Fall 2007

Distributed Cluster-Based Outlier Detection in Wireless Sensor Networks

Swetha Gali
Old Dominion University

DISTRIBUTED CLUSTER-BASED OUTLIER DETECTION IN WIRELESS SENSOR NETWORKS

by

Swetha Gali
B.E. (E.E.E) June 2005, MVSREC, Osmania University, India

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
December 2007

Approved by:

Min Song (Director)

Linda L. Vahala (Member)

Sachin Shetty (Member)

ABSTRACT

DISTRIBUTED CLUSTER-BASED OUTLIER DETECTION IN WIRELESS SENSOR NETWORKS

Swetha Gali
Old Dominion University, 2007
Director: Dr. Min Song

Wireless sensor networks find several potential applications in a variety of fields, such as environmental monitoring and control, battlefields, surveillance, smart buildings, human health monitoring, etc. These sensor networks consist of a large number of very tiny, inexpensive, and low power sensor nodes, which are deployed in a variety of harsh environments that may result in the sensor data getting corrupted. It is thus critical to detect and report these abnormal values in the sensor data, in order to have a better understanding of the monitored environment. Detection of the abnormal values is of special interest for the sensor network research community because it helps in the identification of certain events of interest, malicious activities, or faulty nodes in the network.

In contrast to a centralized scheme of outlier detection, a distributed approach deals with the detection of outliers at the granularity of the sensor nodes itself. This thesis addresses the problem of outlier detection in a sensor network scenario. The main contribution of this thesis is to develop a distributed cluster based outlier detection algorithm for sensor data. At each node, the local data is first partitioned into a fixed number of clusters based on the principles of the popular *k*-means clustering paradigm. The cluster summaries are then sent to few higher capacity nodes in the network. It is at these high capacity nodes that the outlier clusters are detected using the *K*-nearest

neighbor algorithm by identifying the K closest clusters. The technical challenge, however, is to be able to achieve the goal of outlier detection in the network, i.e. at the node level itself. This is because the transmission of raw data from all the nodes to a centralized location for further analysis may result in increased energy consumption and communication overhead. In order to address the issue of achieving outlier detection in sensor data with minimized energy and communication costs, a distributed scheme of outlier detection is used. In such a scheme, only a summary of cluster information is being transmitted across the network, rather than the entire set of raw data. This greatly reduces the number as well as the size of messages being transmitted across the network and, thereby, results in reduced communication costs. A set of simulation experiments were performed and results obtained. The performance of the proposed approach is then studied based on varying a set of system parameters.

This thesis is dedicated to Jesus Christ, my Lord.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor, Dr. Min Song, for his support and guidance during my research. My utmost appreciation also extends to my co-advisor, Dr. Sachin Shetty, for his limitless support, patience, and guidance and especially for his help during the simulation runs for this thesis. I would also like to thank my committee member, Dr. Linda L. Vahala, for consenting to be on my thesis advisory committee.

I would like to extend my special appreciation to my mother Mrs. Elizabeth Rani, my father Mr. Showri Reddy, my brother Karthik, my aunt Mrs. Mary Showrilu, my uncles Mr. Marreddy Allam, and Mr. Bhasker Allam and my well wishers Mrs. Kathy Hardison, and Mr. Kurnia Foe for being a great source of moral support. I would also like to express my gratitude to my friends Lalitha, Diana, Vijetha, Madhuri, Kirthi, Nandini, and Swetha for being there for me throughout and helping me in the preparation of this manuscript. Lastly, I would like to thank Mr. Sarath Chandra for his immense help with coding in JAVA.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER I

INTRODUCTION

The area of wireless sensor networks (WSNs) is constantly being addressed by the research community today, due to the various applications they find in several areas of human and machine deployments. In some instances, a sensor network may be deployed in harsh environmental conditions. This may lead to erroneous data, which demands immediate attention for the network to work without intervention. Thus, detection of abnormal values becomes a prime concern in sensor network research. This chapter deals with a brief introduction and background on the concept of wireless sensor networks and detection of outlier values in sensor data. The challenges and contributions of this thesis are also discussed.

