EFFICIENT VISUALIZATION OF STREAMING SENSOR NETWORK DATA USING APPROXIMATION TECHNIQUE

BY

SUNIL PAI

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE & ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

MAY 2007

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Yonghe Liu, for his constant guidance and support, and for giving me a wonderful opportunity to work on such a challenging project and also for discussing various practical and interesting aspects of the problem in hand. It was a pleasure working under him and learning from his immense knowledge.

I am also grateful to Dr. David Kung and Mr. Mike O'Dell and for serving on my committee.

I would also like to thank Mr. Joshua Been, GIS librarian, UTA for helping me with the GIS and Google Maps related stuff in the thesis. It was fascinating to discuss the new development in the field of GIS with him. Our interaction definitely shaped my thesis in many ways.

I would like to thank my family for their constant support throughout my academic career. I would like to thank all my friends for their consistent support during my thesis work. They were always there to help me out with some of the challenges I faced while working on the thesis.

April 10, 2007

ABSTRACT

EFFICIENT VISUALIZATION OF STREAMING SENSOR NETWORK DATA

USING APPROXIMATION TECHNIQUE

Sunil Pai

The University of Texas at Arlington, 2007

Supervising Professor: Dr. Yonghe Liu

By commanding a large number of wireless sensor nodes capable of sensing, communicating, and computing, wireless sensor networks have revealed their vast potential in a plethora of applications. However, due to the stringent resources limitations on each sensor node ranging from energy and computation to network bandwidth and storage, efficient and light weight systems must be designed in order to accommodate the resource limited environment. Furthermore, due to the potentially large number of sensor nodes deployed, the amount of sensory data gathered therein can be overwhelming for processing and visualizing in a resource limited central controller, such as mobile devices.

In this thesis, we describe a data processing and visualization paradigm to handle the large bursts of streaming data from wireless sensor network based monitoring systems. We employ efficient approximation techniques in order to process the information and visualize them in a timely manner as and when data arrives.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

CHAPTER 1

INTRODUCTION

Recent advances in low power wireless sensor networks are enabling new applications for wireless devices. Also, the power and flexibility of the web services is expanding the power of the Internet.

The recent advances in micro sensor technology have led to the quick development and large scale deployment of low cost and low power sensing devices with limited sensing and computational facilities. Since wireless sensor networks have a limited communication range, in order foster access to the huge chunks of data obtained from these networks, we have to build a system that facilitates access to this information in a timely and power efficient manner. A lot of effort has been directed in order to overcome the obstacles associated with connecting and sharing these heterogeneous sensor resources using architectures such as service oriented sensor web [1].

The current web based sensor network architectures are designed for specific applications, having a strong coupling between the application and its underlying data communications protocols [2]. The future sensor networks are envisioned as comprising a large range of applications providing a lot more information and intelligence to the user rather than just simple sensing and computation. To achieve this goal, a new architecture is needed where it's easy to add new services over the set of basic underlying services already provided. Also, increasingly new programming languages like Java, AJAX and ASP are providing rich set of functions and support to using new programming constructs like mash-ups, which are web based applications that combine data from more than one source into an integrated experience.

In the pretext to these new and exiting developments, a meaningful way ahead for sensor networks would be to integrate it with the power and vastness of the Internet. This would not just open up the sensor networks to a vast community of users, but also create a new paradigm of web applications which will open up avenues for new a set of services being built on top of the ones that are already exist. Our work focuses on extracting information like temperature, pressure etc from habitants and regions that are being monitored or of some interest to the user using deployed wireless sensor networks.

We will present an architecture which can be used to spawn new services on top of a base set of functions thus allowing the system to evolve and expand in a flexible manner. Example: If we have built a system to gather information about temperature and pressure changes at a costal region, linking this information to warning systems can help us develop a new set of intelligent services with acts as a early warning system for the costal region, helping protect its members against problems that my arise out of abrupt climatic changes. Another example is monitoring a nuclear power station, Sudden changes in temperature and pressure at critical regions can be observed to alert the emergency response team so that they can take timely action and avert a possible disaster.

Deployed sensor networks provide us with a large amount of data which arrives, from its deployed habitant, in a continuous stream rather than static relations. Not only are the size of these applications unbound, but the data arrives in a burst mode which makes it very difficult to effectively process all the data in a timely and efficient manner so as to input it to systems that carryout the visualization of the habitant being monitored.

In [1] the authors propose a XML based architecture to share data across different applications. But, given the large amount of streaming data that these systems can receive it is necessary to explore alternative techniques to processing this information, rather than just embed XML tags into the data which do not just result in processing and data transmission delays but also quickly drain out the energy of the deployed sensor networks, which is a cause of concern given the power constrained nature of these networks. Transmitting additional XML tags along with the data will drain out the power of these networks and reduce its life time. So, the system has to be designed to move the processing issues to the servers or proxy that collect the data, leaving the deployed sensor network to perform only the basic set of functionalities.

In [4] and [5] the authors discuss the nature of data intensive streams and the challenges they pose to process the streaming data. Since the data involved in such scenarios is huge, it can get demanding on system resources like memory and buffer space, requiring faster processors to process all of the stream's data. Even having met such resource demands, the task of visualizing this information before the next set of streaming data arrives is very challenging.

To address the problem discussed above we use techniques like approximation and random sampling to process the incoming data in a quick and efficient manner with very small reduction in efficiency so that the data can be visualized on web based systems in a very timely manner. In particular, we use simple random sampling based techniques along with some new technologies like AJAX and Google Maps APIs to create a system where information can be processed and visualized to the user as and when the data arrives. We will discuss how we can use the right approximation technique to calculate the results with minimal errors.

Techniques like approximation are necessary as the data arriving from the sensor networks tend to be overwhelmingly large in nature. If a traditional approach is used in this scenario it will result in long delay times and thus hampering the system from effectively visualizing the habitant as it dynamically changes. Also, given our need to insert huge volumes of data, and run queries on them in a very quick manner, traditional DBMS are not good enough.

We will also discuss some emerging web based technologies like AJAX and Google maps APIs to see how they can be used to help us build a dynamically changing system, which changes and adapts as and when new data sets arrives from the habitant being monitored, thus providing us with real time visualization of the habitant .

This thesis is organized as follows: Chapter 2 speaks about some related work in this area. In Chapter 3, we present the individual components, our overall framework and our motivation behind the new architecture. In Chapter 4, we present the complete architecture of the system. In Chapter 5, we will look at some experiments and analysis of the system. We conclude in Chapter 6.

CHAPTER 2

RELATED WORK

Basically there are two major approaches which are used currently to build applications involving integration of sensor network data with the Internet. One approach involves building a middleware service for sensor networks. The other approach is to define a system architecture that integrates sensor network with the Internet. We will briefly look at the work done in both these areas.

A lot of effort has been invested in building middleware support in order to make the development of sensor network applications much simple and faster [1]. Impala (Liu and Martonosi, 2004) designed for ZebraNet project designs a system that provides modular service for applications, thus providing small updates that require little transmissions that consume little energy [6]. MiLAN (Middleware linking applications and networks, Heinzelman et al., 2002) is an architecture that extends the network protocol stack and allows network specific plug-ins to convert MiLAN commands into protocol-specific commands [7]. Mires (Soutoo et al., 2005), a message oriented publish/subscribe middleware, encapsulates its interfaces and provides higher level services to the application [8]. A main component of Mires is a publish/subscribe service that intermediates communication between middleware services, which might be used as the foundation for sensor web middleware described in [1].

A slightly different approach is to integrate sensor network with grid computing. This approach will require distributed algorithms to process data from the sensor networks. A data collection network approach to address many of the technical problems of integrating resource constrained wireless sensors into traditional grid applications have been suggested by Gaynor et al., 2004. This approach is in the form of a network infrastructure, called Hourglass [10],that can provide API to a heterogeneous group of sensors.

