | Service A | Service B |
|----------------------------------|-----------|-----------|
| Testing duration (minute) | 355 | 16,830 |
| Average time difference (second) | 28.926 | 4,603.339 |
| Number of empty-hit requests | 0 | 1,005 |
| Total number of requests | 23 | 1,123 |

Since the data retrieving behavior of service A is relatively stable, we only test it for around six hours (355 minutes). However, for service B, since we found that service B’s data

updating behavior varied, we test it for a longer period, which is around 11 days (16,830 minutes).

As we can see in the column of service A (*i.e.*, the best scenario) in Table 4.1, the adaptive feeder can retrieve new data with around 29 seconds time difference. In addition, Figure 4.1 shows that the maximum delay is less than 3 seconds, which is 32.4 - 30 seconds, during our experiment. Therefore, for future usages, the buffer time for service A can be set around 3 seconds to retrieve data more promptly. Moreover, during the testing period, as all 23 requests are able to retrieve new data, there is no unnecessary request in the case of service A. Hence, this evaluation proves that if a service matches our assumptions, the adaptive feeder is able to retrieve near-real-time sensor data while avoiding unnecessary requests.

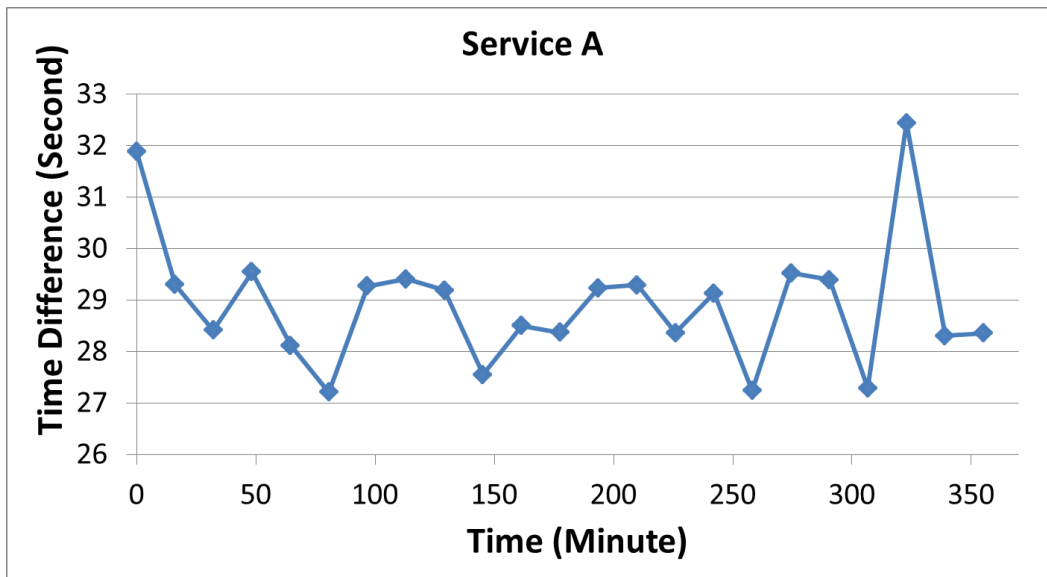


Figure 4.1 Time differences of adaptive feeder requests for service A.

However, as we can observe from Figure 4.2, service B does not make data available online as soon as they are measured, and the delay varies over time. To be more specific, every new data available on service B are measured by sensors more than one hour ago. We suspect

that this delay could be because of data post-processing, data transmission, system design, or limitations from sensors' power constraints. While the adaptive feeder cannot retrieve near-real-time data from service B, it still sends retrieving requests every detected sampling period (about 15 minutes). Although many of the requests would be empty-hit requests (about 90%), the adaptive feeder is able to retrieve new data no later than the detected sampling period after they become available in the service.

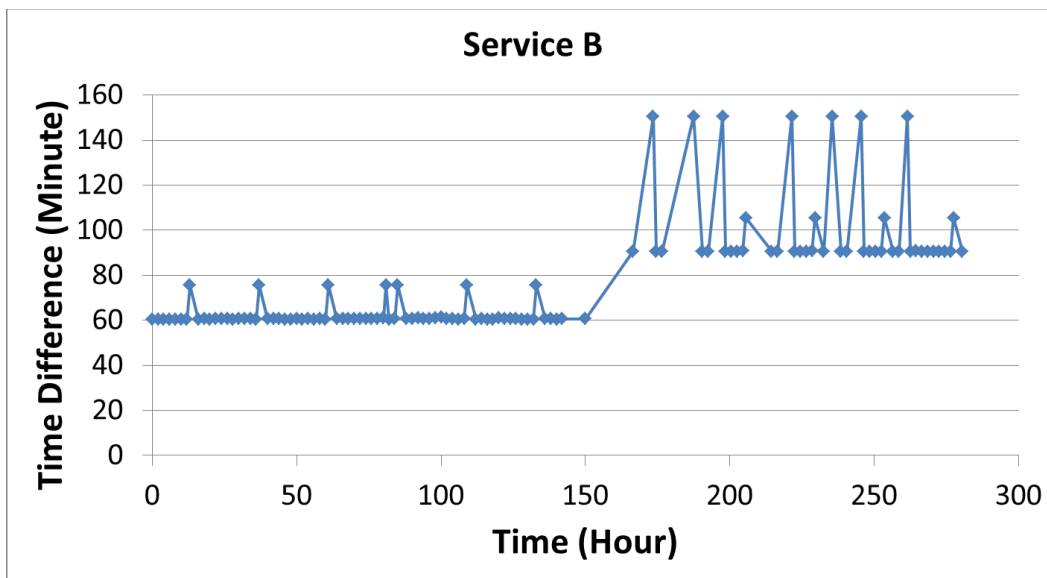


Figure 4.2 Time differences of adaptive feeder requests for service B.

To sum up, as shown in the experimental results, if a service makes data available online as soon as they are measured, the proposed adaptive feeder can retrieve sensor data in a timely manner without unnecessary request. For services that do not release their data efficiently, with a trade-off of redundant requests, the adaptive feeder can still retrieve new data no later than the detected sampling period after they become available online. However, it is arguable that this type of services (such as service B) may not be suitable as a data source for time-critical applications in the first place.

4.2 Evaluation on the LOST-Tree

In order to provide a comprehensive analysis, we evaluated the performance of our LOST-Tree implementation from three perspectives. First, in order to demonstrate the efficiency of the data loading, we compared the end-to-end latencies of loading sensor data with and without the LOST-Tree scheme. Second, in order to show that LOST-Tree is lightweight and scalable in terms of the number of LT_{CCubes} , we measure the LOST-Tree size on spatial, temporal, and spatio-temporal aggregations. Third, in order to demonstrate that LOST-Tree can effectively and efficiently filter out LT_{CCubes} from $LT_{STCubes}$ while keeping the number of requests small, we analyzed the LOST-Tree performance (including latency of filtering, filter efficiency, and number of requests) on different L_q and L_{gc} settings.

Since LOST-Tree was originally proposed as the data loading component in a sensor web browser, the evaluation presented here is mainly based on the perspective of a sensor web browser. As the GeoPubSubHub input adaptor also utilizes LOST-Tree to aggregate spatio-temporal cubes, it focuses more on the effectiveness of filtering out the overlapped spatio-temporal areas (*i.e.*, filter efficiency).

For the second and the third evaluations, we simulated three testing environments: (1) modify the spatial components (*i.e.*, spatial aggregation and L_q) with fixed temporal components; (2) modify the temporal component (*i.e.*, temporal aggregation and L_{gc}) with fixed spatial components; and, (3) modify both spatial and temporal components. The evaluations were performed on a desktop-class machine, which runs an Intel Core2 Quad Q8200 @ 2.33GHz, 3GB RAM, Western Digital WD5000AAKS, and ATI Radeon HD 3450.

4.2.1 Evaluation of data loading efficiency

In this section, we present the analysis of the data loading efficiency between the proposed scheme and a naïve solution. The naïve solution is the case that the client does not store loaded data in the local cache and needs to load all data from the server. We compare the end-to-end latencies of the loading data from a real-world OGC SOS. The end-to-end latency includes the latency of retrieving sensor data (from local cache or server) and the latency of creating icons for visualization. The server located in the same local network with the testing machine and hosted data from the U.S. National Oceanic and Atmospheric Administration (NOAA), which contains 2,412 sensors collecting observations every hour.

Figure 4.3 shows the evaluation results. The end-to-end latencies with and without cache are on a different order of magnitude. Without the proposed scheme, sensor data have to be transmitted from the server through network. With LOST-Tree and a local cache, the previously loaded sensor data could be retrieved from a local disk, which is much faster than transmitting data over the network. While the network is relatively unstable and results in large latency differences between each transmission, data loading from the disk has a more stable performance.

The latency of loading data from a disk is mainly related to the data management method. As LOST-Tree is flexible enough to work with any data management method, we simply used an R-Tree and a B-Tree to manage sensor data in this evaluation. As a result, Figure 4.3 indicates that the proposed scheme can load sensor data significantly faster than the naïve solution. Although the R^2 values of regression lines in Figure 4.3 are low, we can still observe that the proposed scheme could load sensor data at least 100 times faster than the naïve solution. In addition, as the naïve solution needs to handle SOS XML responses containing a large number of

readings, the evaluation stopped earlier than the proposed scheme due to out-of-memory exceptions. Although the latency difference between loading data from a disk and a server is already well-known, LOST-Tree is the component enabling a spatio-temporal data cache in a sensor web browser by serving as a data loading layer.

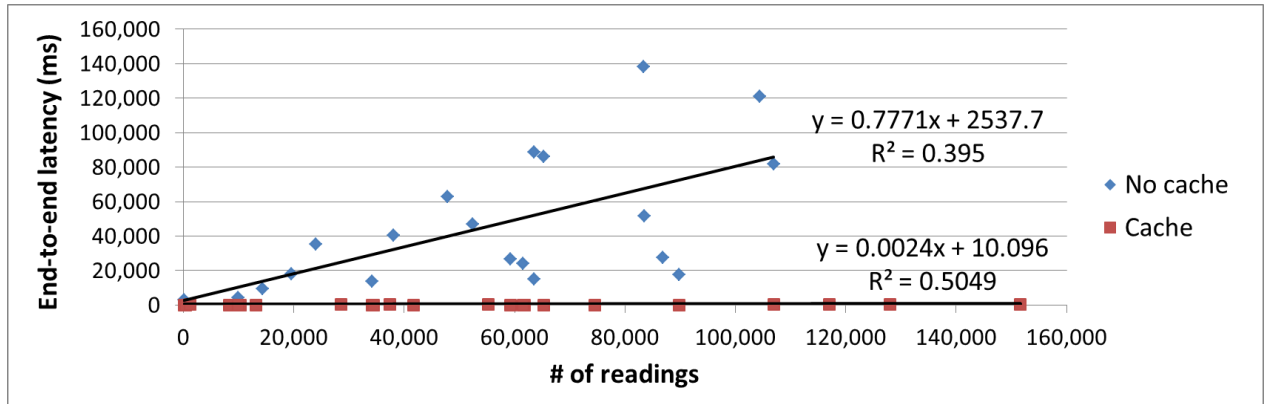


Figure 4.3 End-to-end latencies

4.2.2 Evaluation of LOST-Tree size

Keeping tree size small is very important for storage and computation efficiency. With the predefined hierarchal spatial and temporal frameworks in LOST-Tree, we can aggregate multiple LT_{CCubes} into one LT_{CCube} to reduce the LOST-Tree size. In this section, we first show the aggregation behaviors of quadtree and the Gregorian calendar separately by assigning a simulated loading sequence. We then present the integrated spatio-temporal aggregation behavior with a random loading sequence.

First, in order to show the spatial aggregation behavior, we loaded $LT_{STCubes}$ with all the quadkeys on the fourth level and a fixed Gregorian calendar index. We used the Z-order sequence (Morton 1966) as the simulated loading sequence. Z-order is a space-filling curve that can map multi-dimensional data into one dimension. The Z-order sequence on the fourth level of

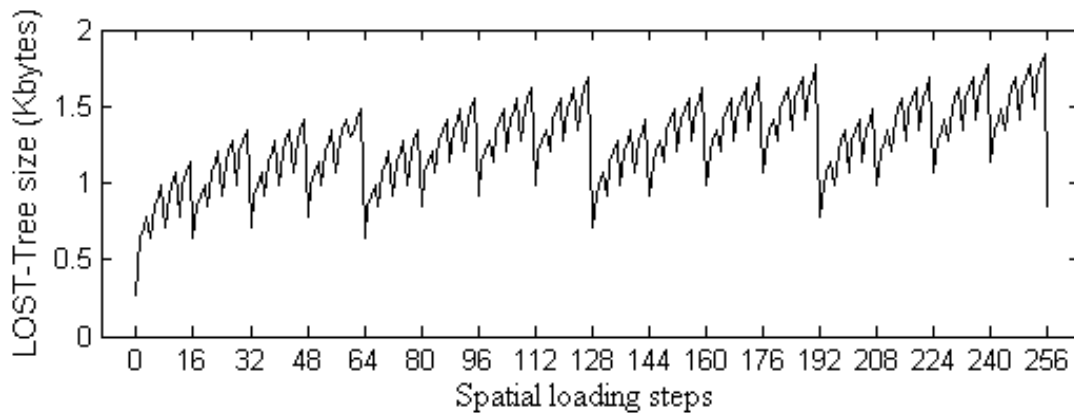


Figure 4.5 The LOST-Tree size reductions on spatial aggregations.

Second, in order to show the temporal aggregation behavior, we load every hour for a one-year period sequentially with a fixed quadkey. In this scenario, LOST-Tree performs temporal aggregation when meet any of the following three scenarios: (1) after loading the last hour in a day, (2) after loading the last day in a month, and (3) after loading the last month in a year. While temporal aggregation combines 60 (seconds, minutes), 24 (hours), 28 or 29 or 30 or 31 (days), and 12 (months) LT_{CCubes} into 1 (an aggregated minute, hour, day, month, year) LT_{CCube} , Figure 4.6 shows that the LOST-Tree size decreases when a temporal aggregation happens.

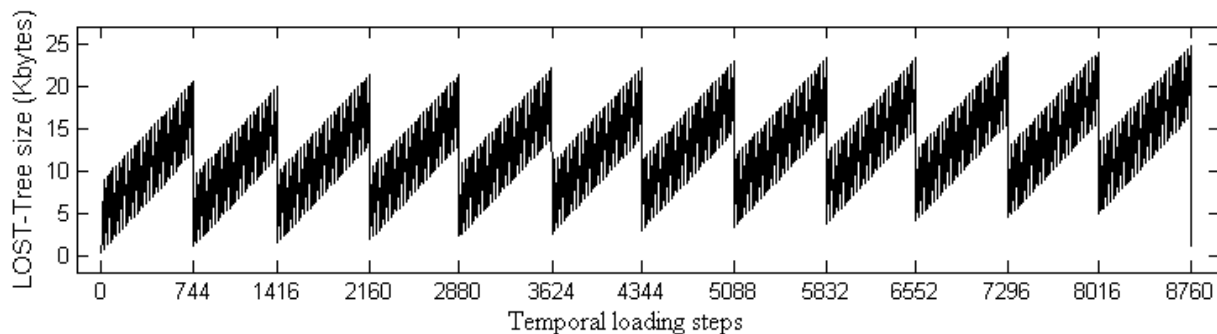


Figure 4.6 The LOST-Tree size reductions on temporal aggregations.

Finally, in order to show the integrated spatio-temporal aggregation behavior, we load $LT_{STCubes}$ in a random sequence, where the quadkeys are in the fourth level and Gregorian calendar indices are the days of a year. As shown in Figure 4.7, the LOST-Tree size decreases whenever a spatial or temporal aggregation is performed. Once the whole globe and the year are loaded, LOST-Tree becomes very small (2.7 Kbytes).

To sum up this section, by applying hierarchal spatial and temporal frameworks in LOST-Tree, spatial and temporal indices can be aggregated to reduce the number of indices. During the evaluation, the size of LOST-Tree is always small enough (from 0.3 Kbytes to 163.8 Kbytes) to be fit into memory for storage and computational efficiency. This evaluation also demonstrates that LOST-Tree is scalable in terms of the number of LT_{CCubes} .