1.1 Background

The area of research dealing with the detection of abnormal values in a variety of systems caught the interest of the research community in the 1990's. But the main emphasis then was on the determination of outliers in the fields of statistics, machine learning, and databases. Outlier detection in sensor data has emerged as an active research topic only in the current decade. Detecting outliers in a sensor network will help in identifying any faulty sensor nodes, malicious activities, or misbehaviors in the network. Sensor networks, which are a type of peer-to-peer (P2P) networks, have

themselves evolved as a vast area of research in the past few years with significant emphasis on several issues concerning them. These issues include routing algorithms, data transmission, energy and communication constraints, network dynamics, architecture and topology, application protocols, information processing and data analysis, optimization of energy, and communication costs, etc.

A critical factor that drives outlier detection research in the field of sensor networks is the location of data processing. This means that the choice of the location in the network where the processing of data is carried out affects the performance of the network to a large extent. Performance may be determined with respect to factors like time required for data processing, communication overhead and costs, energy costs, and the like. While detection of the erroneous data values seems to be a critical issue by itself, it has also become important to be able to achieve this objective in the network, i.e. at the node level. This, however, poses several challenges, due to the large number of sensor nodes in the network as well as the nature of sensor data itself. This thesis attempts to address the problem of detecting abnormal sensor values (termed as outliers), by performing data processing at the node level in a distributed manner based on a clustering algorithm.

1.2 Wireless Sensor Network Characteristics and Applications

In recent years, sensor networks have been finding several potential applications in various fields. A number of different settings, such as environmental monitoring and control, intrusion detection in buildings and cars, health monitoring and control, traffic monitoring and control, battlefields, weather forecast, etc., have been successfully

deploying sensor networks. Sensor networks may be installed in wide geographical areas and can be used for monitoring or detecting various events of interest. In such scenarios, sensor networks are generally used for high-level data processing tasks. Wireless sensor networks typically consist of spatially distributed autonomous devices using tiny, light-weight, inexpensive, and low power sensors, which monitor environmental parameters, such as temperature, pressure, light, sound, etc. These devices, which host the sensors on them, are called nodes and are connected via wireless links to each other or to a more powerful base station. The base station is in turn connected through either wired or wireless links to base stations of other networks or to a computer processor. It is very likely that these sensor nodes work for long periods of time generating large amounts of streaming data. In such instances, it is also possible that these sensor nodes fail or report faulty values due to several reasons, such as deterioration of power or low fidelity of the nodes. Some of the key reasons for erroneous data in sensor networks are listed below [1]:

- Harsh environments in which a sensor network is deployed
- Interferences in the wireless medium
- Sleeping modes of the sensors
- Cheap and low quality sensors.

The abnormal, unreal, or faulty values reported by the sensor nodes are termed as 'outliers' or 'anomalies.' An outlier has been formally defined by Hawkins in [2] as "an observation, which significantly deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism." In general statistical terms, outliers are those values that significantly deviate from the norm. The abnormal values in

a sensor network scenario are also termed as 'misbehaviors' in [3] by Rajasegarar et. al. In a majority of cases, the abnormal values reported by sensor nodes can cause errors in sensing queries and sensing data analyses. Hence, it is critical to detect the outliers in the sensor data and either repair them or completely remove them from the data set according to the application requirement.

It is also important that the processing of sensor data is done in the network itself rather than at a centralized location due to several reasons. Transferring the entire raw data from individual sensor nodes to a centralized base station is not desirable because it increases the communication overhead and network traffic. This results in an increase in the energy consumption in the network. Communication overhead and network traffic mentioned above relate to the number of messages or packets being transmitted across the network from one node to another. A desirable solution would be to develop a system that deals with the outlier detection task in the network itself, i.e. at the node level in a distributed manner.