Nickerson et al., 2005 describe a sensor web language with provides a more robust environment to deploy, maintain and operate sensor networks for mesh architectures. With mesh architecture support in SWL, multiple sensor networks are provided with greater flexibility, more reliable operation and machinery to better support self-diagnosis and inferencing with sensor data [11].

Xingchen et al. and Flavia Coimbra et al. have defined a service oriented web architecture for sensor networks in [1] and [2] respectively. In [1] the authors use the sensor web enablement (SWE) standard defined by the OpenGIS consortium (OGC), which is composed of specifications, including sensor model language (sensorML), sensor collection service, planning service and web notification service. It presents a reusable, scalable, extensible and interoperable service oriented sensor web architecture [1].

The authors use compact binary representations of XML and SOAP messages to reduce the size of messages. XML gives the flexibility of communication across heterogeneous platforms. But as discussed earlier, in the introduction, XML and SOAP introduce additional requirements to build messages and transmit them over the network. While handling large data oriented applications this can be taxing on the sensor network and can drain out it's power quickly, thus rendering them useless after a particular period. Most sensor network deployments are a one time activity and the power sources cannot be replaced or recharged.

In [4] the authors describe a general-purpose system for processing continuous queries over multiple continuous data streams and stored relations. It is designed to handle high-volume burst mode data streams with large numbers of complex continuous queries.

S. Babu and J. Widom specify a general and flexible framework in [12] for query processing in the presence of data streams. The framework captures most previous work on continuous queries and data streams, as well as subsuming related concepts such as triggers and materialized views. They map out problems, techniques, and challenges in processing continuous queries over data streams.

In [13] the authors present the problem of resource sharing when processing large numbers of continuous queries. They specifically address sliding-window aggregates over data streams.

Daniel J. Abadi et al. in their project titled Aurora [14] describe the basic processing model and architecture of Aurora, a new system to manage data streams for monitoring applications. Monitoring applications differ substantially from conventional business data processing. The fact that a software system must process and react to continual inputs from many sources (e.g., sensors) rather than from human operators requires one to rethink the fundamental architecture of a DBMS for this application area.

This thesis is influenced mainly by aggregation and approximation techniques and stream processing using continuous queries. Also, the visualization of the data is carried out by using techniques tailored to suite the needs of our application. In the next section we will discuss the individual components of our design and the significance of each component in the over all picture. We will present the architecture of the system and discuss how it differs from existing architectures. We will see some advantages of using this approach and some trade off that can occur.

CHAPTER 3

FRAMEWORK

In this chapter we describe the framework and our approach to develop the system. Our actual algorithms are described in more detail in the subsequent chapters. Let us look at the individual components first and the significance of each in the overall system. Once we understand the individual components we can understand how they fit into the overall architecture of the system.

3.1 Existing frameworks

Before describing our framework we need to take a look at some existing systems and see how they function. We can point out some short comings in these systems and see how this is addressed in our system.

The diagram below shows us the series of events that take place from the time the user opens his browser and requests for the temperature information of a region.

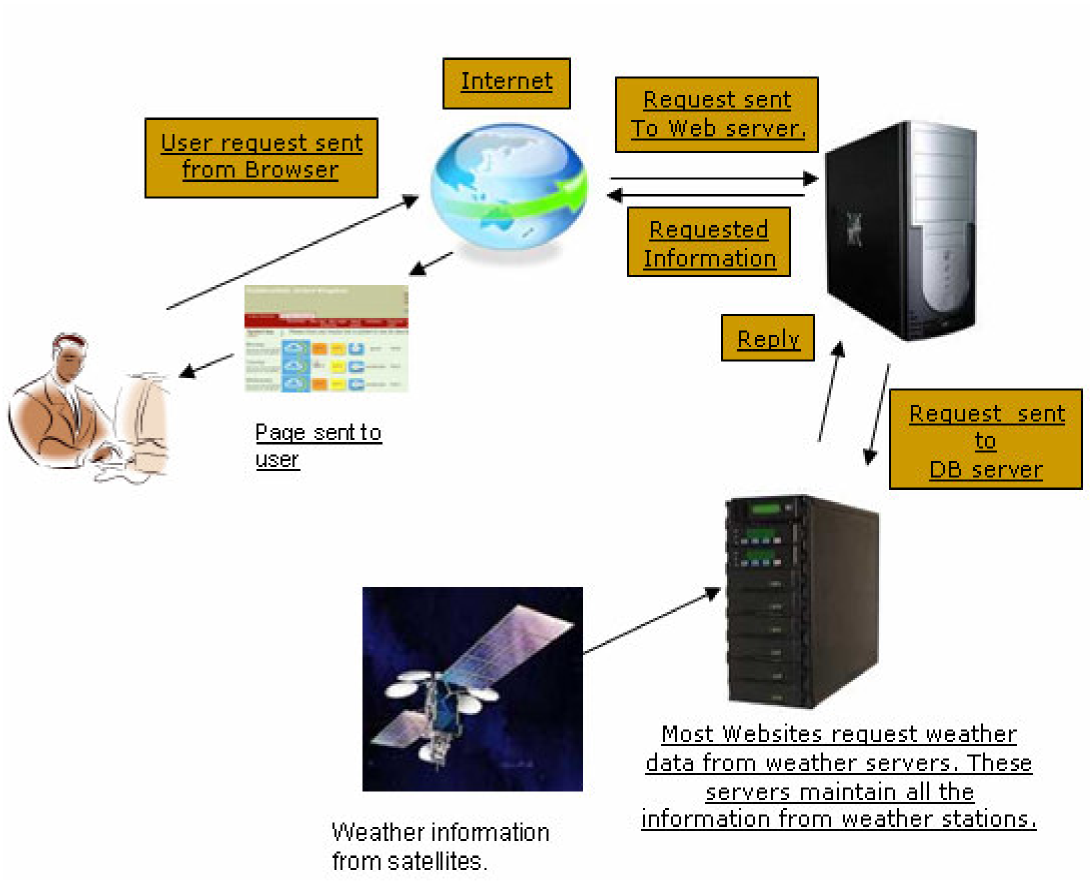

Internet
User request sent from Browser
Request sent To Web server.
Requested Information
Reply
Request sent to DB server
Page sent to user
Most Websites request weather data from weather servers. These servers maintain all the information from weather stations.
Weather information from satellites.

Figure 3.1 Example of how a user interacts with current weather systems

The series of steps that take place when user requests information about the temperature of a particular region are as follows:

1. The user enters request in the browser, generally it's an address or a zip code.

2. The user’s request is sent to the web server via the Internet.
3. Most web servers today get information from weather servers which are maintained by sites having large processing resources and storage space to store data. Example: weather.com.
4. This information is sent to the requesting server in the form of a XML message.
5. The web server, on receiving this XML message, removes the tags and formats the information with relevant images (example: Sun for a sunny day, rain icon for showers etc) and sends the html page back to the user’s browser.
6. The user’s browser displays the information to the user.

Let us discuss some problems with the current systems and see some short comings which prevent the systems from scaling up for processing very large data sets. Some problems with existing architectures are:

- They are static: As pointed out earlier, the current systems are pull based, rather than push based. Information has to be requested from the server when it is needed. One can change this by auto refreshing pages but it will yet be a pull based system which means it is static.

- Require powerful servers: Weather information is generally stored on powerful servers with fast processors and huge amount of system resources like memory and data transfer equipment. Since processing is done on the server, the data is processed on the server and an image consisting rendering of color for the region based on the temperatures at different points is created. This image is then sent to the client browser, to be displayed. The problem, with this approach, is that for very small changes in a particular region, the whole image will have to be reproduced and displayed at the client browser. Also, this approach is not dynamic. The display cannot be changed as data streams in.