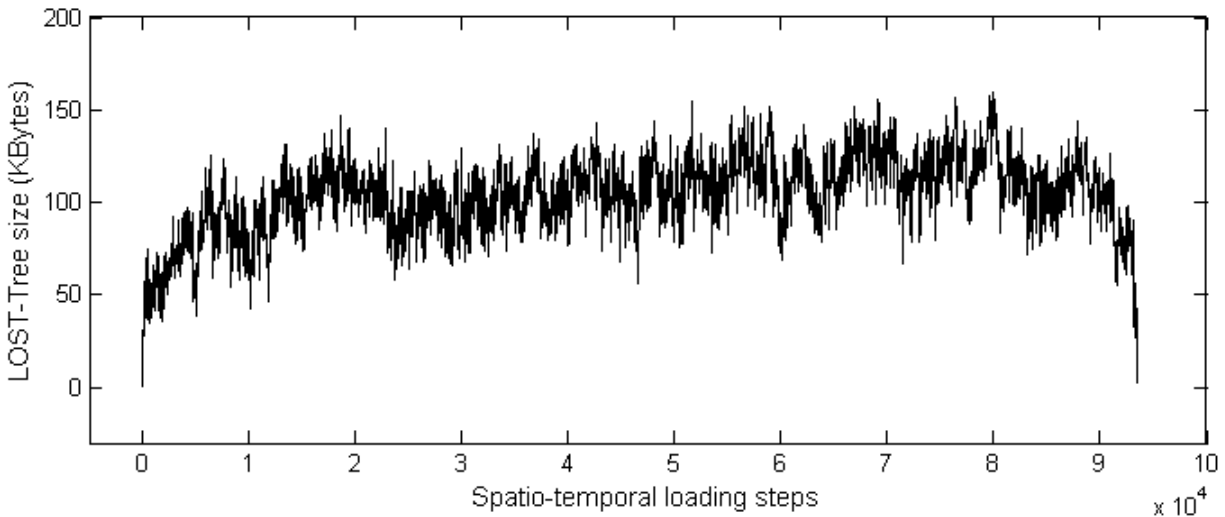


Figure 4.7 The LOST-Tree size reductions on both spatial and temporal aggregations.

4.2.3 Evaluation of LOST-Tree performance

In this section, we evaluate LOST-Tree's performance. As mentioned in the LOST-Tree methodology section (Chapter 3.4.2.2), in order to control the trade-off between reducing the

unnecessary transmissions and keeping the number of requests small, users can use L_q and L_{gc} to specify the lowest q and gc levels to which $LT_{STCubes}$ can be decomposed. Since L_q and L_{gc} may affect the computational load in the filtering process, we analyzed the LOST-Tree performance with different L_q and L_{gc} settings. We used three metrics to evaluate LOST-Tree's performance, namely the query latency (*i.e.*, latency of the filtering process), filter efficiency (*i.e.*, percentage of the unloaded portion in the $FilteredLT_{STCubes}$, which is the larger the better), and number of requests (*i.e.*, number of $FilteredLT_{STCubes}$, which is the smaller the better).

The evaluation in this section has two objectives. The first objective is to evaluate whether or not LOST-Tree can completely filter out LT_{CCubes} that have higher levels of q and gc than L_q and L_{gc} . We first simulate LT_{CCubes} with different levels of q and gc , and then examine the LOST-Tree performance with different L_q and L_{gc} settings. The second objective is to propose a reasonable L_q and L_{gc} setting for a sensor web browser application. We first simulate more realistic LT_{CCubes} based on our knowledge about users' behavior on a sensor web browser. We then analyze LOST-Tree performance with different combinations of L_q and L_{gc} .

In general, we have three evaluations in this section. The first evaluation (Chapter 4.2.3.1) analyzes how different L_q settings affect the LOST-Tree performance when filtering out LT_{CCubes} on different levels of q . The second evaluation (Chapter 4.2.3.2) examines the LOST-Tree performance using different L_{gc} settings to filter out LT_{CCubes} on different levels of gc . Finally, the third evaluation (Chapter 4.2.3.3) demonstrates the expected LOST-Tree performance in a real-world application with different combinations of L_q and L_{gc} .

Table 4.2 details the simulated scenarios for the above three evaluations. In the first two evaluations, since we try to examine the LOST-Tree performance under all possible circumstances, we evaluated four different numbers for LT_{CCubes} (*i.e.*, 1, $33\% \times n$, $66\% \times n$, and $n-1$

cubes, where n is the total number of spatio-temporal cubes with the assigned q and gc levels). However, based on our experience from the GeoCENS sensor web browser, the number of LT_{CCubes} is usually small in reality. Thus, the results in the first two evaluations cannot represent the LOST-Tree performance in a real-world application as they are affected by the large number of simulated LT_{CCubes} .

Based on our experience, we have also noticed that in reality, q usually occurs on the second to eighth levels of quadtree (mostly at the second to fifth levels), and gc usually happens on the month to hour level of the Gregorian calendar (mostly at the day level). Therefore, for the third evaluation, we simulate LT_{CCubes} with q and gc from these levels to provide the expected LOST-Tree performance in a real-world sensor web browser. Moreover, for the third evaluation, the numbers of LT_{CCubes} are also set closer to real-world scenarios (*i.e.*, 1, 50, and 150 cubes). Finally, in order to reduce the influence of outliers, all these evaluations were tested 10 times and the final results are the average values of these tests.

Table 4.2 Settings of simulated scenarios for LOST-Tree evaluations.

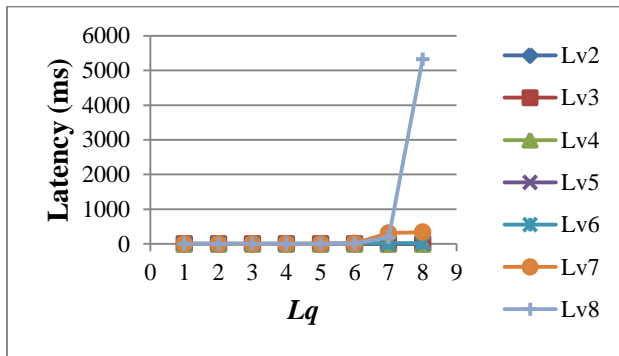
| Settings Evaluations | L_q | L_{gc} | q | gc | Number of LT_{CCubes} * |
|------------------------------------|-------|-----------|--|--|--|
| L_q | 1~8 | year | Randomly picked from levels 2 to 8 in quadtree | Fixed | 1, 33% $\times n$, 66% $\times n$, $n-1$ |
| L_{gc} | 1 | year~hour | Fixed | Randomly picked from levels of month to hour in the Gregorian calendar | 1, 33% $\times n$, 66% $\times n$, $n-1$ |
| Combinations of L_q and L_{gc} | 1~8 | year~hour | Randomly picked from levels 2 to 8 in quadtree | Randomly picked from levels of month to hour in the Gregorian calendar | 1, 50, 150** |

* n is the total number of spatio-temporal cubes with the level of q in quadtree and the level of gc in the Gregorian calendar.

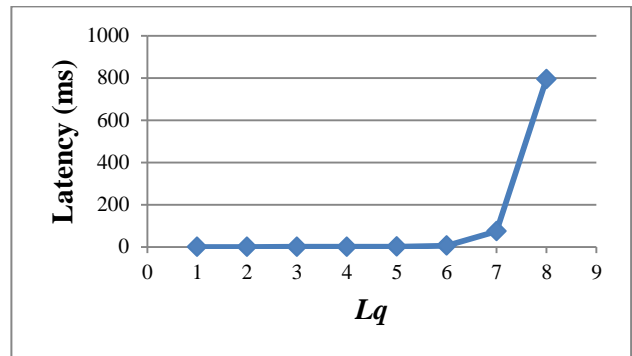
** If n is smaller than the assigned number of LT_{CCubes} , skip the scenario.

4.2.3.1 Evaluation of LOST-Tree Performance on Different L_q

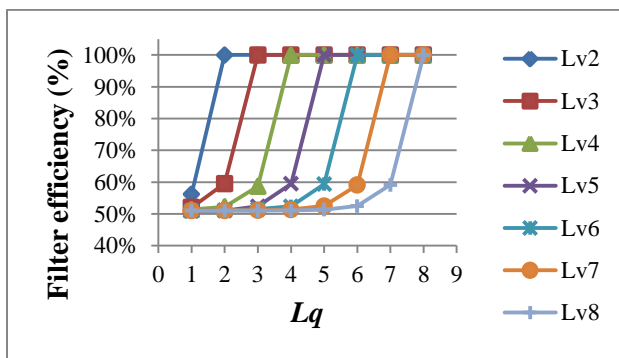
Figures 4.8(a) to 4.8(c) depict the LOST-Tree performance on different L_q and levels of q , and Figures 4.8(d) to 4.8(f) present the average LOST-Tree performance across simulated scenarios. As shown in Figure 4.8(b), LOST-Tree can completely filter out the LT_{CCubes} whose q is smaller than or equal to L_q . Although a large number of LT_{CCubes} seldom occur in reality, we simulated large numbers of LT_{CCubes} to demonstrate that LOST-Tree can filter out LT_{CCubes} under any scenario. Therefore, filtering out the large number of LT_{CCubes} results in the increases of the query latency and the number of requests. For example, in the case of L_q equal to 8, the average query latency jumps from less than 100 milliseconds to more than 800 milliseconds (Figure 4.8(d)); and more than 2,500 requests, on average, are generated (Figure 4.8(f)). Therefore, if consider all possible scenarios, we would make a theoretical suggestion to configure L_q as 5 or 6 to avoid large query latency and a large number of requests and still attain a filter efficiency more than 80%.



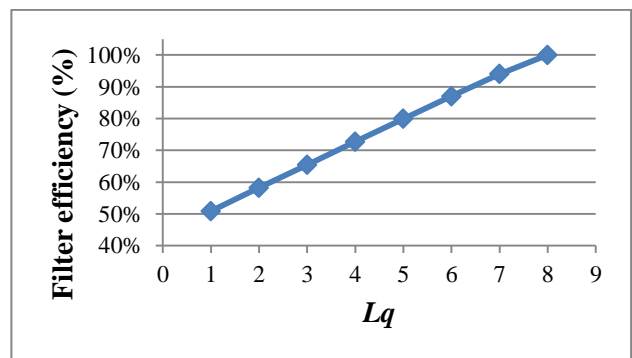
(a)



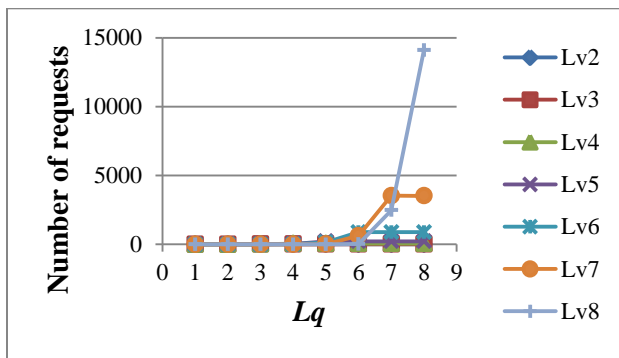
(d)



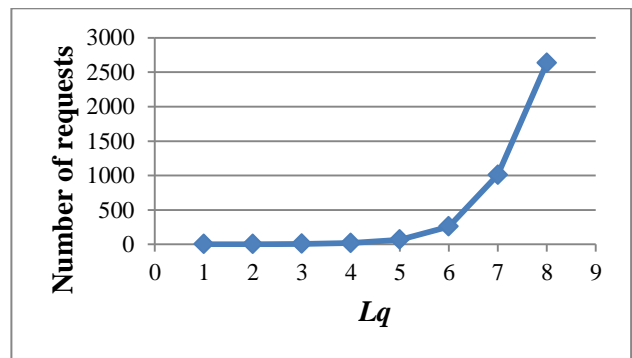
(b)



(e)



(c)



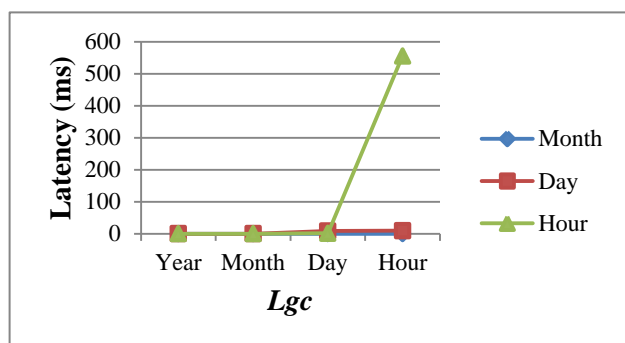
(f)

Figure 4.8 LOST-Tree performance on different L_q : (a) query latencies for different levels of q ; (b) filter efficiencies for different levels of q ; (c) number of requests for different levels of q ; (d) average query latencies; (e) average filter efficiencies; (f) average number of requests.

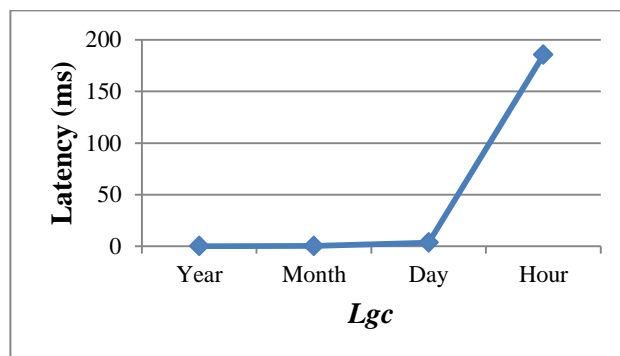
4.2.3.2 Evaluation of LOST-Tree Performance on Different L_{gc}

Figures 4.9(a) to 4.9(c) depict the LOST-Tree performance on different L_{gc} and levels of gc in LT_{CCubes} , while Figures 4.9(d) to 4.9(f) present the average LOST-Tree performance across simulated scenarios. As shown in Figure 4.9(b), LOST-Tree can completely filter out LT_{CCubes} that have gc smaller than or equal to L_{gc} .

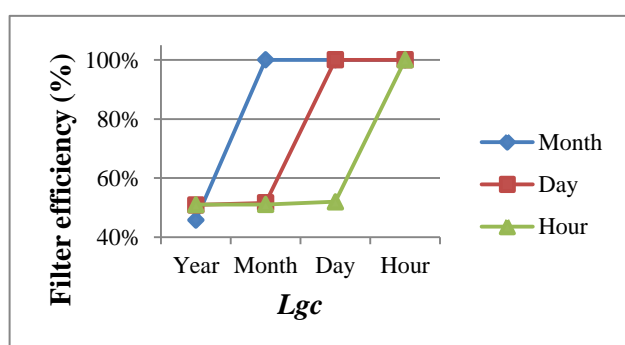
As mentioned previously, although a large number of LT_{CCubes} seldom occur in reality, we simulated large numbers of LT_{CCubes} to demonstrate that LOST-Tree can filter out LT_{CCubes} under any scenario. As a result, filtering out the large number of LT_{CCubes} results in the increases of the query latency and number of requests. For example, in the case of configuring L_{gc} as an hour, the average latency grows from 3 milliseconds to 185 milliseconds (Figure 4.9(d)) and 343 requests are required (Figure 4.9(f)). Therefore, if consider all possible scenarios, we would make a theoretical suggestion to configure L_q as the day level in order to avoid large query latency and a large number of requests and still attain a filter efficiency more than 80%.