1.3 General Clustering

The process of clustering organizes a set of patterns, usually vectors in a multi-dimensional space, into coherent and contrasted groups, such that patterns in the same group are similar in some sense and patterns in different groups are dissimilar in the same sense [4]. In [5], Han and Kamber define clustering as a process of grouping the data set into classes or clusters so that objects within a cluster are more similar to each other but are very dissimilar to objects in other clusters. Distance measures are usually used to determine the dissimilarity between objects; this dissimilarity is based on the attribute or

feature values describing the objects. Clustering the data helps in identifying the dense and sparse regions in a data set and, in turn, the overall distributions and interesting correlations among various data attributes or features. This information can later be used to observe the characteristics of each cluster and to focus on a particular set of clusters for further analysis. The application of cluster analysis described above is useful for the identification of outliers in the data.

There are several approaches developed by statistics, machine learning, and databases communities for the purpose of clustering their data. Partitioning methods, density-based methods, hierarchical methods, model-based, and grid-based methods are some of the major classifications of clustering methods popularly used. More recently, clustering has also found its way into the sensor networks community for either clustering or grouping the nodes and/or data at the nodes.

A brief description of several popular clustering techniques is provided in [5] by Han and Kamber. Among the most popular techniques are the *k*-means, *k*-medoids, and CLARANS methods used for partitioning. BIRCH, CURE, and Chameleon are among the hierarchical category of clustering methods. DBSCAN, OPTICS, and DENCLUE fall in the density-based methods category. STING, WaveCluster, and CLIQUE are among the grid-based techniques. Model-based approaches include neural network methods and statistical methods.

1.4 General Outlier Detection

Outliers, as discussed earlier, are those values that deviate significantly from the norm. In [2], Hawkins describes two mechanisms that may give rise to samples termed as

outliers. The second mechanism discussed by him defines data arising from two distributions. 'Basic distribution' generates 'good' observations, whereas 'contaminating distribution' generates 'contaminants.' In [6], Barnett and Lewis deal with a different version for the definition of outliers. These authors suggest that "in almost every true series of observations, some are found, which differ so much from the others as to indicate some abnormal source of error not contemplated in the theoretical discussions." It thus becomes very important for every system dealing with data collection and processing to have a good outlier detection mechanism installed in it.

Petrovskiy discusses outlier detection algorithms used in data mining systems in [7]. Various methods of outlier detection are presented. These include statistical methods, distance-based approaches (such as the *K*-nearest neighbor algorithm), kernel functions-based approaches, and fuzzy approaches with the use of kernel functions, etc.

An elaborate discussion on mining distance-based outliers in large data sets is made by Knorr and Ng in [8]. An object *O* in a data set is a *DB*(*p*, *D*)-outlier if at least a fraction *p* of the objects in the entire data set lies at a distance greater than *D* from *O*. Several algorithms such as the index-based algorithm, nested-loop algorithm, and cell-based algorithm are described.

In [9], authors Ramaswamy et. al. deal with some algorithms for mining outliers from large data sets. A novel approach for a partition-based algorithm is discussed. This approach involves an initial partitioning of the input data space and then pruning the partitions as soon as it is discovered that they cannot contain outliers. Any clustering algorithm can be used to partition the input data set depending on the application requirements.

1.5 Cluster-Based Outlier Detection

This thesis proposes a cluster-based outlier detection approach for a sensor network environment. The details of the approach that is based on *k*-means technique for clustering the sensor data and later finding the *K* closest clusters for detecting outlier clusters among them, are provided in Chapter III of this report. Also, a detailed description of previous attempts made by researchers to achieve the objective of outlier detection in sensor networks using clustering and other data mining techniques is presented in the related work section in Chapter II of this report.

1.6 Challenges of this Research

A critical point to consider in a sensor network scenario is that the distribution of data is not known a priori. This makes it difficult to model the sensor data. Hence, identifying outliers in data without prior knowledge about its input distribution poses an important challenge for this research.

Also, the fact that the communication costs are more than the computation costs in sensor networks presents another challenge for this research. Thus, while designing new algorithms for sensor networks, it is important to consider their energy, communication, and power limitations.

While the process of detecting erroneous values seems to be a critical issue by itself, it is also important to achieve this in the network, i.e. at the node level. This, however, poses several challenges due to the large number of sensor nodes in the network as well as the nature of sensor data itself.

Finally, sensor nodes produce streaming data, and this poses another challenge for data processing in a sensor network. To accommodate this problem, a sliding window of a certain number of measurements is considered for this thesis.