Example: Current temperatures printed from weather.com

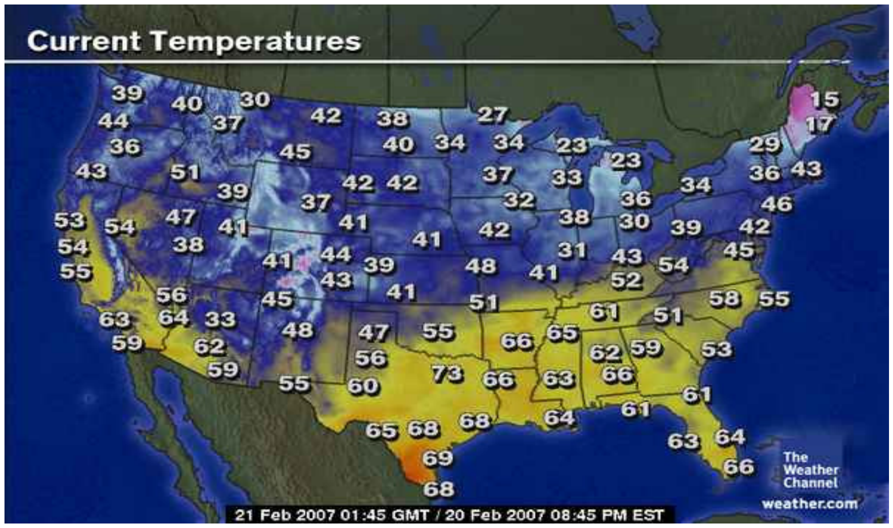


Figure 3.2 Image of temperature data rendering

- Traditional DBMS techniques cannot be used: Data rates into the system can exceed 1 million records per second, traditional databases cannot be used to run queries on this data because as is not feasible to insert such large volumes of data into DBMS and run queries on it in a timely manner. We have to use other techniques to process these streams quickly and release the system's resources, like buffers, for the next set of data coming in.

- It cannot be used to get information out of indoor locations: The general weather monitoring systems cannot be used to visualize areas of specific interest to us like a battle field or a nuclear power station. Deploying sensors give us more insight into these regions.

- It is not scalable: In most cases the middleware and applications are tightly coupled to the backend architecture. This makes is difficult to add more services on top of an underlying system. Such systems cannot scale well to add more services in the future.

3.2 Motivation behind new framework

Let us have a look at some of the motivations in building the new framework. It is important to understand these parameters because they heavily influence the design criteria of individual components that make our framework. While building the system we paid attention to the following key ideas.

i. Dynamic instead of static: The system should be able to handle incoming data and change dynamically as we observe some variation in temperature within the habitant being monitored. Example: if we are monitoring the state of TX and the temperature in DFW changes, only this part of the visualization should be updated, thus not requiring us to reload everything. This can be achieved by using AJAX in our web pages.

ii. Reduced resources and processing: The system should not require large amount of buffer space and advanced processing in order to process the incoming data and carryout visualization. In our experiments we will see how we use normal desktop systems and carry out computation in very small time intervals using approximation techniques.

iii. Handle large amounts of streaming data: Deployed sensor networks provide us with a large amount of data which arrives in a continuous stream rather than static relations. Not only are the size of these applications unbound, but the data arrives in a burst mode which makes it very difficult to process all the data in a timely manner. Due to the high data rates coming into the system, it can soon overflow the system's buffers. If data is not processed timely, it this may result in data being overwritten by the next set of data being written on to the limited buffer space available.

iv. Provide a framework to bring more services together: One of the basic motivations of this architecture was to build a base system which can be extended by using services like mash-ups (which brings together data from multiple sources on to one web page or source) so that the system can evolve over time. Example: Consider that we are displaying temperature information for regions. This information can be linked to an ecosystem and predict the impact of changes of temperature on animals in the zoo using some data available to researchers.

v. Run faster queries: If aggregates are to be calculated and displayed for a region in real time, we need to run queries in milliseconds so that the information can be sent to the client browser in a timely manner. We use techniques like simple random sampling of data and aggregations to achieve this goal.

CHAPTER 4

ARCHITECTURE

Let us look at the individual components of the system, on understanding which we can see how it all fits in the overall picture of the complete system architecture. The system can be divided into two parts

1. The front end: The front end contains two components.
    i. Google maps APIs.
    ii. AJAX.

2. The back end system: The back is made up of three components.
    i. Streaming processing
    ii. Sampling of data
    iii. Calculating results and aggregates

4.1 System components

4.1.1. Google Maps API

The Google Maps API was created by Google to facilitate developers integrate Google Maps into their web sites, with their own data points. By using the Google Maps API you can embed Google Maps on an external web site. Creating your own map interface involves adding the Google JavaScript code to your page, and then using JavaScript functions to add points and overlays to the map.

The advantage of using Google Maps is its rich set of APIs which can be used to program and create your own set of web pages with custom maps. Also, it provides different views like map view, satellite views etc to the region. It provides us some build in java scripts to carry out geo-coding which is a very important functionality in order to convert the string based location information of a sensor into discrete latitude and longitude positions on the surface of the globe. It has features to control the zoom levels depending on the area we can to cover and also mark regions on the map to color code them depending on the values of the temperature varying in that region.

Google Maps features a map that can be navigated by dragging the mouse or using the mouse wheel to zoom in (mouse wheel up) or out (mouse wheel down) to show detailed location information. The user is also able to control the map with the arrow keys to move to the desired location.

To run Google Maps, it has to be ensured that the browser is java script enabled. As in most Google applications, Google Maps relies on java script functions to geo-code addresses to latitude and longitude (called lat-long) positions to pin point the exact location on the surface of earth. Also, this approach ensures that heavy weight client applications are not created, which generally tend to be very demanding on the resources on the client system.

Using Google maps ensures that our mapping application, and visualization using overlays, will work on the following web browsers:

- IE 6.0+
- Firefox 0.8+
- Safari 1.2.4+
- Netscape 7.1+
- Mozilla 1.4+
- Opera 8.02+

4.1.2. Ajax

AJAX, which is a shorthand for "Asynchronous JavaScript and XML", is a web development technique for creating interactive web applications. The intent is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire web page does not have to be reloaded each time the user requests a change. This is intended to increase the web page's interactivity, speed, and usability.

As discussed earlier, one of the primary motivations of the system was to make it dynamic rather than static and AJAX helps us achieve that. We use a technique called overlays on Google Maps. The idea is that when a new set of data about some region arrives, only the particular overlay, where data change has occurred, can be modified without having to reload the entire page.

Using AJAX asynchronous requests can be sent to the server. When the server computes aggregates for some regions the AJAX reply is sent back to the client who, on receiving the reply, will update only the particular overlays that are affected by the new values. This way we can reduce the time required to color regions on the browser. Also, using a similar approach other functions can be carried out as well, like drawing graphs etc.

Having discussed the front end, let’s see how it works in the proposed system.

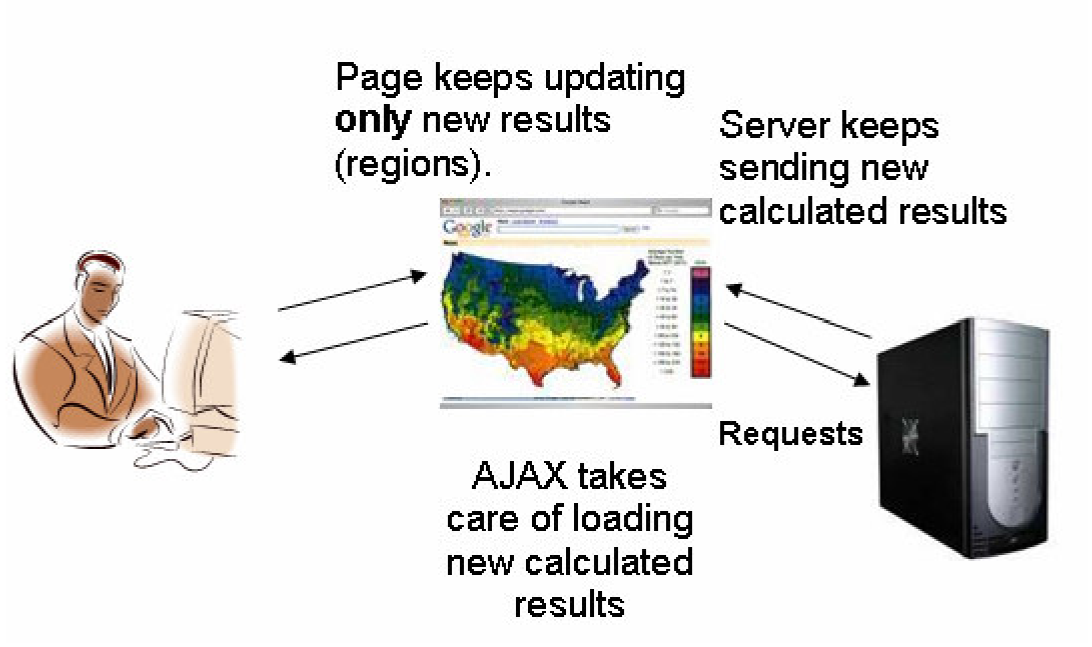

Page keeps updating **only** new results (regions).
Server keeps sending new calculated results
Requests
AJAX takes care of loading new calculated results