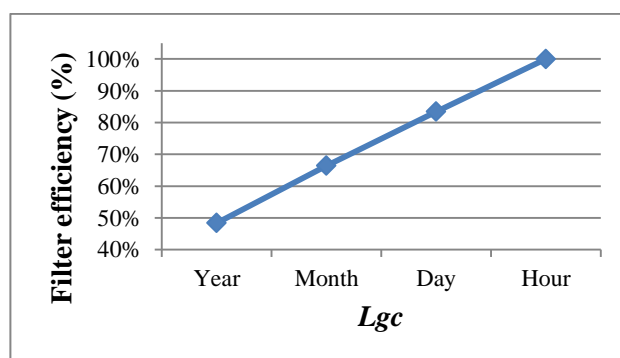
(a)



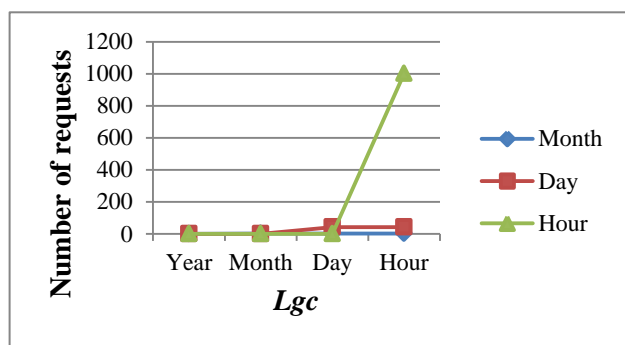
(d)



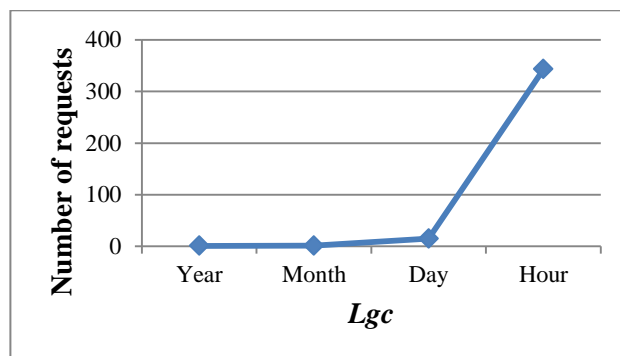
(b)



(e)



(c)



(f)

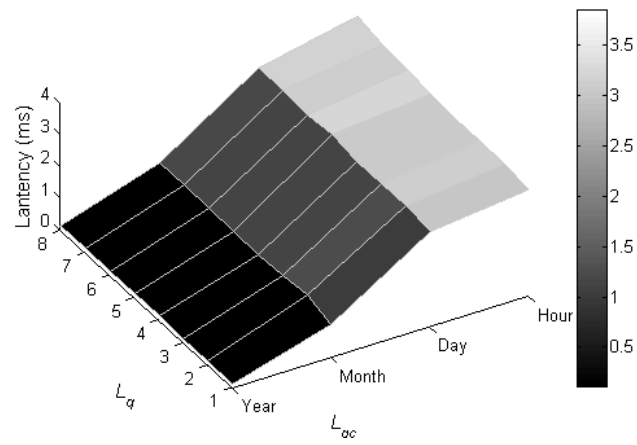
Figure 4.9 LOST-Tree performance on different L_{gc} : (a) query latencies for different levels of q ; (b) filter efficiencies for different levels of q ; (c) number of requests for different levels of q ; (d) average query latencies; (e) average filter efficiencies; (f) average number of requests.

4.2.3.3 Evaluation of LOST-Tree performance on different combinations of L_q and L_{gc}

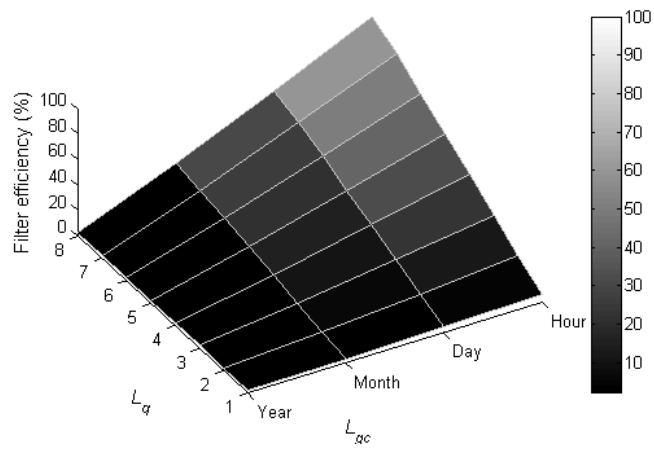
The previous two evaluations focused on showing the LOST-Tree performance on different L_q and L_{gc} in any possible scenario (*i.e.*, from small to large numbers of LT_{CCubes}). In this evaluation, we aim to demonstrate the expected LOST-Tree performance in a real-world sensor web browser with a more realistic scenario (as shown in Table 4.2). We examine LOST-Tree performance with different combinations of L_q and L_{gc} to show a comprehensive evaluation. The evaluation results are shown in Figure 4.10 and Tables 4.3, 4.4 and 4.5.

Based on the evaluation results, we have the following findings. First, in the case of a more realistic scenario, LOST-Tree can efficiently filter out LT_{CCubes} in less than 4 milliseconds (Figure 4.10(a) and Table 4.3). Second, as shown in Figure 4.10(b) and Table 4.4, LOST-Tree can eliminate all unnecessary requests when L_q and L_{gc} are equal to or larger than the smallest q and gc of LT_{CCubes} (*i.e.*, level 8 and the hour level, respectively).

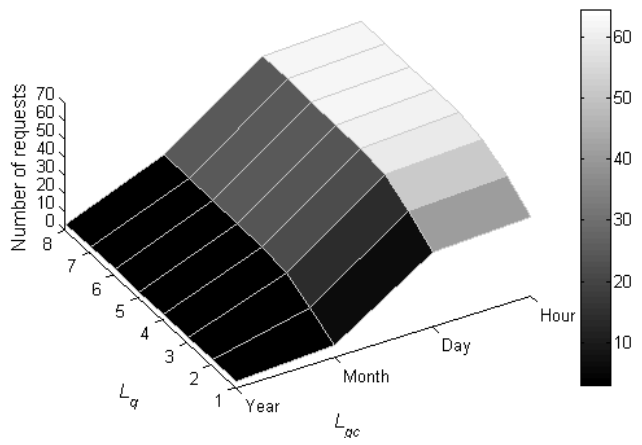
Third, L_q and L_{gc} allow LOST-Tree to control the trade-off between filter efficiency and number of requests for different kinds of sensor web servers (Figures 4.10(b) and Figure 4.10(c) and Tables 4.4 and 4.5). For instance, if a server accepts multiple spatial and temporal extents in a single request, users can configure larger L_q and L_{gc} settings to filter out all unnecessary requests. However, if a server allows only one spatio-temporal cube per request, users have two choices: (1) still try to filter out all unnecessary transmissions as long as the number of LT_{CCubes} is small (*e.g.*, as shown in our evaluation result, 65 requests are still manageable), or (2) modify L_q and L_{gc} settings to trade filter efficiency for a smaller number of requests.



(a)



(b)



(c)

Figure 4.10 LOST-Tree performance on different combinations of L_q and L_{gc} : (a) query latencies; (b) filter efficiencies; (c) number of requests.

Table 4.3 Query latencies with different combinations of L_q and L_{gc} (unit: millisecond)

| $L_{gc} \backslash L_q$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------------|------|------|------|------|------|------|------|------|
| Year | 0.11 | 0.11 | 0.12 | 0.12 | 0.12 | 0.12 | 0.12 | 0.12 |
| Month | 1.03 | 1.25 | 1.15 | 1.12 | 1.14 | 1.15 | 1.18 | 1.18 |
| Day | 2.99 | 3.10 | 3.08 | 3.07 | 3.21 | 3.14 | 3.18 | 3.18 |
| Hour | 3.36 | 3.47 | 3.49 | 3.48 | 3.54 | 3.62 | 3.66 | 3.84 |

Table 4.4 Filter efficiencies with different combinations of L_q and L_{gc}

| $L_{gc} \backslash L_q$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------------|------|-------|-------|-------|-------|-------|-------|--------|
| Year | 2.4% | 2.3% | 2.3% | 2.3% | 2.3% | 2.3% | 2.3% | 2.3% |
| Month | 3.1% | 6.7% | 11.2% | 16.0% | 20.7% | 25.4% | 29.9% | 33.4% |
| Day | 4.7% | 13.0% | 22.4% | 31.8% | 41.3% | 50.7% | 59.6% | 66.7% |
| Hour | 6.3% | 19.3% | 33.5% | 47.6% | 61.8% | 75.9% | 89.3% | 100.0% |

Table 4.5 Number of requests with different combinations of L_q and L_{gc}

| $L_{gc} \backslash L_q$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-------------------------|------|------|------|------|------|------|------|------|
| Year | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 | 3.0 |
| Month | 6.4 | 14.4 | 21.4 | 23.9 | 24.3 | 24.4 | 24.4 | 24.5 |
| Day | 41.4 | 51.7 | 59.0 | 61.2 | 61.8 | 62.1 | 62.1 | 62.0 |
| Hour | 44.0 | 54.3 | 61.7 | 64.0 | 64.5 | 64.6 | 64.6 | 64.6 |

4.2.4 Summary

In summary, we evaluated and compared the end-to-end data loading latencies with and without applying LOST-Tree approach in Chapter 4.2.1. The result shows that the proposed solution is at least 100 times faster than the naïve solution. In Chapter 4.2.2, we measured the changes of LOST-Tree size by testing spatial, temporal, and spatio-temporal aggregations. The

evaluation shows that LOST-Tree is lightweight (always less than 164 Kbytes during our evaluation) and scalable in terms of the number of spatio-temporal requests. In addition, since LOST-Tree only manages the spatio-temporal extents of requests, the increasing amount of sensor data does not affect the LOST-Tree size.

Chapter 4.2.3 evaluates LOST-Tree performance when applying different L_q and L_{gc} settings. The evaluation results show that LOST-Tree can avoid unnecessary transmissions up to 100% in an efficient manner (less than 4 milliseconds for the realistic scenario). In addition, the evaluation results also show that users are able to control the trade-off between filter efficiency and number of requests by setting the L_q and L_{gc} for different kinds of sensor web services.

4.3 Evaluation on the AHS-Model

We evaluate the proposed AHS-Model from four perspectives. First, since the major objective of AHS-Model is to efficiently process topological operators when handling a large number of geometries, we analyze the scalability of AHS-Model in terms of the number of queries/subscriptions by comparing with PostGIS, which we used to represent traditional topological operators.

The second evaluation is for measuring the indexing latency. Since AHS-Model approximates geometries with a quadtree tile system, the indexing for large geometries may be time-consuming. Although we argue that the subscriptions and data in the context of sensor web would not have large geographical coverage, we evaluate the indexing latency of AHS-Model by simulating geometries in various sizes to be comprehensive.

The third evaluation analyzes the matching latency of AHS-Model. To be more specific, this evaluation measures the latency of matching AHS_{PUB} and AHS_{SUB} . Similar to the second

evaluation, we examine with geometries simulated in various sizes to provide a comprehensive evaluation.

Finally, the forth evaluation is the end-to-end performance analysis measuring the latencies of overhead and each of the following steps: (1) index publication, (2) match AHS_{PUB} with AHS_{SUB} , and (3) determine relationship. This evaluation examines all possible relationships between two geometries (the relationships in Table 3.2). The testing data for this evaluation are city-level subscriptions and publications that were manually simulated to be more realistic and provide the expected AHS-Model performance in a real-world application.

4.3.1 AHS-Model scalability evaluation

One of the most important design decisions of AHS-Model is to perform topological operators in an aggregated manner. By aggregating the indices from all subscriptions in a single structure (*i.e.*, AHS_{SUB}) and decoupling the indices and subscriptions, AHS-Model can match new data with all subscription in a single process. In this case, subscriptions that have quadkeys in common can benefit from this design.

Therefore, in order to demonstrate this contribution, we evaluate the scalability of AHS-Model in this section. To be more specific, we measure the query performance while register different number of subscriptions into AHS-Model. Here we choose the point-in-polygon query as our testing case, as point-in-polygon is one of the most common queries. We simulated a subscription with the coverage of a city (*i.e.*, a polygon) and assign *WITHIN* as the topological operator in the spatial predicate. Then we simulated a publication with a point geometry that locates in the city. With the simulated subscription and publication, we register different numbers of subscriptions into the AHS-Model (with different subscription identifier SUB_{ID}) and

measure the query latency every 500 additional subscriptions by sending the publication to the AHS-Model. The same test was performed on an untuned PostGIS database as a comparison.

Since this evaluation is mainly about the scalability in terms of the number of subscriptions, this evaluation was performed on a single machine in order to avoid communication overhead and machine heterogeneity. This evaluation was performed on a desktop-class machine, which runs an Intel[®] Core™ i5-650 @ 3.20GHz, 6GB RAM, and Western Digital WD10EARS-22Y5B1.

The query latencies on different number of subscriptions are shown in Figure 4.11. Based on these experimental results, we observe that the query latency increases with the number of subscriptions for both PostGIS database and AHS-Model. However, as the query latency of AHS-Model increases 2.5 times slower than that of an untuned PostGIS, this shows that AHS-Model is more scalable than the traditional solution in terms of the number of subscriptions.

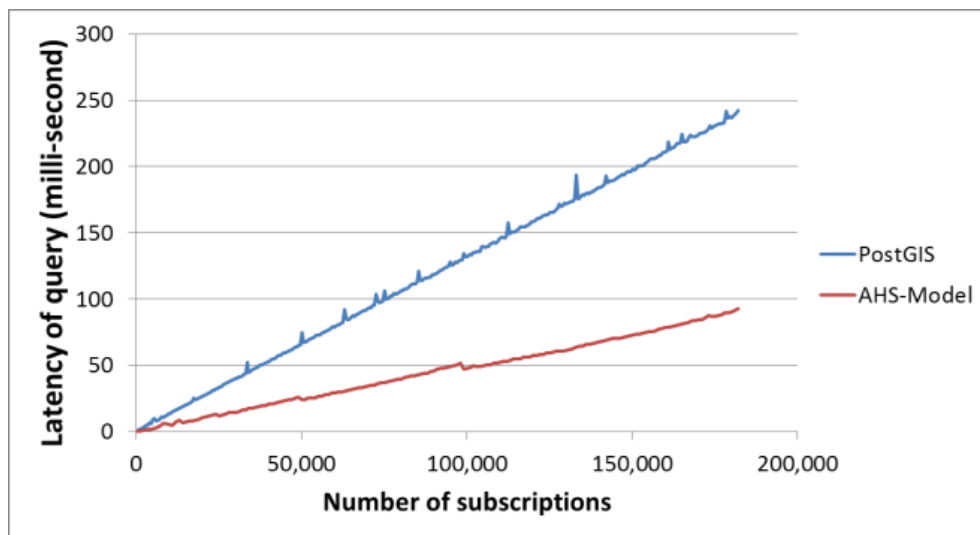


Figure 4.11 The AHS-Model query latency on different number of subscriptions.

4.3.2 Evaluation of AHS-Model indexing performance

This section measures the latency of generating necessary quadkeys from the geometry of subscription (as shown in Table 3.3). Since the time cost for indexing may differ based on the size of geometry, we randomly generate geometries in different sizes and measure the latencies for indexing them. In addition, as mentioned earlier, a lowest level of quadtree tile is needed as the granularity on geometry approximation. The quadtree tile system used in this evaluation has 14 levels.

In addition, as mentioned in Chapter 3.6.4 that distributed AHS-Model can process the indexing tasks in parallel, we also measure the indexing latency when using different numbers of workers. However, since we do not have a large number of machines to perform the actual test, we used a machine to simulate each worker handling different quadkeys in the distributed AHS-Model. Here we simulate scenarios of distributed AHS-Models with 1, 4, 16, 64, and 256 workers. While the 1-worker scenario is basically the stand-alone AHS-Model, each worker in 4-, 16-, 64-, and 256-workers scenarios handles a quadkey in the first, second, third, and fourth level of quadtree, respectively. For example, for the 4-workers scenario, we simulated four workers handling quadkey '0', '1', '2', and '3'. Moreover, as for distributed computing processes, the entire process is considered completed at the time that the last worker finishes its task, here we present the maximum (instead of average) indexing latency from workers in each scenario.