1.7 Contributions of this Thesis

The main contribution of this thesis is to present a distributed cluster-based outlier detection system in a sensor network environment. The methodology described, is based on the principles of the popular *k*-means clustering paradigm. A distributed scheme of data processing is one wherein the data is processed at the individual nodes themselves and not at a centralized base station. The distributed approach used in this work aims to minimize both the communication overhead in terms of number of messages being transmitted and the time required for data processing. This is achieved by communicating only the summary information of the data and not the entire raw data across the network nodes.

The concept of superpeers was first introduced in [10] by Montresor. The approach presented in this work assumes a pre-established network topology based on superpeers. Initially, data at each individual node is partitioned into separate clusters. Each node is then required to transmit only its cluster summary (and not the raw data) to its respective superpeer. It is at these superpeer nodes that the outlier detection algorithm is finally run. The algorithm accurately detects and reports the outlier values in the sensor network for the end user to perform any further analysis. This is done along with a reduction in the communication overhead and in the time required for data processing in the network as compared with a centralized scheme.

1.8 Outline of this Thesis

This thesis is organized into five chapters, including the present chapter. Chapter II gives an elaborate description of related work based on the extensive literature survey done during the course of this thesis. It discusses several successful methods previously implemented by researchers for identifying outliers in sensor network environments and the like.

Chapter III discusses the proposed methodology of distributed cluster-based outlier detection for sensor networks. A stepwise description followed by a flowchart for the proposed approach is provided.

Chapter IV provides a detailed description of the simulation environment, experiments conducted, and data analysis performed. The proposed approach was implemented using a JAVA simulation based on a combination of real and simulated data obtained from the website at http://www.weather.com/ [11]. Performance analysis of the distributed approach is done and several critical observations made.

Chapter V provides a conclusion to this thesis by summarizing the techniques used in the development of the methodology proposed, some critical results obtained, and suggestions for future research in this area.

CHAPTER II

RELATED WORK

Outlier detection in wireless sensor networks has gained tremendous attention in the past few years, due to the variety of applications they are used in and due to the accuracy of data these applications demand. This chapter discusses some related work done in the field of outlier detection in wireless sensor networks. In addition, the issues that were not addressed by previous researchers in this area are discussed.

2.1 Outlier Detection in Sensor Networks

While a fair amount of research has already been done in the area of outlier detection, there is still a lot of scope for further research with respect to sensor networks. This fact has been a constant motivating factor for several researchers to further delve into the possibilities of providing more reliable and error free systems.

The choice of locating the position in the network for processing the data is of critical importance. There are two possibilities of choosing the location in the sensor network to process the data. One is to perform the data processing at a centralized location whereas the other one, which is a more cost effective method, is to perform this operation at the nodes itself in a distributed manner. A lot of research is also being carried out that involves comparing these two approaches. The first approach described above may simply be termed as a centralized approach whereas the second one may be termed as an in-network or a distributed approach.

The figures below give a generalized view of both the approaches mentioned above. In Figure 1 below, the small black circles represent individual sensor nodes, and the large white circle represents the central base station. In Figure 2 below, the small black circles represent low capacity nodes that communicate directly only with the leader nodes (represented by large white circles) for their respective group. A variety of system topologies for the distributed case can be defined for a variety of applications.