Figure 4.1 Example of how the front end works with AJAX

The steps that take place in the interaction between the client and server using the above approach are shown below:

i. The client browser make request for some information.

ii. The server calculates the result and sends it to the client, for display.

iii. The client sends an AJAX request to server, requesting for updates.

iv. On computing the aggregates for the newly arrived data on the server a response is sent to the client.

v. The client updates the respective overlay on the browser window without having to reload the entire page.

4.1.3. Stream processing and random sampling

The data arriving from sensor networks are in the form of continuous streams can be over whelmingly large in nature. This, as pointed out earlier, tends to be demanding on the system's resources. If incoming data is not processed timely, buffers get over written, resulting in data loss.

In order to speed up the processing of the data, in our system, we use techniques like simple random sampling of data which can be described as a group of subjects (a sample) chosen from a larger group (a population). Each subject from the population is chosen randomly and entirely by chance, such that each subject has the same probability of being chosen at any stage during the sampling process. This process and technique is known as Simple Random Sampling. In our system we pick out samples from the incoming streams and move them into sampling tables. Since the number of samples required is quiet small compared to the actual size of the stream the process of picking and transferring the samples can be very quick, thus letting free the buffer space for the next set of data arriving through the streams.

Let us have a brief look at the sampling process and understand its significance in the scenario of the system

Given the standard normal distribution below

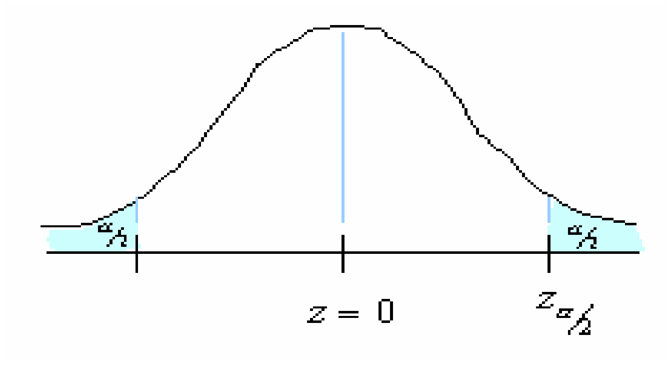


Figure 4.2 Standard normal distribution

For the distribution first we determine the critical value, which is $\mathbf{Z}\boldsymbol{\alpha}_{/2}$. For a sample to be accepted its absolute values should be grater than the critical value so that the null hypothesis can be rejected.

Now if '**σ**' is the standard deviation and '**n**' is the sample size and '**Z**' is the positive vertical boundary. The margin of error (**E**) is given by:

$$E = Z\alpha_{/2} \ . \ \sigma / \sqrt{2}$$

Rearranging the formula we can get '**n**'

$$n = [\, Z\alpha_{/2}\ .\ \sigma / E \,]^{2}$$

The above equation can be used when the size of the population is not known.

For a finite population the formula becomes.

$$(\text{New})\ n = n / 1 + ((n - 1) / \text{population})$$

Using this formula we calculate the sample size for the data chunk that arrives. In our system for simplicity we have assumed that the data arrives in know sizes which are a multiple 100 thousand records. There can be multiple streams each with a different stream size. Also, we calculate a sample size for a minimum size stream (100 thousand records in our case) and give proportionate weight age to other streams which are a multiple of 100 thousand say 200 thousand records or 1 million records. By doing this we ensure that all streams have equal weight age or contribution in the final answer that is calculated. Streams appearing early in the timeline should not have an advantage over the streams arriving late or the streams that are smaller in size.

By placing this restriction we are ensuring that we have uniform random distribution of data over the universal set of the sample table data. This way every sample item will have uniform probability of getting picked and our output will not be biased towards smaller or larger streams.

When the data arrives into the system through streams we program the system I/O to transfer the data into a buffer. Then the samples are picked using simple random sampling and then the buffers are freed for the next set of data that arrives. The samples that get picked are written on to a sample table which is used to carry out aggregations. Once the new aggregates for regions are computed they are sent to the client through the AJAX messages and the client browser updates the respective overlays on the display.

The advantages of using simple random sampling are:

i. Simple and fast: Since the algorithm is not very complex the computation can be carried out very quickly, which is a very important consideration.

ii. Free of classification errors: Simple random sampling does not involve any grouping or clustering to be done before the sampling process is carried out. So we eliminate any risks of grouping and classification errors.

iii. Requires minimum advance knowledge of data: The sampling process does not require us to have any advance knowledge of the data that will arrive. This is ideal to our scenario because we cannot predict the changes in temperature in the regions being monitored. They can vary arbitrarily based on multiple factors.

iv. We can handle multiple streams at once: Since the sample picking process in not biased towards the size of the stream or the time at which it occurs we can process multiple data streams coming from different sources.

Disadvantages of using simple random sampling:

i. 100 percent accuracy not possible: Since we are always picking samples from a universal set we cannot, theoretically, attain 100 percent accuracy. The error here is non-deterministic so there might be one off cases where the results may be accurate but the same process cannot be repeated again to get accurate answers.

Is the loss of accuracy a real concern? Upon understanding the merits of this approach one can accept the small reduction in accuracy in order to achieve reduction in the processing time and increased query execution speeds that we get. In the experiment and analysis section we will see how the error in the system is reduced once a sizable amount of sample is collected.

4.1.4. The complete framework

Having looked at the individual components, let's take a look at the complete picture:

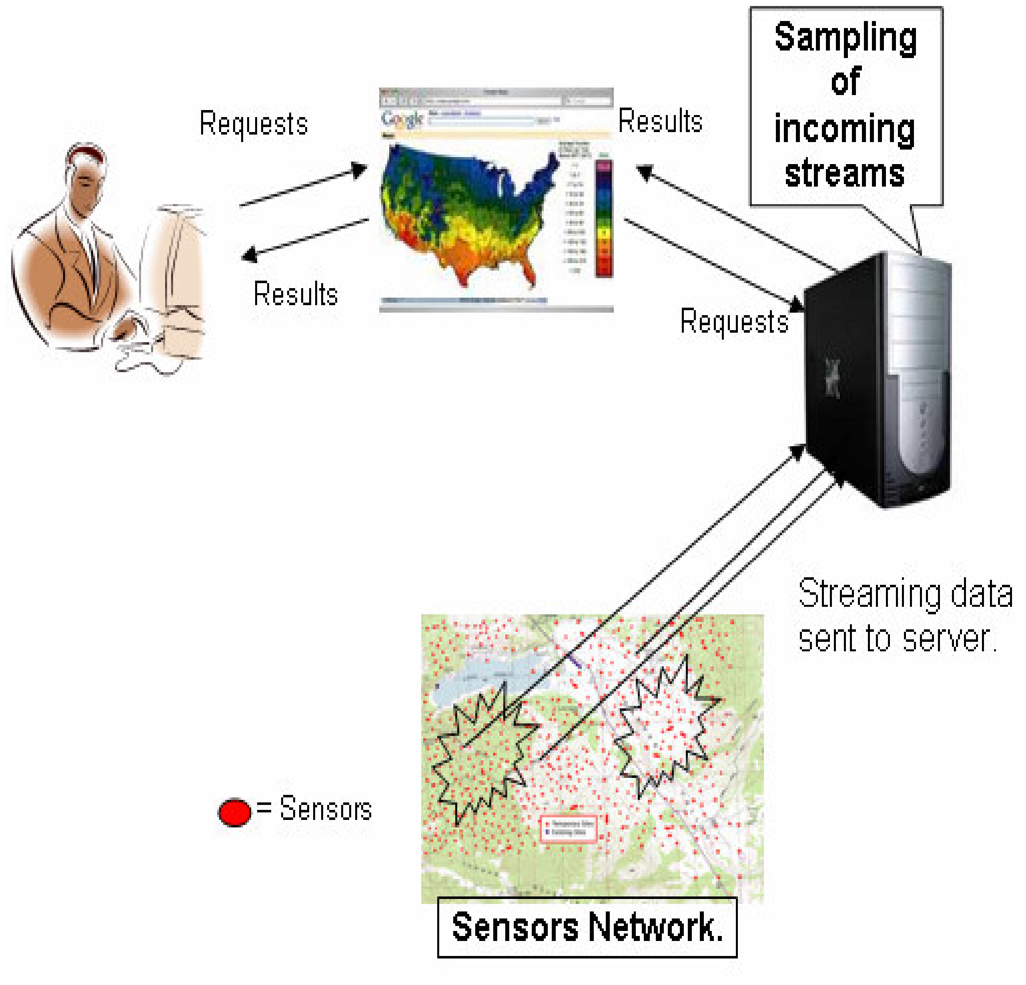


Figure 4.3 A representation of how our framework works

This is how the whole system works:

i. The user requests for temperature information from the server.

ii. The server is set up to receive continuous streams of sensor network data from a deployed habitant.

iii. The server, on receiving a new set of data, carries out simple random sampling on the data and calculates aggregates for the respective regions.

iv. The new value is sent to the client browser which displays the information.

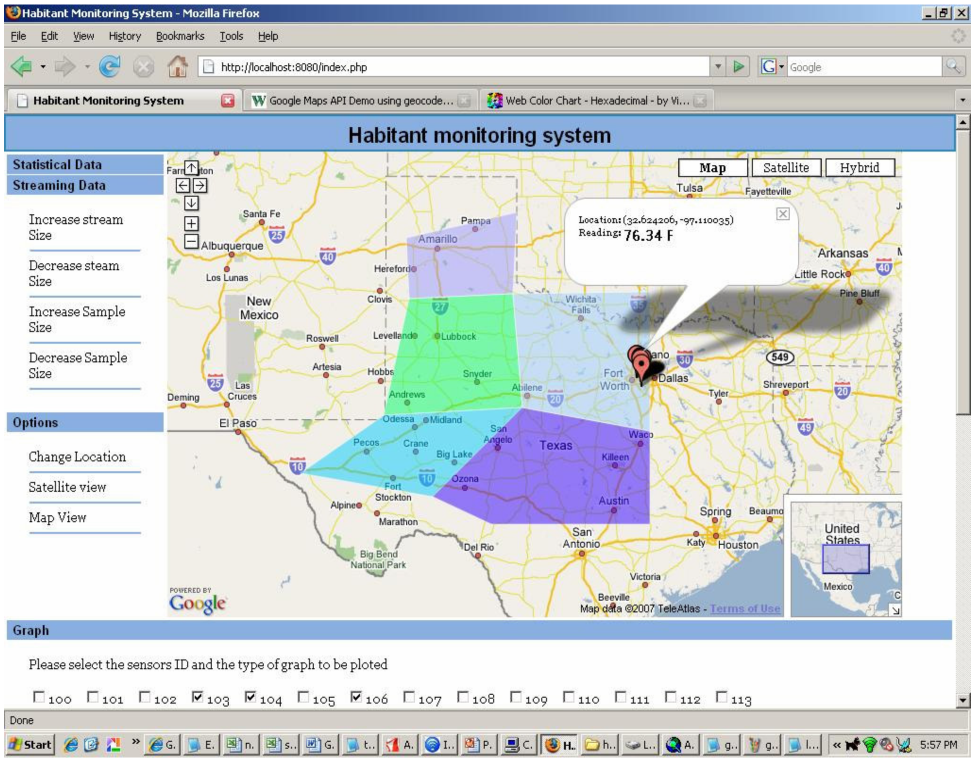
v. The client browser sends AJAX request for new values.

vi. As the new set of data arrives the aggregates are calculated by the server and the new values are sent to the client through an AJAX response message.

vii. The client browser, on receiving the message, updates the respective overlays without having to update the whole web page.

4.1.5. Some screen shots

1. The screen shot of the system showing some regions in Arlington, TX along with color codes for respective zones based on the average temperatures there. The color code represents the average temperature in that region. Since the temperatures were randomly generated the color changes drastically. If actual data is used the color change will be much smoother. On hovering over a particular overlay we can see the average temperature of that region being displayed. The tiny marker, below the display box, represents a point (lat, long) at which the reading is being taken.


Habitant Monitoring System - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/index.php
Habitant Monitoring System
Google Maps API Demo using geocode...
Web Color Chart - Hexadecimal - by Vi...
Habitant monitoring system
Statistical Data
Streaming Data
Increase stream Size
Decrease steam Size
Increase Sample Size
Decrease Sample Size
Options
Change Location
Satellite view
Map View
Map
Satellite
Hybrid
Graph
Done
Start
5:57 PM

Location: (32.624206, -97.110035)
Reading: 76.34 F

Graph

Please select the sensors ID and the type of graph to be ploted

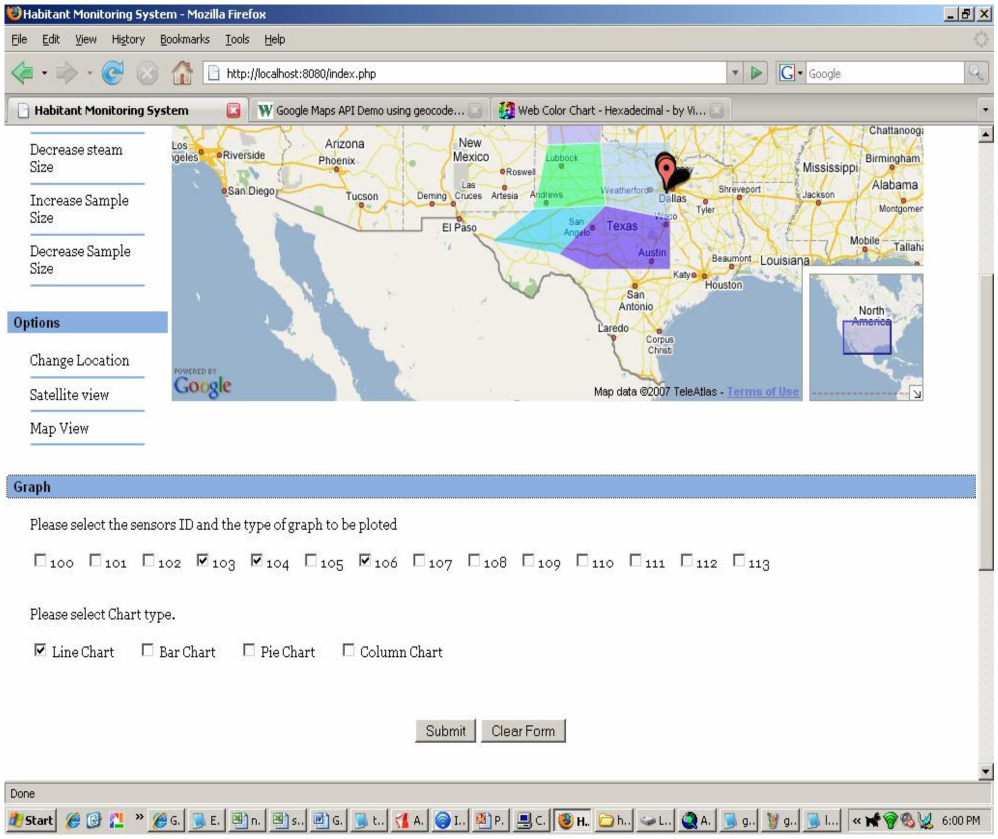
☐ 100 ☐ 101 ☐ 102 ☑ 103 ☑ 104 ☐ 105 ☑ 106 ☐ 107 ☐ 108 ☐ 109 ☐ 110 ☐ 111 ☐ 112 ☐ 113