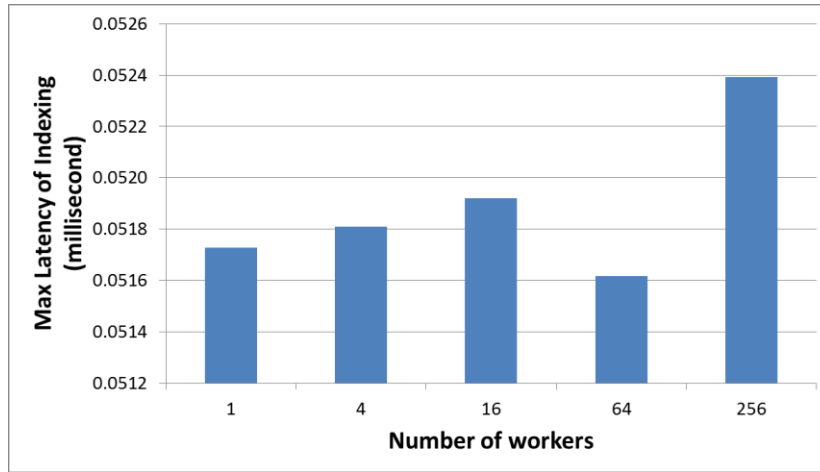
The indexing latency of point, line, and polygon geometries are shown in Figure 4.12. Since the sizes of point geometries are the same (*i.e.*, one area unit), we take the average for each scenario. In general, while the indexing latency for point geometry is much smaller than that for other types of geometries, the indexing latency for line geometry is smaller than that for polygon

geometry. We believe this is because of the different number of quadkeys being indexed, which is related to the size and location of geometries.

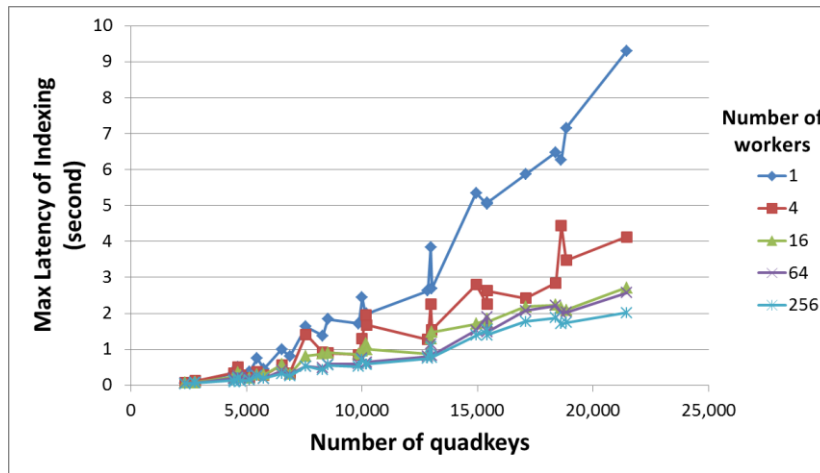
By comparing the indexing latencies based on different numbers of workers, we can observe that the performance can be significantly improved by using more workers in the distributed AHS-Model. Our evaluation results show that the indexing process of using 256 workers can be 5 to 10 times faster than the stand-alone indexing process.

Finally, while this evaluation tests with simulated geometries in various sizes to be comprehensive, some of these geometries are too large in the context of sensor web. Among these simulated geometries, the longest line geometry we generated was 7,112 kilometer long; and the largest polygon geometry was about 37% of Earth's coverage. However, in reality, a major city highway is usually about 100 kilometer; and a city's coverage is usually smaller than 1% of Earth's coverage.

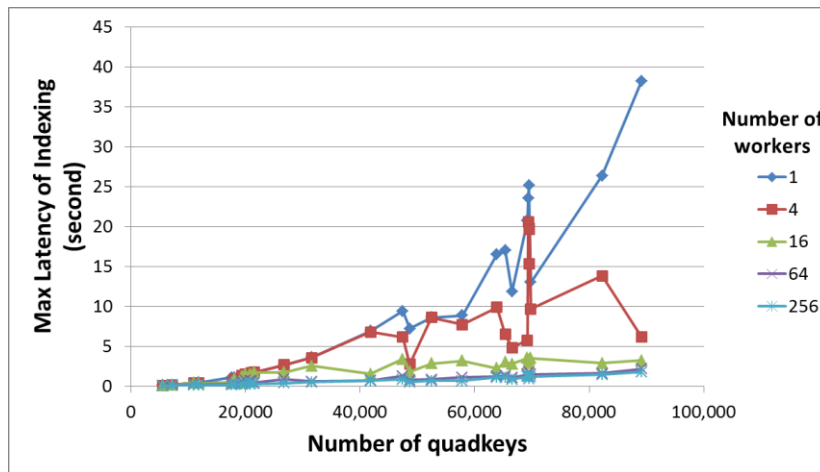
To sum up, while some of the simulated geometries are not realistic, the AHS-Model is able to finish the indexing step in a timely manner with the help of distributed processing. For indexing subscriptions, considering the long-running nature of continuous query, we argue that the measured indexing overhead is acceptable. In addition, as real-world sensor web data usually have much smaller geospatial coverage than the simulated geometries, we believe the indexing for publications would be much faster than that for subscriptions. The evaluation of the AHS-Model performance using a more realistic dataset is presented in Chapter 4.3.4.



(a)



(b)



(c)

Figure 4.12 The indexing latency for (a) point, (b) line, and (c) polygon geometry.

4.3.3 Evaluation of AHS-Model matching performance

This section measures the latency of matching AHS_{PUB} with AHS_{SUB} . Since the time cost for matching may differ based on the number of quadkeys, we randomly generate geometries in various sizes. And in order to make sure that the quadkeys of these geometries will be processed, we first applied the same geometry in both publication and subscription, and then assigned *EQUALS* as the topological operator. In this evaluation, the quadtree tile system had 14 levels.

In addition, similar to the previous evaluation, we also measured the matching latency when applying different numbers of workers. In this evaluation, we still used the same machine to simulate each worker handling different quadkeys in the distributed AHS-Model. Here we simulated scenarios of distributed AHS-Models with 1, 4, 16, 64, and 256 workers. And considering the nature of distributed computing processes, we present the maximum (instead of average) matching latency from workers in each scenario.

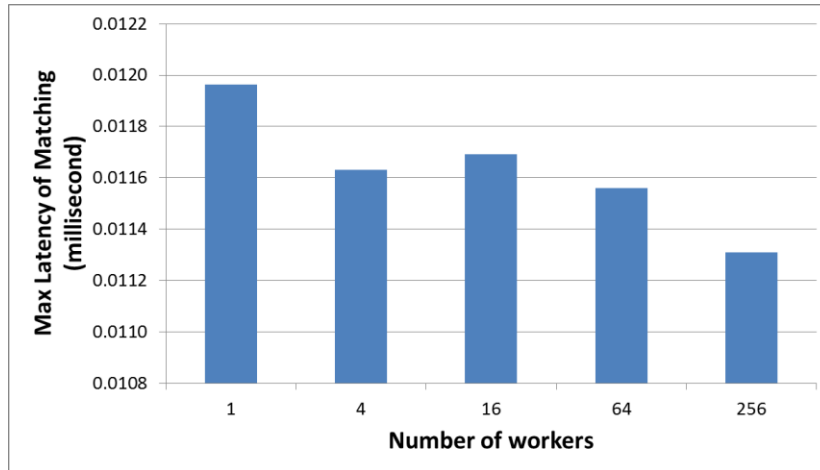
The matching latency of point, line, and polygon geometries are shown in Figure 4.13. Since the sizes of point geometries are the same (*i.e.*, one area unit), we take the average on each scenario. In general, while the matching latency for point geometry is much smaller than that for other types of geometries, the matching latency for line geometry is smaller than that for polygon geometry. We believe this is because of the different number of quadkeys being processed.

By comparing the indexing latencies based on different number of workers, we can observe that the performance can be significantly improved by using more workers in the distributed AHS-Model. Our evaluation results show that the matching process of using 256 workers can be 20 to 300 times faster than the stand-alone indexing process.

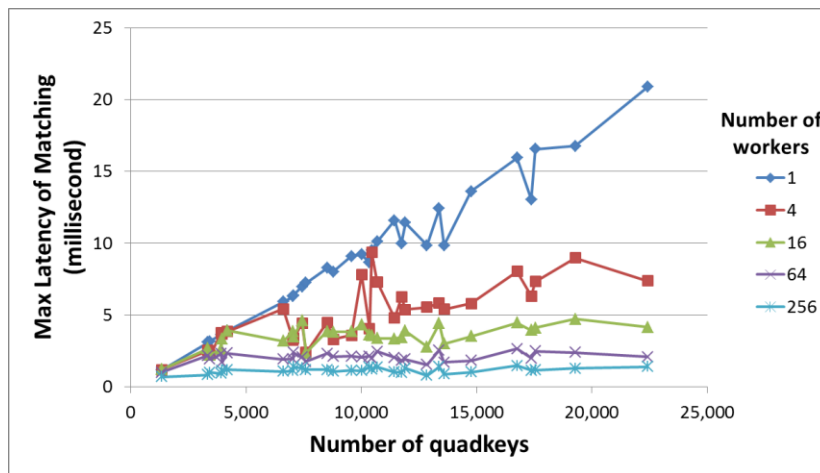
Finally, similar to the previous evaluation, this evaluation tests with simulated geometries in various sizes to be comprehensive. However, some of these geometries may be too large in the

context of sensor web. For example, among these simulated geometries, the longest line geometry we generated was 7,778 kilometer long; and the largest polygon geometry was about 20% of Earth's coverage. However, in reality, a major city highway is usually about 100 kilometer; and a city's is usually smaller than 1% of Earth's coverage.

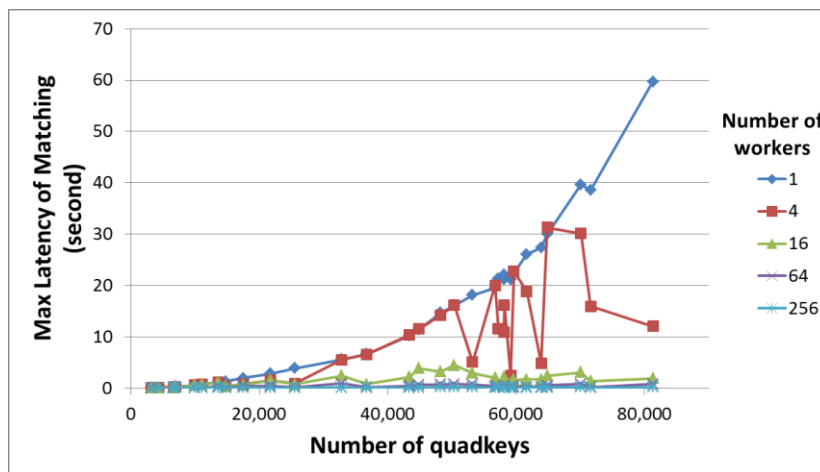
To sum up, while some of the simulated geometries are not realistic, AHS-Model is able to match AHS_{PUB} and AHS_{SUB} in a timely manner with the help of distributed processing. In Chapter 4.3.4, we present the evaluation of AHS-Model performance by using a more realistic dataset.



(a)



(b)



(c)

Figure 4.13 The matching latency for (a) point, (b) line, and (c) polygon geometry.

4.3.4 Evaluation of AHS-Model end-to-end query performance

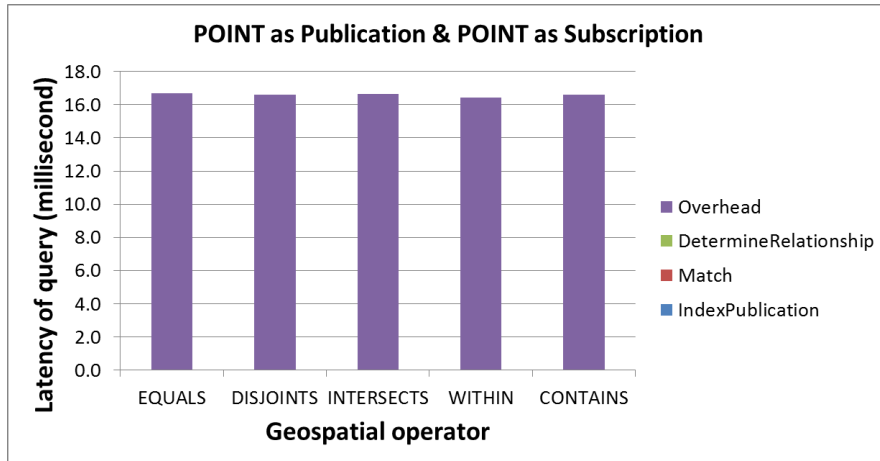
In this section, we evaluate the end-to-end query performance of the AHS-Model. We measure the latency of indexing publication, matching AHS_{PUB} and AHS_{SUB} , determining relationship, as well as the overhead. In addition, we perform this evaluation on any possible relationships (Table 3.2). We simulate one subscription/publication pair for each possible relationship. The sizes of geometries are created on city level as it may be the most common size in many use cases. For example, we create the point, line, and polygon of subscription based on the ideas of a point in a city (e.g., a city landmark), a road crossing a city (e.g., a major highway), and the coverage of a city, respectively. After creating the subscriptions, we manually create publications that match subscriptions for each possible topological relationship (e.g., a sensor locates at a road intersection).

In order to test the overhead of distributed computing, we used two machines in this evaluation. The sets of quadkeys each worker is in charge of are manually configured, so that the workers handle similar amounts of work. Both machines are desktop-class machines. One of them runs an Intel® Core™ i5-650 @ 3.20GHz, and 6GB RAM; and the other has Intel® Core™ i7-3770 @ 3.40GHz, and 10GB RAM. Considering the machine heterogeneity and the possible unequal amount of work assigned, instead of presenting the maximum latencies, this evaluation calculates the average latencies to provide an expected AHS-Model performance in a real-world application.

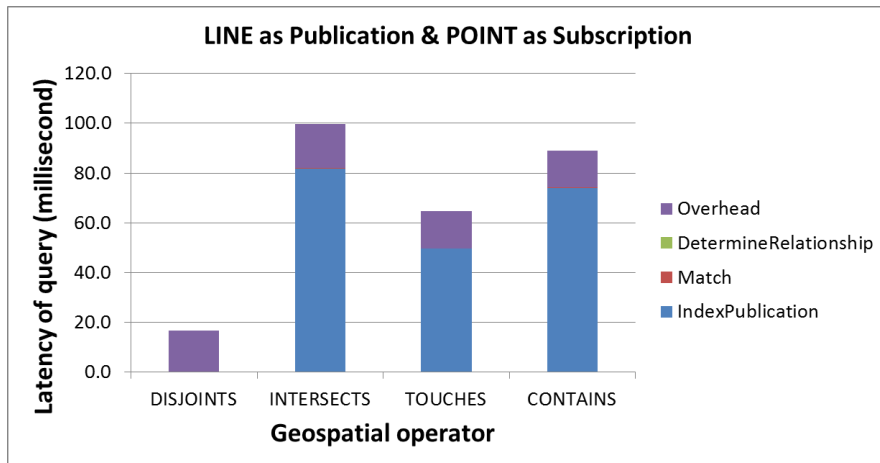
Therefore, we test each scenario 10 times and take the average for each scenario. The end-to-end query performances for using point, line, and polygon geometry as subscription are shown in Figure 4.14, Figure 4.15, and Figure 4.16, respectively. Based on these evaluation results, we found that it is difficult to simulate datasets that are fair enough to be used in the

comparison of different topological operators. We argue that the performance differences between topological operators do not hold much meaning as those differences may come from the machine heterogeneity or the characteristics of dataset such as the geometry size. Yet, this evaluation is still valuable as it measures the overhead of distributed computing, presents the latency of each step, and shows the expected AHS-Model performance in a real-world sensor web application.

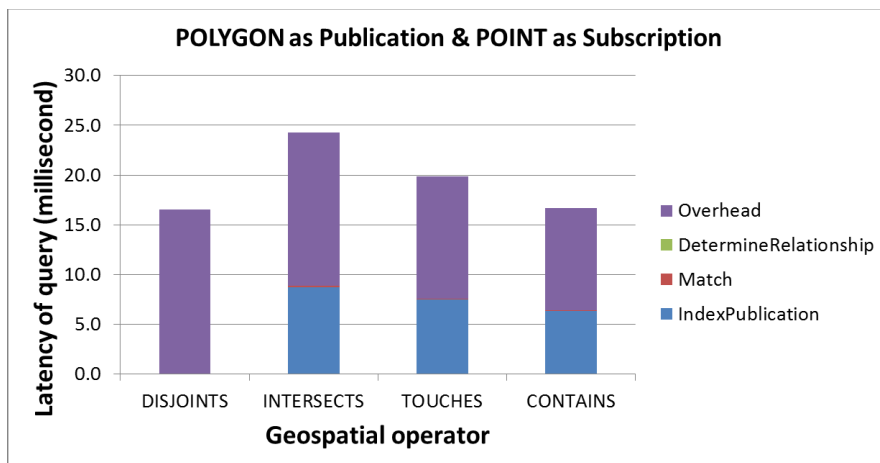
Therefore, based on the experimental results, our first observation is that the indexing and overhead take more than 99% of the end-to-end latency. While the overhead of applying distributed computing process is relatively stable (between 10 to 30 milliseconds), the indexing latency varies largely depending on the geometry size of publications. Our second observation is that the latencies for determining relationships are very small as each determination only handles a two-by-two matrix. Finally, since this evaluation is based on a more realistic dataset, the measured performance is able to represent the expected AHS-Model performance in a real-world application. As we can see from the evaluation results, most of the tests can be finished in 100 milliseconds while more than 70% of them can be completed in 50 milliseconds. Therefore, we believe that AHS-Model can efficiently process any possible topological operators on sensor web data, which is critical for time-sensitive applications.