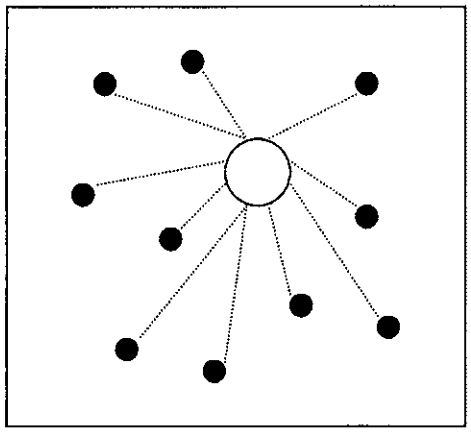

Figure 1. Centralized Approach

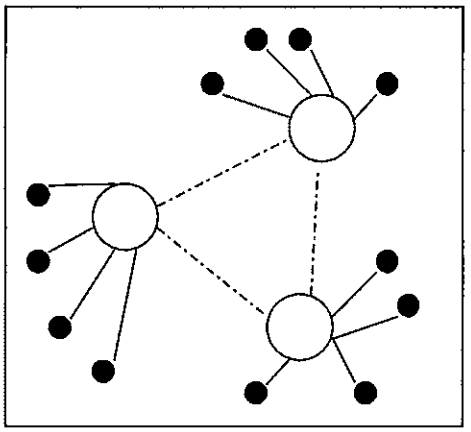

Figure 2. Distributed Approach

2.2 In-Network Cluster-Based Anomaly Detection

In [12], Eskin et. al. address the issue of anomaly detection in the context of intrusion detection based on unlabeled data. This paper does not explicitly discuss the issue of outlier detection in the context of sensor networks as such, but it provides a foundation on which the problem of anomaly detection in a sensor network scenario can be conveniently addressed. The authors base their proposal on the concept of unsupervised anomaly detection and processing of unlabeled data for identifying anomalies. Data elements are mapped to a feature space (typically a vector space) and

anomalies are detected by determining which points lie in the less dense regions of the feature space. The paper provides two types of feature maps: the first one being a data dependent normalization feature map (applied to network connections) and the second one being a spectrum kernel (applied to system call traces). The authors also describe three algorithms for detecting the points that lie in the less dense regions as anomalies. The authors refer to two types of anomaly detection instances. One is a supervised method and the other is an unsupervised method. They further argue that supervised anomaly detection algorithms require a set of purely normal data from which they train their model. Thus, these algorithms may not be able to detect future instances of the attacks if the data contains some intrusions buried within the training data. This setback in the supervised algorithms calls for more robust methods of anomaly detection. For this, an unsupervised anomaly detection algorithm is proposed in this paper. In this method, a set of data is given in which no distinction is made between normal and abnormal data in the training phase. But the goal eventually is to be able to identify the anomalous data. Thus, by processing an unlabeled data set, the paper aims to detect the abnormal values.

The paper proposes a geometric framework for unsupervised anomaly detection by mapping the records from the audit stream to a feature space. In this context, the feature space is nothing but a high-dimensional vector space. It is assumed that some probability distribution generates the input data and it is required to label the data that is in the low density regions of the probability distribution as anomalies.

As mentioned earlier, the authors collected their data from an audit stream of a system and split it into a number of data elements. The input (instance) space is defined

as the space of all possible elements. The input space, however, is solely dependent on the type of data that is being analyzed. This input space is then mapped to points in a feature space and, based on the framework described in the paper, the feature space is used to define relations between elements of the input space. The mapping of an input space to a feature space can be achieved by using the dot products of points in the feature space. The authors use kernel function to compute these dot products. They define a kernel function over a pair of elements in the feature space such that it returns the dot product between the images of those elements in the feature space. The authors also use convolution kernels that are kernels defined over arbitrary input spaces. Convolution kernels may be used to handle different kinds of data in a consistent framework using different kernels but using the same algorithms defined in terms of kernels.

For the detection of outliers in feature spaces, the authors present three algorithms. These three algorithms are implemented in terms of dot products of the input elements that allow the use of kernel functions to perform implicit mappings to the feature space. The first algorithm is a cluster-based one in which, for each point, it approximates the density of points near the given point. This is obtained by counting the number of points within a spherical cluster of fixed width w. Fixed-width clustering is initially performed and later the clusters are sorted based on their size. The second algorithm that is described in the paper detects anomalies based on computing the K nearest neighbors of each point. Here, if the sum of the distances to the K nearest neighbors is greater than a threshold, the point is identified as an anomaly. The third algorithm uses a support vector machine that identifies low support regions of a probability distribution by solving a convex optimization problem.

The experiments in this paper were performed over two different types of data, the first one being network connection records (KDD Cup 1999 Data) and the second one being system call traces. The two parameters used for performance analysis were detection rate and false positive rate. The performance of the algorithms was observed to be good for both the data sets.