Figure 4.4 Snapshot of overlays

2. Options for drawing graphs

If we are interested in drawing graphs for a particular region, we can choose the points that are of interest to us along with the type of graph desired. The data for the graphs can be pulled asynchronously, thus avoiding page reloads every time the values on the graph changes.

This approach can be used to add other services like calculation temperature variations and other calculations that can be of interest to the user in the future.



Habitant Monitoring System - Mozilla Firefox

http://localhost:8080/index.php

Decrease steam Size
Increase Sample Size
Decrease Sample Size

Options

Change Location
Satellite view
Map View

Graph

Please select the sensors ID and the type of graph to be ploted

[ ] 100 [ ] 101 [ ] 102 [✓] 103 [✓] 104 [ ] 105 [✓] 106 [ ] 107 [ ] 108 [ ] 109 [ ] 110 [ ] 111 [ ] 112 [ ] 113

Please select Chart type.

[✓] Line Chart [ ] Bar Chart [ ] Pie Chart [ ] Column Chart

File Edit View History Bookmarks Tools Help
Habitant Monitoring System
Google Maps API Demo using geocode...
Web Color Chart - Hexadecimal - by Vi...
Submit
Clear Form
Done
Map data ©2007 TeleAtlas - Terms of Use

Figure 4.5 Check boxes for choosing graph points

3. Some graphs for the temperature

The screen shot below shows graph for a particular region once the options are chosen (shown in the earlier screen shot). Since both the graph are the same a similar curve is seen. Too much of effort has not been devoted to build graphing applications in this thesis as embedding graphs in the web page is a simple implementation with a lot of material for this available on the Internet. It can serve as add on to a system. We will limit our discussion to our unique contributions.

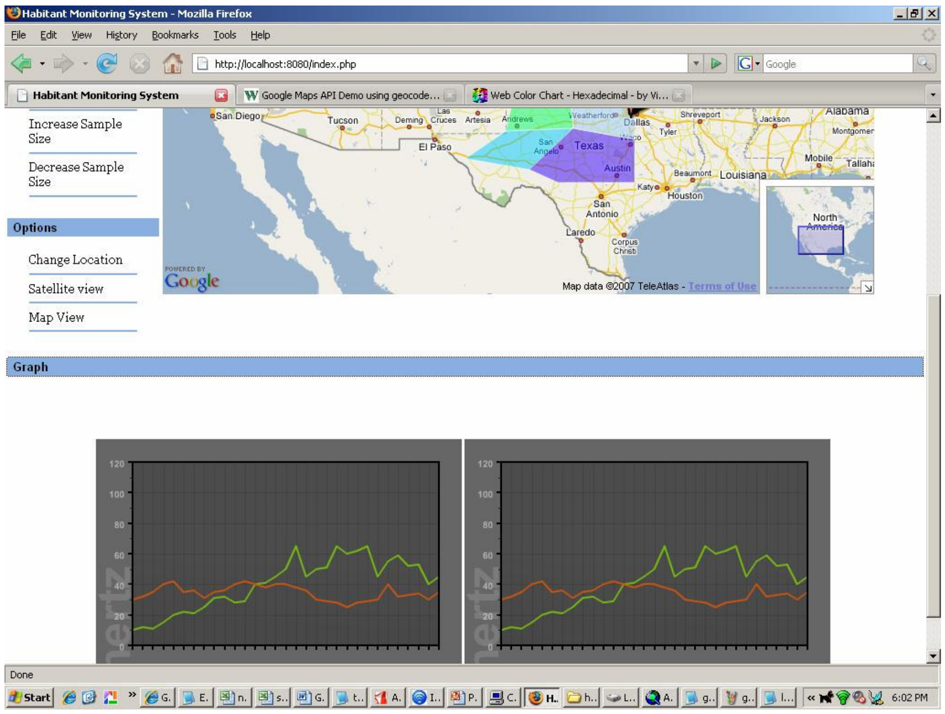

Habitant Monitoring System - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/index.php
Habitant Monitoring System
Google Maps API Demo using geocode...
Web Color Chart - Hexadecimal - by Vi...
Increase Sample Size
Decrease Sample Size
Options
Change Location
Satellite view
Map View
Graph
Map data ©2007 TeleAtlas - Terms of Use
Done
Start
6:02 PM

Figure 4.6 Using graphs in the system

CHAPTER 5

EXPERIMENTS AND ANALYSIS

In this chapter we discuss the experiments and some analysis carried out on it. The basic web pages were built using PHP and AJAX. The server side programming was done using java language. Experiments were run on a lab machine with server and client as http://localhost/ or 127.0.0.1. The machine used for this purpose is an Intel Pentium Duo core 2.7 G. Hz. processor with 512 Mb of RAM. The operating system is Microsoft Windows XP, service pack 2 with x86 32 bit native binaries.

5.1 Experiment methodology

All experiments were run multiple times at different periods of the day to make sure that the behavior is consistent. The experiments for calculating the skew of each reading were performed multiple times and the average values were recorded so that we can eliminate one off results which tend to scatter along the curve with wide variations. As pointed out earlier, the error in the system is non-deterministic so some aggregates that are calculated may end up being 100 percent accurate but this is just ‘by chance’ and cannot be repeated if the exact same steps are followed again.

5.1.1 Sample consistency test

Let us look at some experiments to prove that the samples are consistent in nature.