(a)

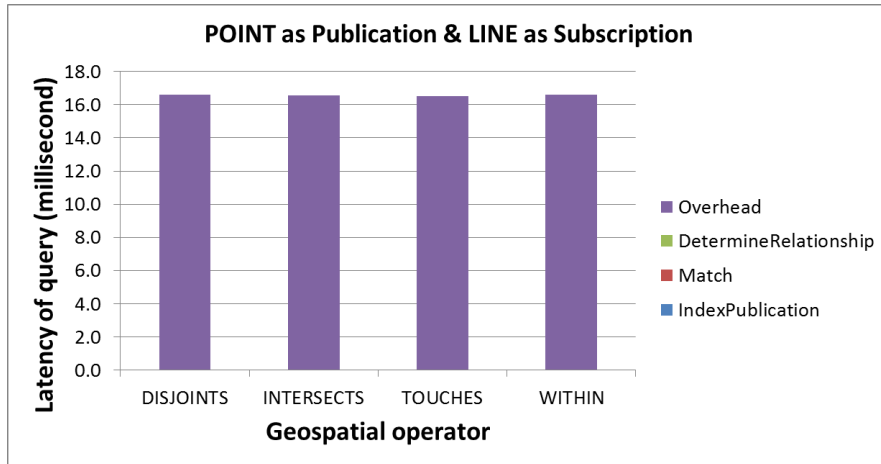


(b)

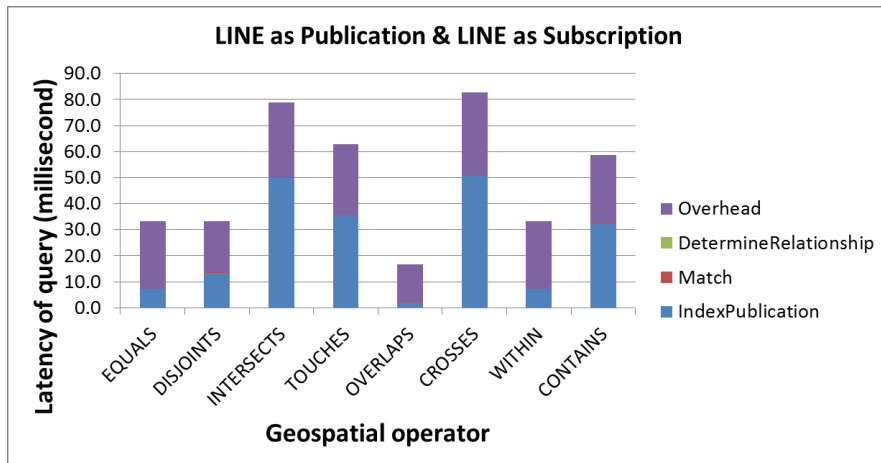


(c)

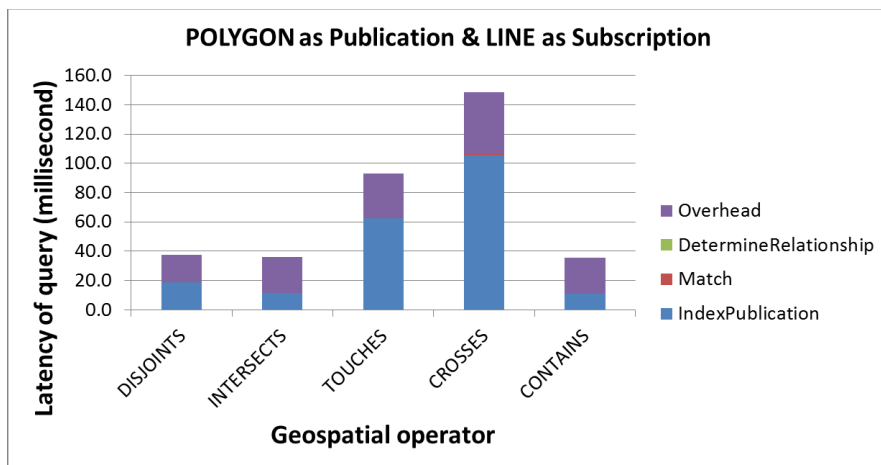
Figure 4.14. The end-to-end query performance for point as subscription and (a) point, (b) line, and (c) polygon geometry as publication.



(a)

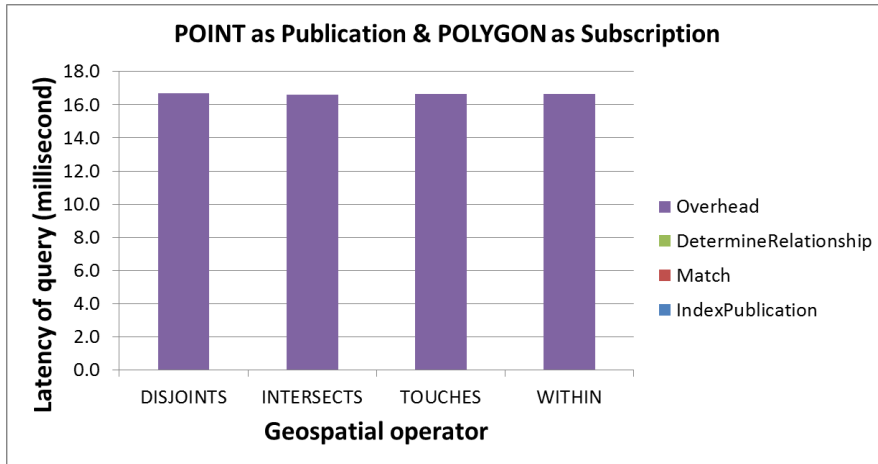


(b)

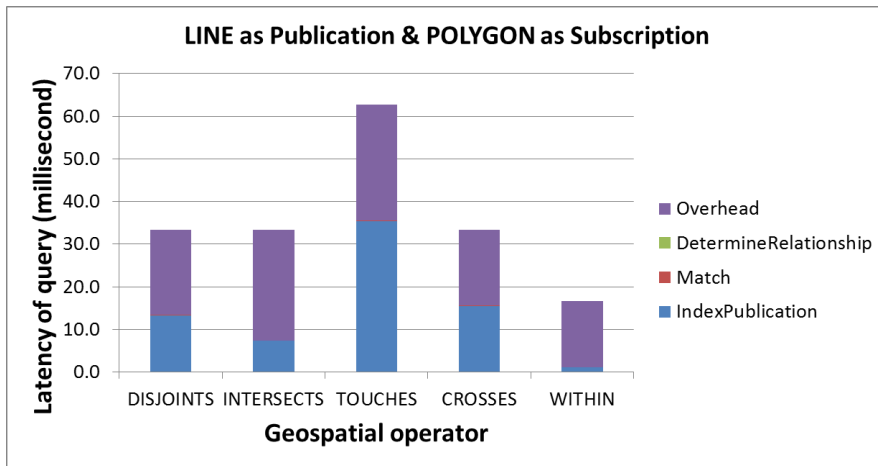


(c)

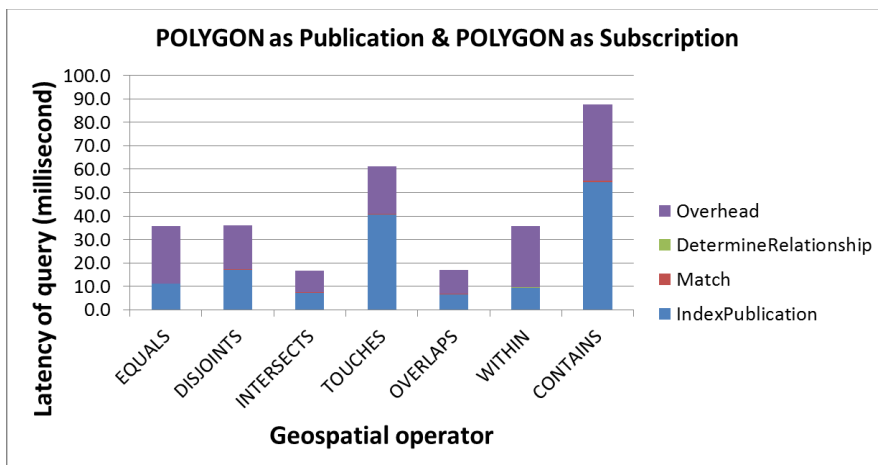
Figure 4.15. The end-to-end query performance for line as subscription and (a) point, (b) line, and (c) polygon geometry as publication.



(a)



(b)



(c)

Figure 4.16. The end-to-end query performance for polygon as subscription and (a) point, (b) line, and (c) polygon geometry as publication.

4.3.5 Summary

In summary, we evaluated and compared the scalability between AHS-Model and PostGIS in terms of the number of subscriptions in Chapter 4.3.1. The result shows that AHS-Model is 2.5 times faster than PostGIS, which indicates that the proposed solution is more scalable than the traditional solution. In Chapter 4.3.2, we simulated different number of workers in the distributed AHS-Model to measure the performance of indexing geometries in various sizes. The evaluation shows that with the help of distributed processing, AHS-Model is able to finish the indexing step in a timely manner even for geometries in large size. In addition, the indexing process of using 256 workers can perform 5 to 10 times faster than the stand-alone indexing process.

Chapter 4.3.3 evaluates the matching performance by simulating geometries in different sizes. The evaluation results show that the distributed computing process can significantly improve the matching performance by 20 to 300 times. And even for large geometries, AHS-Model is able to match AHS_{PUB} and AHS_{SUB} efficiently with the help of distributed processing.

Finally, we evaluated the end-to-end query latency with more realistic datasets in Chapter 4.3.4. We observed that indexing and overhead take more than 99% of the end-to-end latency. While the overhead of applying distributed computing process is relatively stable (between 10 to 30 milliseconds), the indexing latency varies largely depending on the geometry size of publications. As demonstrated earlier that AHS-Model can finish most queries in 100 milliseconds for a more realistic dataset, we believe that AHS-Model is able to efficiently process topological operators in a geospatial sensor web publish/subscribe system.

Chapter Five: **Related Works**

This thesis has two major contributions. The first contribution is more related to the high-level system architecture of a geospatial sensor web publish/subscribe system. We identify the potential challenges and propose an overall system architecture. The second contribution is about the new solutions proposed for selected components that we believe are critical and unique in the context of a sensor web publish/subscribe system.

Therefore, we divide the discussion of related works into two parts accordingly to these contributions. In Chapter 5.1, we present the related system architecture and their mechanisms. And the Chapter 5.2 focuses on the works related to the new solutions proposed to address geospatial sensor web challenges.

5.1 Related systems and their mechanisms

In Chapter 2.1, we have discussed the major issues and existing approaches in general-purpose systems applying the publish/subscribe communication model. As mentioned earlier, publish/subscribe system (Eugster et al. 2003), simple event processing system (Michelson 2006), DSMS (Babcock 2002; Golab and Ozsu 2003; Cugola and Margara 2010), and CEP system (Luckham 2002) are similar as they all apply the continuous query processing model. Although the original designs of these systems are different in terms of the targeted data type and query complexity, their functionalities become similar to each other as these systems are evolving. In this chapter, we first briefly introduce the original designs of these systems and then explain their current architectures and mechanisms.

Based on original designs of these systems, we can differentiate them by the degree of query complexity they handle. In general, publish/subscribe systems handled the simplest queries and simple event processing added some simple filtering functionalities on the basic

publish/subscribe communication model. While publish/subscribe and simple event processing systems focused on the processing of individual data points, DSMS and CEP tried to handle multiple data streams.

In Dr. Luckham's blog post²⁰ "*What's the Difference between ESP and CEP?*" he mentioned that ESP (*i.e.*, event stream processing, which is similar to DSMS) was designed in the database community for real-time data analysis while CEP was designed to model not only the timing of events but also the relationship between events (*i.e.*, pattern). He also mentioned that the fundamental difference between DSMS and CEP is at the types of data stream they dealt with. DSMS handled data that are ordered by time, which allowed DSMS to process events with very little memory since they do not need to cache many data. On the other hand, CEP aimed on processing data that may not be perfectly ordered. CEP then needed to cache many data before discovering the relationship between them.

In general, DSMS was focused on high-speed querying and processing ordered data while CEP was focused more on discovering patterns from not-perfectly-ordered data and extracting information from the patterns. Moreover, at the end of the blog post, Dr. Luckham concluded that there will be no difference between DSMS and CEP in the future.

Therefore, although the original designs of these systems are different, their functionalities overlap as they are evolving. For example, publish/subscribe systems start to support more filtering functions and DSMS starts merging with CEP. In the following two sections, in order to provide enough background for systems that are related to a geospatial

²⁰ Dr. David Luckham, the originator of CEP as proposed in his book, "The Power of Events" published in 2002. He posted this article in a CEP blog: <http://www.complexevents.com/2006/08/01/what%E2%80%99s-the-difference-between-esp-and-cep/>

sensor web publish/subscribe system, we introduce the current high-level architectures and existing approaches of publish/subscribe systems and DSMSs.

5.1.1 Publish/Subscribe systems

Publish/Subscribe is a communication model decoupling publishers and subscribers. Subscribers first register their event of interest, and asynchronously get notifications of events generated by publishers. Unlike the point-to-point synchronous request/response communication model, the asynchronous publish/subscribe model is more suitable for large-scale distributed applications. For example, publish/subscribe model has been widely applied in web blogging with RSS²¹ (RDF Site Summary) and Atom²² technologies. Eugster et al. (2003) wrote a well-cited summary paper about publish/subscribe systems. Based on Eugster's paper, here we introduce publish/subscribe systems from four perspectives, namely (1) architecture; (2) decoupling type; (3) filtering type; and (4) dissemination approach.

1. *Architecture*: The basic publish/subscribe architecture relies on an intermediary component managing subscriptions and events. This intermediary (named as *message broker* or *event notification service*) provides three basic functions, namely storing, filtering, and notifying. First, the intermediary stores the subscriptions from subscribers. Second, when publishers send events to the intermediary, it filters the events based on the criteria in subscriptions. Finally, after discovering events that match subscribers' criteria, the intermediary notifies subscribers of these events. The workflow of this basic publish/subscribe architecture is shown in Figure 1.3.

²¹ RSS 2.0 Specification (<http://www.rssboard.org/rss-specification>)

²² Atom wiki (<http://www.intertwingly.net/wiki/pie/FrontPage>)

2. *Decoupling type*: There are three types of decoupling between publishers and subscribers that publish/subscribe model provides, namely *space decoupling*, *time decoupling*, and *synchronization decoupling*. First, the space decoupling means publishers and subscribers do not know each other. Both publishers and subscribers only communicate with the intermediary. Second, the time decoupling means publishers and subscribers can interact with the intermediary at different time. For example, when a publisher publishes an event, a subscriber could be offline; and similarly, a subscriber could be notified about an event when the publisher of that event is offline. Third, the synchronization decoupling means that both the publishing events and receiving notifications activities can occur while publishers and subscribers perform other concurrent tasks. As stated in Eugster et al. (2003), a publish/subscribe system should provide all these three types of decoupling.

3. *Filtering type*: Publish/Subscribe systems are usually categorized based on how they filter events. Besides two common filtering types: *topic-based* and *content-based*, some systems support a *hybrid* of these two. The earliest publish/subscribe models were all topic-based, such as Birman et al. (1987), Powell (1996), Skeen (1998), and TIBCO (1999). In topic-based publish/subscribe systems, publishers publish events to “topics” as logical channels. Then subscribers receive notifications of events published to the topics they subscribe. However, topic-based publish/subscribe systems are static in terms of the topics they provide. Subscribers cannot express a customized subscription in topic-based systems.