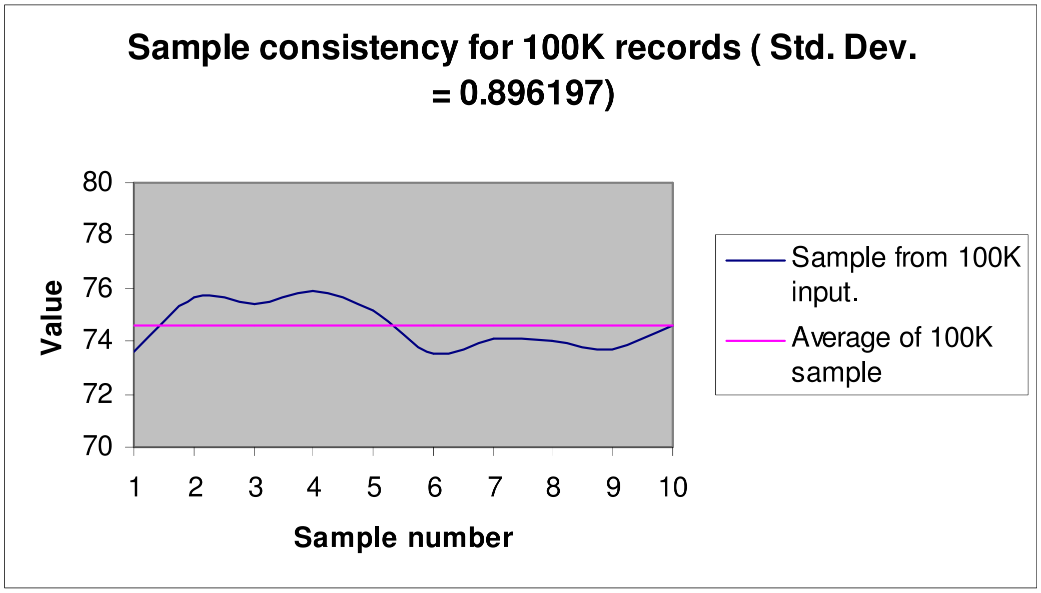


Figure 5.1 Sample consistency test with 100 K samples

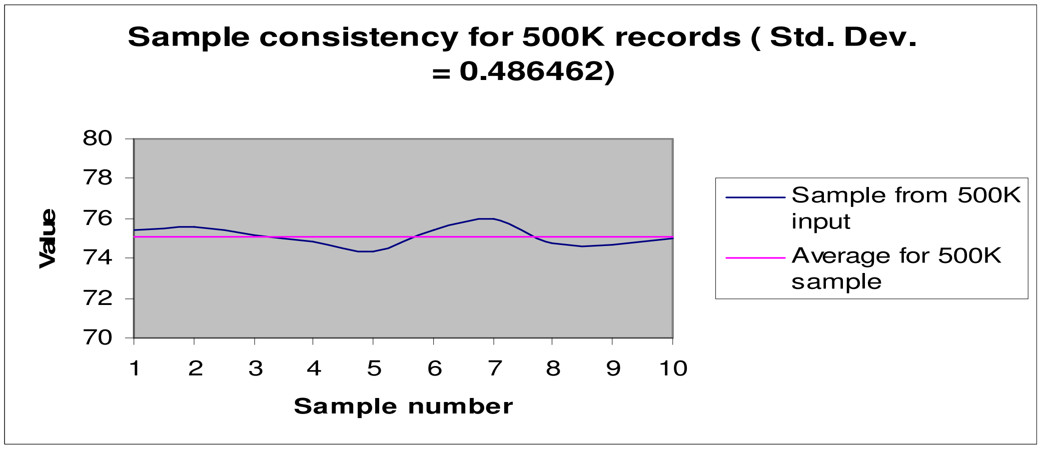


Figure 5.2 Sample consistency test with 500 K samples

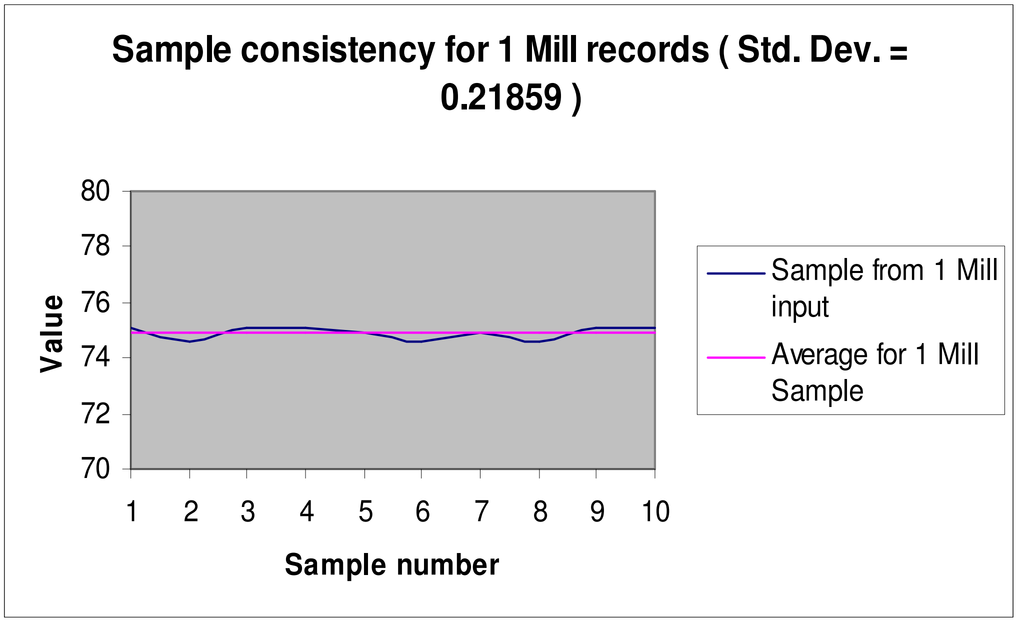


Figure 5.3 Sample consistency test with 1 Million samples

To check for consistency, samples for inputs of various sizes were taken and their aggregates calculated, and the values recorded. The sample table is then discarded and a new table is built picking a new set of samples. This process is repeated to get different set of values. The distribution for 10 such iterations vs. the average of the 10 iterations can be found in the graphs shown above. It is interesting to observe as the sample size increases the variation in the results (standard deviation) reduces. Also, the consistency of the sample increases as the sample size increases. One important point to be noted here is, mathematically speaking; it is very difficult to prove the consistency of the data with 100 percent confidence. Nevertheless, the experiments and graphs above give us a definite indication of the kind of behavior to expect from the data collected.

5.1.2. Skew in the data

Let's take a look at the skew observed in the data for both static and dynamically changing data.

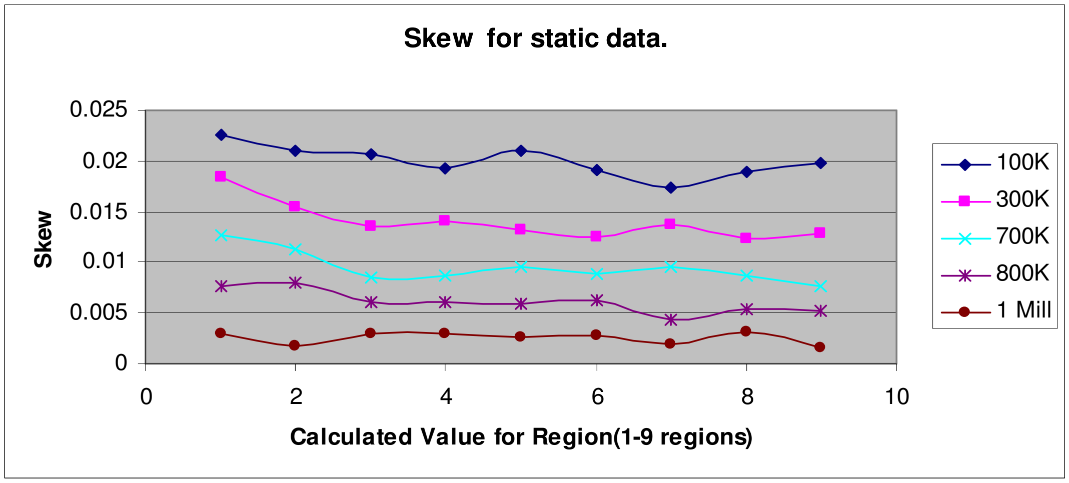


Figure 5.4 Skew for static data of different sizes

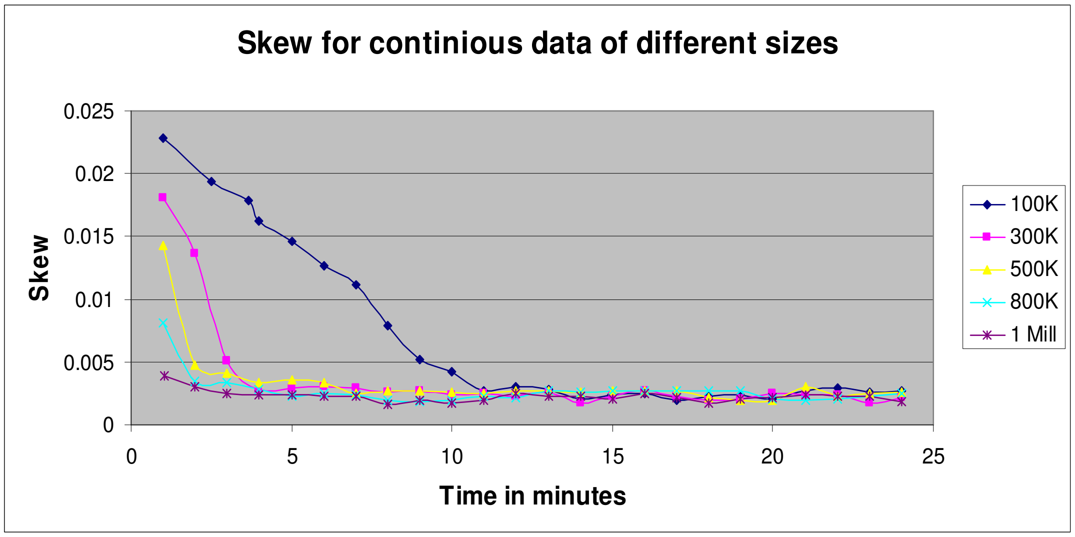


Figure 5.5 Skew observed for dynamically changing data streams of different sizes