On the other hand, content-based publish/subscribe systems allow subscribers to specify filters based on the content of events. With the expressiveness of content-based systems, subscribers can customize subscriptions based on their own interests. Some systems even

support complex subscription patterns or event correlation (Bacon et al. 2000). Siena (Carzaniga et al. 2001), JEDI (Cugola et al. 2001), and Hermes (Pietzuch 2004) are examples of content-based publish/subscribe systems.

Other than topic-based and content-based systems, a hybrid publish/subscribe system allows publishers to publish events to predefined topics while subscribers treat topics as another attribute of events (Szarowski 2003). Based on this definition, our proposed GeoPubSubHub belongs to a hybrid publish/subscribe system as we use SOS property layers or phenomenon elements in the semantic layer service as topics. Another hybrid publish/subscribe example is the OGC Sensor Event Service (SES) (Echterhoff and Everding 2008). SES has three levels of filtering. The level 1 filter uses XPath²³ on single event. The level 2 filter applies logical, spatial, temporal, arithmetic, and comparison operators on incoming events. And the level 3 filter employs the OGC Event Pattern Markup Language (EML) (Everding and Echterhoff 2008) to perform filtering on event streams.

4. *Dissemination approach*: There are various approaches for transmitting events in publish/subscribe systems. These approaches can be categorized into centralized and decentralized approaches. In a centralized approach, the intermediary (*i.e.*, the event notification service) directly communicates with publishers and subscribers. While the centralized approaches provide strong guarantee of transmission, their throughput and scalability are not as good as that of decentralized approaches. The most common decentralized approach is Internet Protocol (IP) multicast (Deering 1989), which is a

²³ <http://www.w3.org/TR/xpath/>

technique for one-to-many communication. With IP multicast, the data source (*i.e.*, the event notification service) only needs to send packets once to the network infrastructure, and the network will handle the packet replication and the transmission to multiple receivers (*i.e.*, subscribers).

In topic-based systems, subscribers can be grouped by the topics they subscribe. IP multicast dissemination approaches can be easily applied for high-throughput (Floyd et al. 1997; Castro et al. 2002). However, an efficient dissemination approach for content-based system remains an issue since subscribers are hard to be grouped with their ad-hoc subscriptions.

Some works construct an overlay network of event notification servers and only transmit events to the servers that manage subscriptions related to these events (Aguilera et al. 1999; Carzaniga et al. 2001). However, the performance of dissemination will strongly depend on the efficiency of filtering processes on each server. Therefore, there have been some researches about efficient filtering in publish/subscribe systems, such as Fabret et al. (2001), Campailla et al. (2001), and Diao et al. (2002). Similar to DSMS, publish/subscribe generates query plans from continuous queries and aggregates identical or similar query operators for efficiency.

In GeoPubSubHub, we have not discussed the dissemination approach yet. Currently, we propose using a centralized approach for a more reliable communication. One of our future directions is to analyze whether other dissemination approaches can improve the performance of GeoPubSubHub.

5.1.2 Data stream management systems (DSMSs)

As many data-intensive applications emerging (e.g. network monitoring, electronic trading, and sensor network monitoring), traditional DBMSs are not designed to query rapid and continuous data streams. Therefore, the database community proposed DSMS (or called ESP) to efficiently query real-time streaming data. Here we compare DBMS and DSMS from three perspectives, namely data, processing model, and query. First, while the data in DBMS usually represent persistent relations, data in DSMS could be removed for processing and storage efficiency. In addition, unlike the data in DBMS, the data streams in DSMS arrive in real-time, are potentially unbounded, and will be processed only once.

Second, the processing model of DBMS is query-driven while that of DSMS is data-driven. In DBMS, users submit queries to *pull out* answers, and data are processed upon query submission. In contrast, the DSMS processing model is similar to the publish/subscribe model, where data are processed upon its arrival and results are then *pushed* to users. Third, the queries in DBMS are *one-time* queries while the queries in DSMS are *continuous* queries. Due to the nature of one-time queries, the query optimization in DBMS is based on each query. On the other hand, DSMS generates a query plan for each query and shares the same query operators between query plans in order to optimize the processing of multiple queries.

In section 2.1, we have discussed the two major issues (*i.e.*, memory and query efficiency) in DSMS and the existing approaches proposed to address these two issue. Here we introduce the basic DSMS architecture and the *continuous query language*.

1. *Architecture*: The basic DSMS architecture consists of seven modules, namely (1) input buffer and input monitor; (2) working storage; (3) summary storage; (4) static data storage; (5) continuous query processor; (6) query repository; and (7) output buffer. The

high-level architecture is shown in Figure 5.1. First, the input buffer receives input streams from different data sources and parses these streams for internal usages while the input monitor collects the statistics of streams (e.g., input rate) and regulates the input rate of streams by dropping data (*i.e.*, load shedding). Second, the working storage temporarily stores the recent portion of streams with sliding windows. Third, the summary storage contains the summarized information that will be used in the later operations (*i.e.*, synopses). Fourth, the static data storage holds the static information required for queries, such as metadata.

Fifth, the continuous query processor executes query plans generated from user queries. The continuous query processor is also responsible for optimizing query processing. In addition, in order to handle the dynamic nature of DSMS, the continuous query processor informs the input buffer and input monitor about the current CPU and memory usage. Then the input buffer performs load shedding according to the system statistics. Conversely, the continuous query processor also receives the statistics of streams from the input monitor in order to optimize query plans according to the current input stream environment. Sixth, the query repository allows users to submit or cancel their queries. Finally, the output buffer streams out the answers of continuous queries.

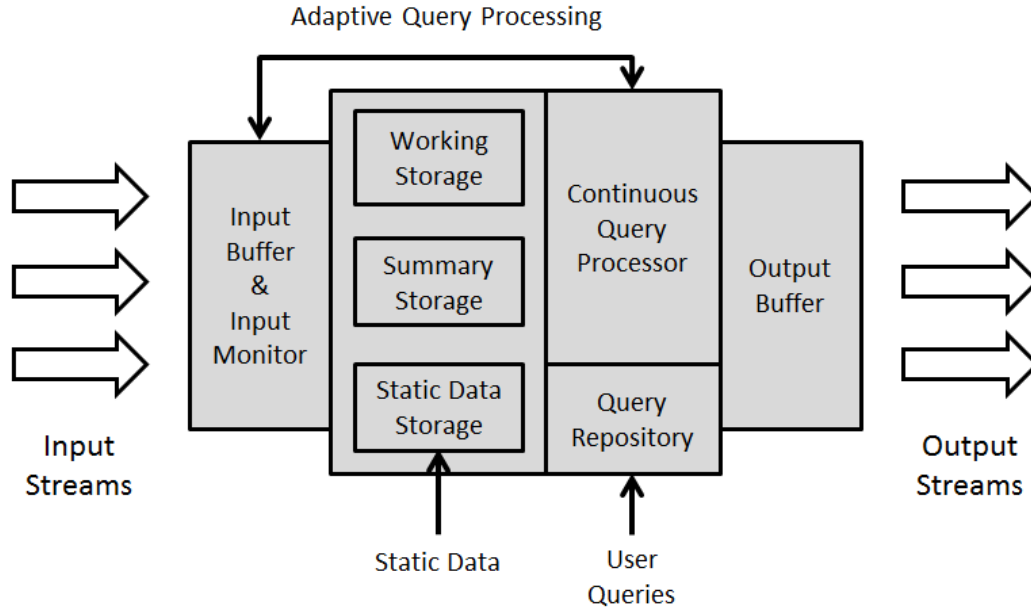


Figure 5.1 The basic DSMS architecture.

2. *Continuous query language*: The other important feature of DSMS we have not discussed is the continuous query language. Like DBMS applies structured query language (SQL) to query databases, DSMS community developed *continuous query languages* to query data streams. Various types of languages were proposed. Cugola and Margara (2010) and Golab and Ozsu (2003) categorized the existing continuous query languages based on the functionality of languages or the backend data structure. Since continuous query languages are usually extended from other standard query languages (e.g., SQL), here we try to group the existing continuous query languages based on those standard query languages, which include (1) *SQL-based continuous query language*, (2) *XML-based continuous query language*, (3) *procedural language*, and (4) *others*.

(1) *SQL-based continuous query language*: Since SQL is defined as the standard query language in DBMS, most DSMS systems developed continuous query languages based

on SQL. These languages construct additional operations on the basic SQL for sliding windows or order-dependent operations. For example, CQL (Continuous Query Language) in STREAM project (Arasu et al. 2004) defined three types of operators, namely *relation-to-relation*²⁴ operator, *stream-to-relation*²⁵ operator, and *relation-to-stream* operator.

First, the relation-to-relation operators are similar to the logical comparison functions in DBMS. Second, the stream-to-relation operators are essentially the sliding window operators, such as tuple-based sliding windows or time-based sliding windows. Third, the relation-to-stream operators contain an insertion of tuples in a relation into a stream, a deletion of tuples in a relation from a stream, and a conversion of tuples in a relation into stream with the current timestamp.

There are many other continuous query languages belong to this group, such as Fjords of TelegraphCQ (Madden and Franklin 2002; Chandrasekaran et al. 2003), AQuery (Lerner and Shasha 2003), COUGAR (Bonnet et al. 2001), PLACE (Mokbel et al. 2005), Tapestry (Terry et al. 1992), Alert (Schreier et al. 1991), Xyleme (Nguyen et al. 2001), OpenCQ (Liu et al. 1999), Gigascope (Cranor et al. 2003), Stream Mill (Bai et al. 2006), Cayuga (Brenna et al. 2007), NextCEP (Schultz-Moeller et al. 2009), SASE (Wu et al. 2006), StreamBase (2011), Oracle CEP (2009), Esper (2012), SPADE of IBM System S (Gedik et al. 2008), LINQ of Microsoft StreamInsight (2013).

(2) *XML-based continuous query language*: Some DSMSs are designed to retrieve information from data streams stored in the form of XML datasets. In this case, there

²⁴ In STREAM, *relation* is defined as a time-varying bag of tuples.

²⁵ In STREAM, *stream* is defined as an unbounded bag of tuples.

were continuous query languages developed based on *XML-QL*²⁶, a query language for XML, and *XPath*, a language for addressing parts of an XML document. Both XML-QL and XPath are W3C standards. For example, NiagaraCQ (Chen et al. 2000) allows users to periodically execute a XML-QL query by specifying a start time, an expiration time, and a time interval. On the other hand, XFilter (Altinel and Franklin 2000), a document filtering system, applied XPath language for users to express their queries. The XFilter's filter engine also indexes the XPaths for efficient query processing. In addition, the EML used in OGC Sensor Event Service is also based on XML (Everding and Echterhoff 2008).

(3) *Procedural language*: Other than the previous two declarative query languages, there are some DSMSs that allow users to specify the query flow with operators. For example, in Aurora (Abadi et al. 2003), users can construct query plans through a graphical interface, where operators and flows are represented as *boxes* and *arrows*. STREAM (Arasu et al. 2004), on the other hand, not only provides SQL-based continuous query language but also developed a *graphical query and system visualizer*, which allows users to view the structure of query plans, detail properties of each entity (*i.e.*, operator, queue, and synopsis), dynamically adjust entity properties, and view real-time monitoring graphs of properties. Similar to STREAM, Oracle CEP (2009) provides both a SQL-based query language and a visualizer for users to create, manage, and monitor CEP applications.

(4) *Others*: Beside the aforementioned three types of languages, some DSMSs defined their own continuous query languages. For example, Tribeca (Sullivan and Heybey 1998)

²⁶ <http://www.w3.org/TR/NOTE-xml-ql/>

created ad-hoc data description languages (DDL) to specify stream sources, filter tuples, apply sliding windows, and output results.

5.1.3 Summary

As we can see from this Chapter 5.1, publish/subscribe systems and DSMSs are similar in terms of their high level architectures and functionalities. Instead of “reinventing the wheel,” we have tried to adopt the ideas and solutions from the existing works when proposing GeoPubSubHub. However, there are still some important components we have not yet discussed in the GeoPubSubHub, such as the dissemination approach, the adaptive query processing, and the continuous query language. We believe that these topics are certainly worth investigation; and they are in our future directions.

5.2 Related approaches for the sensor web context

As mentioned earlier, GeoPubSubHub is a publish/subscribe system specifically designed for the sensor web. As this topic is very domain-specific and was seldom discussed, some critical issues (as we discussed in Chapter 2.2) have not yet been addressed. Therefore, in this thesis, we have proposed new solutions to solve some of these issues, namely the *sensor web input adaptor*, *LOST-Tree*, and *AHS-Model*. Here we introduce the existing approaches related to these three solutions. Please note that since the work of semantic layer service is a cooperative work and the details have been published in other documents, readers interested in those details are referred to Knoechel et al. (2011) and Knoechel et al. (2013).

5.2.1 Works related to sensor web input adaptor

The sensor web input adaptor in GeoPubSubHub is designed to retrieve sensor data from data sources in a timely manner. As the input adaptor utilizes LOST-Tree to aggregate

subscriptions and the works related to LOST-Tree are presented in Chapter 5.2.2, this chapter mainly focus on the adaptive sensor stream feeder.

One of the major issues in the input adaptor is that most sensor data sources only support request/response communication (e.g., OGC SOS), which means that a client needs to initiate the communication in the first place. However, users cannot know when a new data will be available in data sources. We propose the adaptive sensor steam feeder to first *pull* data from sources in a timely manner and then *push* the new data to consumers. Hence, we name this type of processing as a *hybrid pull-push* approach.

Since the publish/subscribe paradigm has been wildly applied in the web blogging applications, users can get notifications of new posts (*i.e.*, feeds) through technologies such as RSS or Atom. However, similar to most sensor data sources, some websites do not support publishing their contents. In this case, a hybrid pull-push approach similar to the proposed adaptive feeder was used to retrieve contents from those websites. For example, Superfeedr²⁷ acts like a proxy that pulls feeds in one to fifteen minutes frequency from any subscribed webpages and forwards new content to subscribers. In addition, in order to deal with the scalability issues, Superfeedr also utilizes the cloud computing infrastructure.

The other example is the ifttt.com²⁸, which is shorthand for “if this then that”. This application allows users to create or use existing “recipes” while each recipe contains a trigger and an action. Triggers can be events that happen in supported applications (which are called as “channels”). Actions are functions in these channels. For example, one existing recipe is “if a new Gmail arrives, change my Philips hue light bulb to red”. In order to use this recipe, users

²⁷ <http://superfeedr.com/>

²⁸ <https://ifttt.com/>

need to first activate Gmail and the Philips hue channels by allowing ifttt.com to access these applications on behalf of them (through OAuth²⁹). Although ifttt.com does not explain their algorithms, we believe it would be similar to the hybrid pull-push approach that the adaptive feeder and Superfeedr use.

In addition, PubSubHubbub³⁰ is a standard protocol designed to extend the Atom (and RSS) protocols for data feeds. PubSubHubbub can be applied to any resource as long as it is accessible via HTTP. There are three types of components in PubSubHubbub, namely *publishers*, *subscribers*, and *hubs*. First, a subscriber pulls an HTTP resource from a web server. If the HTTP response contains a header that references to a hub, the subscriber is able to subscribe the resource on that hub. For publishers, whenever they update resources that reference to a hub in the HTTP headers, they post notifications to the hub. Then the hubs notify their subscribers. Superfeedr supports the PubSubHubbub protocol as well.

Compared to the aforementioned solutions, the proposed input adaptor is unique in that (1) it handles spatio-temporal subscription aggregation with the help of LOST-Tree and (2) it tries to reduce the number of requests by detecting sensor data sampling periods and scheduling new retrieving requests accordingly.

5.2.2 Works related to LOST-Tree

The LOST-Tree was originally designed to be a sensor data loading component in a sensor web browser to filter out redundant requests for efficient sensor data loading. Efficiently transmitting large amounts of sensor data over the WWW is known as a major challenge (Nath et al. 2006). To date there have been few investigations of efficient spatio-temporal data loading

²⁹ OAuth is an open standard for authorization for clients to access and control server resources on behalf of a resource owner.