From figure 5.4 we clearly see that as more samples are collected in the system, the error rate decreases. The streams of smaller sizes take a long time to reduce the errors because they have to collect enough samples for the sampling process to get efficient. Also, from the nature of the graph we clearly observe that beyond a certain point, the accuracy of the system varies, and stays, around a certain value. The system, theoretically speaking, cannot get more efficient than 99 percent. So on reaching these levels they vary around the same range.

5.1.3. Time taken to compute aggregates for different regions

As you can see it takes only a fraction of a second to compute the aggregates for different regions using the tables built by the sampling process. We observe a huge reduction in time compared to databases as we are doing away with database insert operations which can be expensive in terms of the time taken.

The sample tables are built in main memory and the query is executed on them. Once the sample table size gets very large in nature, the older records can be removed and stored, asynchronously, in a database system. This also gives us an advantage of being able to retrieve old tables in the future for comparison purposes.

Input Size	Time to cal aggregate (in ns)	Time to build sample table from buffer (in ns)
100K	521994	425207
200K	1183536	767389
300K	1527262	1727808
400K	1752461	1449254
500K	2074481	2252460
600K	2287410	2104830
700K	2547390	2364716
800K	2850053	2652902
900K	3176503	3287654
1 million	3482338	3500371

Figure 5.6 Time taken to compute the aggregates for different input sizes

CHAPTER 6

CONCLUSION

Through this thesis we presented an approach for processing large volumes of streaming data from sensor networks and visualizing them dynamically using a web based system in a power efficient and timely manner.

As pointed out earlier this framework can be extended to incorporate more features by building more services on top of the base service. We have discussed and analytically demonstrated several properties of our framework with regard to its behavior. Also, through a detailed experimental study we have demonstrated the how things work with this approach.

REFERENCES

[1] Service Oriented Sensor Web.

Xingchen Chu and Rajkumar Buyya.

White paper on Sensor Web Enablement (SWE) standard defined by the OpenGIS Consortium (OGC), 2006.

[2] A Flexible Web Service Based Architecture for Wireless Sensor Networks

Flavia Coimbra Delicato, Paulo F. Pires, Luci Pirmez, Luiz Fernando Rust da Costa Carmo, 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03) p. 730.

[3] A service-oriented model for wireless sensor networks with Internet.

Jinglun Shi Weiping Liu**,** The Fifth International Conference on Computer and Information Technology. CIT 2005, page(s): 1045- 1049.

[4] STREAM: The Stanford Stream Data Manager.

IEEE data engineering bulletin. 2003.

[5] Dynamic Load Distribution in the Borealis Stream Processor.

Y. Xing, S. Zdonik, J.-H. Hwang.

In proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan, April 2005

[6] Implementing Software on Resource-Constrained Mobile Sensors: Experiences with Impala and ZebraNet (2004)

Ting Liu, Christopher M. Sadler, Pei Zhang, Margaret Martonosi

Proceedings of the 2nd international conference on Mobile systems, applications, and services, 2004.

[7] Milan: Middleware Linking Applications and Networks

Amy Murphy, Wendi Heinzelman

University of Rochester technical report. 2002.

[8] Mires: a publish/subscribe middleware for sensor networks

Eduardo Souto, Germano Guimarães, Glauco Vasconcelos, Mardoqueu Vieira , Nelson Rosa, Carlos Ferraz, Judith Kelner

Journal Personal and Ubiquitous Computing

Issue Volume 10, Number 1 / February, 2006

Pages 37-44

[9] High Availability Algorithms for Distributed Stream Processing.

J.-H. Hwang, M. Balazinska, A. Rasin, U. Çetintemel, M.Stonebraker, S. Zdonik.

In proceedings of the 21st International Conference on Data Engineering (ICDE'05), Tokyo, Japan, April 2005.

[10] Hourglass: An Infrastructure for Connecting Sensor Networks and Applications.

Jeffrey Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, Matt Welsh.

Harvard Technical Report TR-21-04.

[11] A sensor Web language for mesh architectures

Nickerson, B.G.; Sun, Z.; Arp, J.-P.

Communication Networks and Services Research Conference, 2005. Proceedings of the 3rd Annual Volume , Issue , 16-18 May 2005 Page(s): 269 – 274

[12] Continuous Queries over Data Streams.

S. Babu and J. Widom.

In SIGMOD Record, Sep. 2001

[13] Resource Sharing in Continuous Sliding-Window Aggregates

A. Arasu and J. Widom.

In Proc. of VLDB 2004, Sep. 2004

[14] Aurora: A New Model and Architecture for Data Stream Management.

D. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C.Convey, S. Lee, M. Stonebraker, N. Tatbul, S. Zdonik.

In VLDB Journal (12)2: 120-139, August 2003.

[15] http://en.wikipedia.org/wiki/Google_Maps

[16] http://en.wikipedia.org/wiki/AJAX

[17] Online Aggregation

Joseph M. Hellerstein, Peter J. Haas, Helen J. Wang

SIGMOD Conference, 1997

[18] Practical Selectivity Estimation through Adaptive Sampling

Richard J. Lipton, Jeffrey F. Naughton, Donovan A. Schneider

ACM SIGMOD Conference, 1990.

[19] Design of a Web-Based Application for Wireless Sensor Networks

Sajid Hussain, Nick Schofield, Abdul W. Matin.

17th International Conference on Database and Expert Systems Applications (DEXA'06)

pp. 319-326

[20] Simultaneous Web-based real-time temperature monitoring using multiple wireless sensor networks

Hayes, J.; Crowley, K.; Diamond, D

Sensors, 2005 IEEE

Volume, Issue , 30 Oct.-3 Nov. 2005 Page(s): 4 pp

[21] A distributed agent system for managing a web-based sensor network with field servers

Tokihiro FUKATSU, Masayuki HIRAFUJI, Takuji KIURA

Computers in Agriculture and Natural Resources, 4th World Congress Conference, Proceedings of the 24-26 July 2006 (Orlando, Florida USA)

[23] Implementation techniques for main memory database systems

D. DeWitt, R. Katz, F. Ohlken, L. Shapiro, M. Stonebraker, and D. Wood.

In Proc. of the ACM SIGMOD Conf. on Management of Data, pages 1--8, 1984

[24] Processing Aggregate Relational Queries with Hard Time Constraints

W. Hou, G. Ozsoyoglu, and B.K. Taneja

Proc. ACM SIGMOD Int'l Conf. Management of Data, 1989.

[22] SIA: Secure Information Aggregation in sensor networks

Bartosz Przydatek, Dawn Song, Adrian Perrig

SenSys’03, November 5–7, 2003, Los Angeles, California, USA.

BIOGRAPHICAL INFORMATION

Sunil Pai was born in Karnataka, India in 1982. He received his BE in Information Science & Engineering from SDMCET, Visweswaraiaha Technological University in 2005. His constant interest on web based systems & computer networks influenced him to pursue his study on computer networks while doing masters at UT Arlington.