³⁰ <https://code.google.com/p/pubsubhubbub/>

mechanisms for a sensor web browser. Many tree structures have been extended for supporting temporal query on R-Tree (Guttman 1984) or quadtree (Finkel and Bentley 1974), including RT-Tree (Xu et al. 1990), 3D R-Tree (Theodoridis 1996), MRA-Tree (Lazaridis and Mehrotra 2001), aRB-Tree (Papadias et al. 2002), SB-Tree (Yang and Widom 2003), MV3R-Tree (Tao and Papadias 2001), and linear quadtree (Tzouramanis et al. 1998). However, these structures were all originally developed for data management rather than for data loading. ArchRock (Woo 2006), IrisNet (Gibbons et al. 2003) and GSN (Aberer et al. 2006) studied sharing sensor data over the Internet. However, these three works did not focus on the construction of a sensor web browser; and no efficient data loading mechanism was discussed.

One of the most relevant works is implemented by 52°North. They provide an online platform called SensorWeb Client³¹, which allows users to access data from OGC SOSs, visualize sensors on a map, and retrieve time-series data after selecting sensors. However, although their client is also a sensor web browser, they do not employ any loading management mechanism. They use a naïve data loading approach as their client re-loads data whenever the view of map is changed.

The other works similar to LOST-Tree were presented by Nath et al. (2006) and Ahmad and Nath (2008). They presented an architecture for a sensor data portal (*i.e.*, SensorMap) and proposed COLR-Tree to address challenges of sensor data loading and management on the portal side. Although COLR-Tree is designed for a sensor data portal instead of a sensor web browser, its strategy on the sensor data loading is worth to be analyzed.

³¹ <http://sensorweb.demo.52north.org/ThinSweClient2.0/Client.html>

COLR-Tree uses *slot-cache* and *sampling* mechanisms to reduce transmission load. The slot-cache technique appends slots on each R-Tree node, where each slot represents a certain time period for sensor data aggregation. When the portal receives spatio-temporal requests, COLR-Tree checks whether the number of cached sensors is larger than a predefined number. If yes, the portal returns cached data; otherwise, COLR-Tree uniformly loads additional sensors to fulfill the predefined number.

We argue that COLR-Tree has two major issues. First, COLR-Tree couples data loading and data management. This means in order to decide whether or not to send out requests, the data loading component needs to traverse through an index tree of the cached sensor readings (*i.e.*, the data management component). Because of the big sensor web data phenomenon presented in Chapter 1.1, we argue that the index tree size would grow rapidly and the COLR-Tree solution would subsequently become inefficient.

The second issue of COLR-Tree is that COLR-Tree aggregates and samples sensors' raw readings to prevent the growth of the data management tree. While we argue that users, especially scientists, require raw readings to develop their models and applications, a sensor web client that provides aggregated readings may not be suitable for many applications.

As a result, we recommend that the data loading and management should be decoupled in sensor web browsers. In this case, not only the transmission load can be reduced but also raw sensor readings can be provided. Hence, we propose LOST-Tree to manage data loading and a client-side cache to manage sensor readings.

5.2.3 Works related to AHS-Model

AHS-Model is proposed to efficiently process topological operators in a geospatial sensor web publish/subscribe system. A geospatial publish/subscribe system was seldom discussed

compared to the general-purpose systems. Although there have been some works that discussed the topic of supporting spatial operators in a publish/subscribe system, most of them simply applied spatial database join operations to prove the concept.

Kassab et al. (2010) implemented the *WITHIN* operator in a publish/subscribe system for fire emergencies application. They applied ArcGIS Engine v9.3 .NET SDK as a black box in their notification service component to determine topological relationships. PLACE (Mokbel et al. 2005), as a DSMS application, first groups the geometries of objects (*i.e.*, publications) and queries (*i.e.*, subscriptions) into two tables and then performs spatial join on these two tables. PLACE implemented *INSIDE* (*i.e.*, *WITHIN*) and *kNN* (*i.e.*, k Nearest Neighbor) operators to prove the concept. Ali et al. (2010) applied the Microsoft SQL Server Spatial Library to support spatial queries in their Microsoft StreamInsight system. However, we argue none of these researches discussed about improving the efficiency of geospatial algorithms for publish/subscribe systems, and some of them do not support all topological relationships.

As we mentioned in Chapter 3.6.2, DE-9IM (Clementini et al. 1993) is the typical approach to determine topological relationships; and OGC Simple Feature Access Specification has defined eight topological relationships. However, since the topological relationship between two geometries is determined independently, the determination process is computationally expensive (Clementini et al. 1994). In order to reduce the computation load of determining topological relationships, Clementini et al. (1994) proposed a two-steps approach: filter and refinement. As many spatial join techniques applied MBRs (*i.e.*, minimum bounding rectangles) to reduce computation load (Jacox and Samet 2007), Clementini et al. (1994) also used MBR as approximations to find the candidate geometries in the filter stage. Then the refinement stage performs the actual DE-9IM process to determine the topological relationships.

However, although the two-step approach is suitable on spatial joins and determinations of topological relationship in spatial databases, we argue that this approach could be further improved for publish/subscribe systems. For example, in a publish/subscribe system, queries are continuous and pre-defined. We can pre-generate and pre-index the required information from subscriptions' geometries. Then, similar to the idea of sharing operators across similar query plans in publish/subscribe systems (Arasu et al. 2004), we can aggregate pre-generated subscriptions indices into a single data structure and directly intersect publications with this structure for all the subscriptions.

Moreover, in order to aggregate as much process as possible, instead of using MBRs to find candidates (*i.e.*, the filter step), we try to move the intersection processes from the refinement step to the filter step. For example, we generate indices with a quadtree tile structure (Gaede and Gunther 1998) and use the indices to represent geometries. Therefore, while matching a publication with the aggregated subscription indices, the process not only finds candidates but also performs intersections between the publication and subscriptions. With these ideas, we propose AHS-Model to efficiently process topological operators in GeoPubSubHub.

Chapter Six: **Conclusions and Future Work**

We have presented the GeoPubSubHub, a geospatial sensor web publish/subscribe system. GeoPubSubHub utilizes the continuous query processing model to address the defined big sensor web data challenges and provide timely notifications. Although there has been many investigations on general-purpose systems applying continuous query processing model (e.g., publish/subscribe systems, DSMS), only few literatures discussed about a geospatial sensor web publish/subscribe system.

In order to provide a comprehensive overview of a geospatial sensor web publish/subscribe system, Chapter 2 presented the identified challenges for constructing a geospatial publish/subscribe system in the sensor web context. And Chapter 3 presented the design of GeoPubSubHub including proposed solutions for addressing the identified challenges and overall system architecture.

While some of the challenges are common with general-purpose systems, some of them are unique because of the nature of sensor web data and data sources, such as the *pull-based data sources*, *large number of data sources*, *heterogeneous sensor web data*, *geospatial data and queries*, and *sensor web data visualization*. While some of the proposed solutions are similar to existing solutions, we put our focus on the modules that we believe are most unique and critical in the context of a geospatial sensor web publish/subscribe system. Hence, we proposed the sensor web input adaptor, LOST-Tree, semantic layer service, AHS-Model, and sensor web browser.

The sensor web input adaptor has two major functionalities. First, it applies the proposed LOST-Tree to aggregation overlapped spatio-temporal cubes in order to avoid redundant sensor data transmission. Second, the adaptive sensor stream feeder tries to detect the sensor sampling

period and schedule next requests accordingly. As shown in the experimental results, if a service makes data available online as soon as they are measured, the proposed adaptive sensor stream feeder can retrieve sensor data in a timely manner without any unnecessary request.

In GeoPubSubHub, LOST-Tree is applied in both the sensor web input adaptor and the sensor web browser to avoid redundant sensor data transmission. We evaluated LOST-Tree with an OGC SOS. Our evaluation results demonstrated that, with LOST-Tree and the local cache, a sensor web browser can attain sensor data loading 100 times faster than the naïve solution. LOST-Tree is lightweight (always less than 164 Kbytes during our evaluation) and scalable in terms of the number of spatio-temporal requests. Also, LOST-Tree can avoid unnecessary transmissions up to 100% in an efficient manner (less than 4 milliseconds for the realistic scenario).

The semantic layer service was proposed to integrate the heterogeneous sensor web data and provides users a coherent view on the sensor web data. The semantic layer service uses a bottom-up approach to address the semantic and syntactic heterogeneity issues. As the semantic layer service is a cooperative contribution, the detail methodology and evaluation can be found in Knoechel et al. (2011) and Knoechel et al. (2013).

The AHS-Model was proposed to efficiently determine topological relationships between new sensor web data and a large number of predefined subscriptions. The experimental results show that AHS-Model is more scalable than the traditional solution in terms of handling a large number of subscriptions. With the help of distributed processing, AHS-Model is able to perform the indexing and matching steps in a timely manner. In addition, AHS-Model can finish most queries with 100 milliseconds end-to-end latency for a more realistic dataset.

To sum up, while the world-wide sensor web is generating tremendous volumes of priceless data that enables scientists to observe previously-unobservable phenomena, a software component that efficiently converts sensor web data into information is necessary for time-critical applications such as emergency response systems. To harness the full potential of sensor web, the GeoPubSubHub was proposed to efficiently process geospatial sensor web data streams and provide timely notifications. The major contributions of this thesis are as follows:

1. Based on the challenges and existing approaches in a general-purpose publish/subscribe architecture we summarized, we identified seven challenges in a sensor web publish/subscribe architecture. While some of them are common with general-purpose systems, some of them are unique because of the nature of geospatial sensor web data and data sources.
2. We proposed solutions with overall system architecture and workflow to address the identified challenges in order to cover all aspects of constructing a geospatial sensor web publish/subscribe system.
3. This thesis focused on the modules that we believe are most unique and critical in the context of a geospatial sensor web publish/subscribe system. The proposed sensor web input adaptor is able to aggregate users' spatio-temporal requests and efficiently retrieve sensor data from pull-based data sources while avoiding unnecessary requests. The proposed LOST-Tree can serve as a data loading component in a sensor web browser to effectively and efficiently aggregate spatio-temporal cubes and avoid redundant requests. The proposed AHS-Model is able to efficiently determine the topological relationships in a sensor web publish/subscribe system, which demonstrates that time-consuming geospatial operators can be revised for time-critical applications.

For future directions, we believe that each of the identified challenges (in Chapter 2.2) is an interesting and important research question that is worth further investigation. In addition, as mentioned in Chapter 5, we have not yet discussed the dissemination approaches, adaptive query processing, and continuous query language in the GeoPubSubHub. These topics are also important.

Furthermore, some of the proposed solutions could be further improved. For example, LOST-Tree may be improved to attain the best performance with an automatic and adaptive configuration mechanism based on different cache scenarios or sensor web services. By collecting the information of sensor data sampling rate and geographical density, LOST-Tree could further improve the utilization of each request. For distributed AHS-Model or even the GeoPubSubHub architecture, further investigation is required for more sophisticated load balancing approaches.

In general, despite the fact that there are still many important topics and directions to be investigated, as one of the first geospatial sensor web publish/subscribe system, we believe that the proposed solutions and GeoPubSubHub architecture serve as a promising initiative to address the unique big sensor web data challenges and consequently allow us to harvest the full potential of the world-wide sensor web.

References

1. Abadi, D.J., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S., 2003. Aurora: A New Model and Architecture for Data Stream Management. *Very Large Data Bases (VLDB) Journal*, 12, 2.
2. Aberer, K., Hauswirth, M., and Salehi, A., 2006. A Middleware for Fast and Flexible Sensor Network Deployment. In *32nd International Conference on Very Large Data Bases (VLDB)*, 1199-1202.
3. Aguilera, M.K., Strom, R.E., Sturman, D.C., Astley, M., and Chandra, T.D., 1999. Matching Events in a Content-based Subscription System. In *Proceedings of the 18th ACM Symposium on Principles of Distributed Computing*, New York, USA, 53-61.
4. Ahmad, Y., Nath, S., 2008. Colr-Tree: Communication-Efficient Spatio-Temporal Indexing for a Sensor Data Web Portal. In *Proceedings of IEEE International Conference on Data Engineering*, (Cancun, Mexico, 7-12 April 2008), 784-793.
5. Ali, M., Chandramouli, B., Raman, B.S., and Katibah, E., 2010. Real-Time Spatio-Temporal Analytics using Microsoft StreamInsight. In *18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS '10)*. San Jose, CA, USA.
6. Altinel, M. and Franklin, M.J., 2000. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proceedings of the 2000 International Conference on Very Large Databases (VLDB)*, 53-64.
7. Arasu, A., Babcock, B., Babu, S., Cieslewicz, J., Datar, M., Ito, K., Motwani, R., Srivastava, U., and Widom, J., 2004. STREAM: The Stanford Data Stream Management System.

8. Auyunirundronkool, K., Chen, N., Peng, C., Yang, C., Gong, J., and Silapathong, C., 2012. Flood Detection and Mapping of the Thailand Central Plain using Radarsat and MODIS under a Sensor Web Environment. *International Journal Applied Earth Observation and Geoinformation*, 14, 245-255.
9. Avnur, R. and Hellerstein, J., 2000. Eddies: Continuously Adaptive Query Processing. *ACM Special Interest Group on Management of Data (SIGMOD)*, 261-272.
10. Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J., 2002. Models and Issues in Data Stream Systems. *ACM Symposium on Principles of Database Systems*, 1-16.
11. Bacon, J., Moody, K., Bates, J., Hayton, R., Ma, C., McNeil, A., Seidel, O., and Spiteri, M., 2000. Generic Support for Distributed Applications. *IEEE Computation*, 33, 3, 68-76.
12. Bai, Q., Guru, S.M., Smith, D., Liu, Q., Terhorst, A. 2010. A Multi-Agent View of the Sensor Web. In *Advances in Practical Multi-Agent Systems*, 325, 435-444.
13. Bai, Y., Thakkar, H., Wang, H., Luo, C., and Zaniolo, C., 2006. A Data Stream Language and System Designed for Power and Extensibility. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, 337-346.
14. Bakillah, M., Liang, S.H.L., Zipf, A., 2012. Toward Coupling Sensor Data and Volunteered Geographic Information (VGI) with Agent-Based Transport Simulation in the Context of Smart Cities. In *Proceeding of the First ACM SIGSPATIAL Workshop on Sensor Web Enablement, Redondo Beach, (CA, USA, 6–9 November 2012)*, 17-23.
15. Bayer, R. and McCreight, E., 1972. Organization and Maintenance of Large Ordered Indexes. *Acta Informatica*, 1 (3), 173-189.

16. Birman, K. and Joseph, T., 1987. Exploiting Virtual Synchrony in Distributed Systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, 21 (5), 123-138.
17. Bonnet, P., Gehrke, J., and Seshadri, P., 2001. Towards Sensor Database Systems. In *Proceedings of International Conference on Mobile Data Management*, 3-14.
18. Botts, M., Percivall, G., Reed, C., and Davidson, J., 2007. OGC[®] Sensor Web Enablement: Overview and High Level Architecture (OGC 07-165). *Open Geospatial Consortium White Paper*, 28 December 2007.
19. Brenna, L., Demers, A., Gehrke, J., Hong, M., Ossher, J., Panda, B., Riedewald, M., Thatte, M., and White, W., 2007. Cayuga: A High-Performance Event Processing Engine. In *Proceedings of the 2007 ACM Special Interest Group on Management of Data (SIGMOD)*, 1100-1102.
20. Bröring, A., Echterhoff, J., Jirka, S., Simonis, I., Everding, T., Stasch, C., Liang, S., Lemmens, R., 2011. New Generation Sensor Web Enablement. *Sensors*, 11, 2652-2699.
21. Campailla, A., Chaki, S., Clarke, E., Jha, S., and Veith, H., 2001. Efficient Filtering in Publish-Subscribe Systems using Binary Decision. In *Proceedings of the International Conference on Software Engineering*, 443-452.
22. Carzaniga, A., Rosenblum, D., Wolf, A., 2001. Design and Evaluation of a Wide-Area Event Notification Service. *ACM Transactions on Computer Systems*, 19 (3), 332-383.
23. Castro, M., Druschel, P. Kermarrec, A.M., and Rowstron, A., 2002. SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communications*, 20 (8), 1489-1499.

24. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S.R., Reiss, F., and Shah, M.A., 2003. TelegraphCQ: Continuous Dataflow Processing. *ACM Special Interest Group on Management of Data (SIGMOD)*.
25. Chen, C.P., Chuang, C.L., Jiang, J.A., 2013. Ecological Monitoring using Wireless Sensor Networks-Overview, Challenges, and Opportunities. In *Advancement in Sensing Technology, Smart Sensors, Measurement and Instrumentation*, 1, 1-21.
26. Chen, J., DeWitt, D.J., Tian, F., and Wang, Y., 2000. NiagraCQ: A Scalable Continuous Query System for Internet Databases. In *Proceedings of the 2000 ACM Special Interest Group on Management of Data (SIGMOD)*, 379-390.
27. Clementini, E., Di Felice, P., and van Oosterom, P., 1993. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In *Proceedings of 3rd International Symposium on Large Spatial Databases*, Lecture Notes in Computer Science, 692, Singapore, June 1993, 277-295.
28. Clementini, E., Sharma, J., and Egenhofer, M.J., 1994. Modeling Topological Spatial Relations: Strategies for Query Processing. *Computers and Graphics*. 18 (6), 815-822.
29. Craglia, M., Goodchild M.F., Annoni, A., Camara, G., Gould, M., Kuhn, W., Mark, D., Masser, I., Maguire, D., Liang, S., and Parsons, E., 2008. Next-Generation Digital Earth. *International Journal of Spatial Data Infrastructures Research*, 3, 146-167.
30. Cranor, C., Johnson, T., Spataschek, O., and Shkapenyuk, V., 2003. Gigascope: A Stream Database for Network Applications. In *ACM Special Interest Group on Management of Data (SIGMOD)*, 647-651.

31. Cugola, G. and Margara, A., 2010. Processing Flows of Information: From Data Stream to Complex Event Processing. *Technical report*. Politecnico di Milano.
32. Cugola, G., Nitto, E.D., and Fugetta, A., 2001. The JEDI Event-based Infrastructure and Its Application to the Development of the OPSS WFMS. *IEEE Transaction on Software Engineering*, 27 (9), 827-850.
33. Dean, J., Ghemawat, S., 2008. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM*, 51 (1).
34. Deering, S., 1989. Host Extensions for IP Multicasting (IETF RFC 1112) [online]. Available: <http://tools.ietf.org/html/rfc1112> [Accessed 2 December 2013].
35. Delin, K., 2005. Sensor Webs in the Wild. In *Wireless Sensor Networks: A Systems Perspective*, Artech House, London, UK, 2005.
36. Diao, Y., Fischer, P., Franklin, M., and To, R., 2002. YFilter: Efficient and Scalable Filtering of XML Documents. In *Proceedings of the 18th International Conference on Data Engineering*.
37. Echterhoff, J. and Everding, T., 2008. Sensor Event Service Interface Specification (proposed) (OGC 08-133). *Open Geospatial Consortium Discussion Paper*, 10 October 2008.
38. Esper, 2012. [online] Available: <http://esper.codehaus.org/> [Accessed 2 December 2013].
39. Eugster, P., Felber, P.A., Guerraoui, R., and Kermarrec, A.M., 2003. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35 (2), 114-131.
40. Everding, T. and Echterhoff, J., 2008. Event pattern Markup Language (EML) (OGC 08-132). *Open Geospatial Consortium Discussion Paper*, 5 November 2008.

41. Fabret, F., Jacobsen, H., Llibat, F., Pereira, J., Ross, K., and Shasha, D., 2001. Filtering Algorithms and Implementations for Very Fast Publish/Subscribe Systems. In *Proceedings of ACM Special Interest Group on Management of Data (SIGMOD)*, 115-126.
42. Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L., 1997. A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing. *IEEE/ACM Transactions on Networking*, 5 (6), 784-803.
43. Finkel, R. and Bentley, J.L., 1974. Quad Trees: A Data Structure for Retrieval on Composite Keys. *Acta Informatica*, 4 (1), 1-9.
44. Gaede, V. and Gunther, O., 1998. Multidimensional Access Methods. *Journal of ACM Computing Surveys*, 30 (2), 170-231.
45. Gedik, B., Andrade, H., Wu, K.L., Yu, P.S., and Doo, M., 2008. SPADE: The System S Declarative Stream Processing Engine, In *Proceedings of the 2008 ACM Special Interest Group on Management of Data (SIGMOD)*, 1123-1134.
46. Gibbons, P.B., Karp, B., Ke, Y., Nath, S., and Seshan, S., 2003. Irisnet: An Architecture for a World Wide Sensor Web. *IEEE Pervasive Computing*, 2 (4), 22-33.
47. Golab, L. and Ozsu M.T., 2003. Issues in Data Stream Management. In *the 2003 ACM Special Interest Group on Management of Data*, 32 (2), 5-14.
48. Guttman, A., 1984. R-Tree: A Dynamic Index Structure for Spatial Searching. In *ACM Special Interest Group on Management of Data (SIGMOD)*, 47-57.
49. Hart, J.K., Martinez, K., 2006. Environmental Sensor Networks: A Revolution in the Earth System Science? *Earth Science Review*, 78, 177-191.

50. Herring, J.R., 2011. OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 1: Common architecture (OGC 06-103r4). *OpenGIS® Implementation Standard*, 28 May 2011.
51. Hsieh, T.T., 2004. Using Sensor Networks for Highway and Traffic Applications. *IEEE Potentials*, 23, 13-16.
52. Huang, C.Y. and Liang, S., 2013. LOST-Tree: A Spatio-Temporal Structure for Efficient Sensor Data Loading in a Sensor Web Browser. *International Journal of Geographical Information Science*, 27 (6), 1190-1209.
53. Jacox, E.H. and Samet, H., 2007. Spatial Join Techniques. *ACM Transactions on Database Systems*, 32 (1).
54. Kassab, A., Liang, S., and Gao, Y., 2010. Real-Time Notification and Improved Situational Awareness in Fire Emergencies using Geospatial-based Publish/Subscribe. *International Journal of Applied Earth Observation and Geoinformation*, 12 (6), 431-438.
55. Knoechel, B., Huang, C.Y., Liang, S.H.L., 2011. Design and Implementation of a System for the Improved Searching and Accessing of Real-World SOS Services. In *Proceedings of 2011 International Workshop on Sensor Web Enablement*, (Banff, AB, Canada, 6–7 October 2011).
56. Knoechel, B., Huang, C.Y., Liang, S.H.L., 2013. A Bottom-Up Approach for Automatically Grouping Sensor Data Layers by their Observed Property. *ISPRS International Journal of Geo-Information*, 2, 1-26.
57. Laney, D., 2001. 3D Data Management: Controlling Data Volume, Velocity and Variety [online] Available: <http://blogs.gartner.com/doug-laney/files/2012/01/ad949-3D-Data->

Management-Controlling-Data-Volume-Velocity-and-Variety.pdf [Accessed 2 December 2013].

58. Lazaridis, I., and Mehrotra, S., 2001. Progressive Approximate Aggregate Queries with a Multi-Resolution Tree Structure. In *ACM Special Interest Group on Management of Data (SIGMOD)*.
59. Lerner, A. and Shasha, D., 2003. AQuery: Query Language for Ordered Data, Optimization Techniques, and Experiments. *Very Large Databases*, 345-356.
60. Levenshtein, V., 1966. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics, Doklady*, 10, 707-710.
61. Liang, S.H.L., Chang, D., Badger, J., Rezel, R., Chen, S., Huang, C.Y., and Li, R.Y., 2010. Capturing the Long Tail of Sensor Web. *International Workshop on Role of Volunteered Geographic Information in Advancing Science*, Sep. 14, Zurich, Switzerland.
62. Liang, S.H.L., Croitoru, A., Tao, C.V., 2005. A Distributed Geospatial Infrastructure for Sensor Web. *Computers & Geosciences*, 31, 221-231.
63. Liang, S.H.L. and Huang, C.Y., 2013. GeoCENS: A Geospatial Cyberinfrastructure for the World-Wide Sensor Web. *Sensors*, 13 (10), 13402-13424.
64. Liu, L., Pu, C., and Tang, W., 1999. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Transactions on Knowledge and Data Engineering*, 11 (4), 583-590.
65. Luckham, D., 2002. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley.

66. Madden, S. and Franklin, M.J., 2002. Fjording the Stream: An Architecture for Queries over Streaming Sensor Data. In *Proceedings of the 2002 International Conference on Data Engineering*, 555.
67. Mainwaring, A., Polastre, J., Szewczyk, R., Culler, D., Anderson, J., 2002. Wireless Sensor Networks for Habitat Monitoring. In *the 2002 ACM International Workshop on Wireless Sensor Networks and Applications*, (Atlanta, GA, USA, 28 September 2002).
68. Michelson, B.M., 2006. Event-Driven Architecture Overview, Patricia Seybold Group.
69. Microsoft, 2013. Microsoft StreamInsight 2.1 [online]. Available: [http://msdn.microsoft.com/en-us/library/hh750619\(v=SQL.10\).aspx](http://msdn.microsoft.com/en-us/library/hh750619(v=SQL.10).aspx) [Accessed 2 December 2013].
70. Mokbel, M.F., Ghanem, T.M., and Aref, W.G., 2003. Spatio-Temporal Access Methods. *IEEE Data Engineering Bulletin*, 26 (2), 40-49.
71. Mokbel, M.F., Xiong, X., and Aref, W.G., 2005. Continuous Query Processing of Spatio-Temporal Data Streams in PLACE. *GeoInformatica*, 9 (4), 343-365.
72. Moodley, D., Simonis, I., 2006. New Architecture for the Sensor Web: The SWAP Framework. In *Proceedings of the 5th International Semantic Web Conference*, (Athens, GA, USA, 5-9 November 2006).
73. Morton, G.M., 1966, A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. *Technical Report*, Ottawa, Canada: IBM Ltd.
74. Na, A. and Priest, M., 2007. OpenGIS Sensor Observation Service (OGC 06-009r6). *Open Geospatial Consortium Implementation Specification*, 26 October 2007.

75. Nath, S., Liu, J., and Zhao, F., 2006. Challenges in Building a Portal for Sensors World-Wide. In *First Workshop on World-Sensor-Web: Mobile Device Centric Sensory Networks and Applications*, (Boulder, CO, October 2006).
76. Nguyen, B., Abiteboul, S., Cobena, G., and Preda, M., 2001. Monitoring XML Data on the Web. In *Proceedings of the 2001 ACM Special Interest Group on Management of Data (SIGMOD)*, 437-448.
77. Oracle, 2009. Oracle Complex Event Processing: Lightweight Modular Application Event Stream Processing in the Real World, White Paper [online]. Available: <http://www.oracle.com/technetwork/middleware/complex-event-processing/overview/oracle-37.pdf> [Accessed 2 December 2013].
78. Papadias, D., Tao, Y., Kalnis, P., and Zhang, J., 2002. Indexing Spatio-Temporal Data Warehouse. In *Proceedings of IEEE International Conference on Data Engineering (ICDE)*.
79. Pietzuch, P.R., 2004. Hermes: A Scalable Event-based Middleware. Queens' College, University of Cambridge, Cambridge, UK.
80. Powell, D., 1996. Group Communication. *Communications of the ACM*, 39 (4), 50-97.
81. Resch, B., Mittlboeck, M., Girardin, F., Britter, R., Ratti, C., 2009. Live Geography - Embedded Sensing for Standardised Urban Environmental Monitoring. *International Journal on Advanced Systems and Measurements*, 2, 156-167.
82. Rogers, S. 2011. Big data is scaling BI and analytics [online]. Available: http://www.information-management.com/issues/21_5/big-data-is-scaling-bi-and-analytics-10021093-1.html [Accessed 3 October, 2013].

83. Schreier, U., Pirahesh, H., Agrawal, R., and Mohan, C., 1991. Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS. In *Proceedings of the 1991 International Conference on Very Large Databases (VLDB)*, 469-478.
84. Schultz-Moeller, N.P., Migliavacca, M., and Pietzuch, P., 2009. Distributed Complex Event Processing with Query Optimisation. In *International Conference on Distributed Event-Based Systems*, (Nashville, TN, USA).
85. Schut, P., 2007. OpenGIS[®] Web Processing Service (OGC 05-007r7). OpenGIS[®] Standard, 8 June 2007.
86. Simonis, I. and Dibner, P.C., 2007. OpenGIS[®] Sensor Planning Service Implementation Specification (OGC 07-014r3). *OpenGIS[®] Implementation Specification*, 2 August 2007.
87. Simonis, I. and Echterhoff, J., 2006. Draft OpenGIS[®] Web Notification Service Service Implementation Specification (OGC 06-095). *OpenGIS[®] Best Practices Paper*, 18 November 2006.
88. Skeen, D., 1998. Vitria's Publish-Subscribe Architecture: Publish-Subscribe Overview [online]. Available: <http://www.vitria.com> [Accessed 2 December 2013].
89. Stasch, C., Foerster, T., Autermann, C., Pebesma, E., 2012. Spatio-Temporal Aggregation of European Air Quality Observations in the Sensor Web. *Towards A Geoprocessing Web*, 47, 111-118.
90. StreamBase, 2011. *StreamSQL Guide* [online]. Available: <http://www.streambase.com/developers/docs/latest/streamsql/index.html> [Accessed 2 December 2013].

91. Sullivan, M. and Heybey, A., 1998. Tribeca: A System for Managing Large Databases of Network Traffic. In *Proceedings of the annual conference on USENIX Annual Technical Conference*, (Berkeley, CA, USA), 13-24.
92. Szarowski, R., 2010. Hybrid Publish-Subscribe: A Compromise Approach for Large-Scale [online]. Available: http://ceur-ws.org/Vol-74/files/FORUM_61.pdf [Accessed 2 December 2012].
93. Tao, Y. and Papadias, D., 2001. MV3R-Tree: A Spatio-Temporal Access Method for Timestamp and Interval Queries. In *27th International Conference on Very Large Data Bases (VLDB)*, 431-440.
94. Terry, D., Goldberg, D., Nichols, D., and Oki, B., 1992. Continuous Queries over Append-Only Databases. In *Proceedings of the 1992 ACM Special Interest Group on Management of Data (SIGMOD)*, 321-330.
95. Theodoridis, Y., Vazirgiannis, M., and Sellis, T., 1996. Spatio-Temporal Indexing for Large Multimedia Applications. In *Proceedings of the IEEE Conference on Multimedia Computing and Systems*.
96. TIBCO, 1999. TIB/Rendezvous, *White paper* [online]. Available: http://www.tibco.com/multimedia/ds-rendezvous_tcm8-826.pdf [Accessed 2 December 2013].
97. Tzouramanis, T., Vassilakopoulos, M., and Manolopoulos, Y., 1998. Overlapping Linear Quadrees: A Spatio-Temporal Access Method. In *the 6th ACM International Symposium on Advances in Geographic Information Systems*, (Washington, D.C., United States, 1998), 1-7.

98. Woo, A., 2006. Demo Abstract: A New Embedded Web Services Approach to Wireless Sensor Networks. In *Proceedings of the Fourth ACM Conference on Embedded Networked Sensor Systems*, (Boulder, Colorado, United States, November 2006). ACM, 347.
99. Wu, E., Diao, Y., and Rizvi, S., 2006. High-Performance Complex Event Processing over Streams, In *Proceedings of the 2006 ACM Special Interest Group on Management of Data (SIGMOD)*, 407-418.
100. Xu, N., 2002. A Survey of Sensor Network Applications, *IEEE Communications Magazine*, 40, 102-144.
101. Xu, X., Han, J., and Lu, W., 1990. RT-Tree: An Improved R-Tree Indexing Structure for Temporal Spatial Databases. In *Proceedings of the International Symposium on Spatial Data Handling*, 1040-1049.
102. Yang, X., Song, W., De, D., 2011. LiveWeb: A Sensorweb Portal for Sensing the World in Real-Time. *Tsinghua Science & Technology*, 16, 491-504.
103. Yang, J. and Widom, J., 2003. Incremental Computation and Maintenance of Temporal Aggregates. *Very Large Databases (VLDB) Journal*, 12, 3.
104. Zimbrão, G. and Souza, J.M., 1998. A Raster Approximation for the Processing of Spatial Joins. In *Proceedings of the 24th Very Large Data Base Conference (VLDB)*. (San Francisco, CA, USA), 558-569.