The paper provides extensive details about the use of fixed-width clustering and *K*-nearest neighbor algorithm for anomaly detection in a stream of data from a single source. It, however, does not explicitly consider an environment like the sensor networks which consists of data streams from multiple nodes.

2.3 Distributed Clustering in Sensor Networks

A majority of data mining algorithms available in the literature consider data being transmitted to a central location before any further analysis can be performed. Bandyopadhyay et. al. aim to present a technique in [4] for clustering homogenously distributed data in a peer-to-peer environment such as a sensor network. The proposed technique is based on the principles of the traditional *k*-means clustering algorithm applied to a data stream environment. The main concentration of this paper is on separating the data obtained from a peer-to-peer network into a set of similar clusters. The authors base their proposal on the assumption that each node consists of only a subset of the overall data to be clustered and all nodes observe the same set of attributes. Also, every node is assumed to communicate with the neighboring nodes asynchronously about its current status and the data that it holds.

According to the authors, a major motivation for their work has been from the scene segmentation and monitoring applications of sensor networks. Segmentation, according to the authors, is a type of clustering. The main contribution of this paper is to provide a P2P clustering algorithm for a sensor network scenario. The authors use the *k*-means clustering technique for generating partitions in the data. The authors claim to address the issues of dynamism and communication limitations of the nodes in a sensor network scenario, while performing any clustering operations.

The proposed P2P *k*-means clustering technique considers communication between a node and a set of its neighbors. Every node is assumed to have the same random number generator so that the same set of *k* initial centroids is generated at each one. Nodes exchange information consisting of a list of nodes stored in $Comb^{(i)}$ (k) that respond to a particular change in the centroid values (i.e. $Centroids\_Changed^{(i)}$ flag is set) and the number of clusters. Each node will then calculate the weighted mean that includes data from its neighbors and later produces a set of final centroid values at the end of every iteration.

The authors also provide an elaborate analysis of the proposed algorithm. The analysis, according to the authors, allows a node, at a given iteration, to compute an upper bound on the centroid error based on current run-time information. This, they assume, will give a measure of the degree to which accuracy has been sacrificed at the expense of lowered communication costs.

Although the paper presents a novel proposal to cluster data in a sensor network, it does not address the issue of outlier detection for identifying anomalous values in the network.

2.4 Distributed Anomaly Detection in Wireless Sensor Networks

Continuous research is being carried out in the area of distributed or in-network outlier detection in wireless sensor networks. Rajasegarar et. al. have proposed in [3] a distributed anomaly detection algorithm in a wireless sensor network scenario. Their algorithm is based on fixed-width clustering and later using the *K*-nearest neighbor algorithm to finally detect the outliers in the sensor network data. The algorithm proposed in this paper aims to minimize the communication and energy costs by performing anomaly detection in the network itself.

The authors propose a hierarchical sensor network topology for their study. This consists of one gateway node, a set of intermediate parent nodes, and a set of children nodes. This type of a network topology is especially useful in cases where the transmission of data from all the nodes to a central node leads to increased communication and transmission costs. In a hierarchical topology, the children nodes communicate directly only with their immediate parent nodes. These parent nodes in turn communicate with their immediate parents. This process repeats until the entire network is covered and the gateway node is communicated.

The central idea of this paper lies in the fact that an attempt has been made to perform distributed anomaly detection in a sensor network scenario. The paper aims to minimize the communication and transmission costs by providing a distributed scheme of detecting outliers in the data. At the end of a predefined time interval, each node calculates a set of data vectors. Each data vector consists of a fixed number of attributes. Each node then performs a clustering operation on its own data and sends a summary of the clusters to its immediate parent. This summary consists of the number of data vectors

in each cluster and a set of cluster centroids generated by the individual node. The intermediate parent reads the summary information from its immediate children nodes, combines the clusters, and later merges them, thus, producing a new set of clusters. The cluster summary information from the intermediate parent node is again sent to its immediate parent. This process continues recursively up to a gateway node on which the anomaly detection algorithm is finally run. The duty of identifying the anomalies in the sensor network data is shared among all the nodes by using the clustering technique.

The clustering algorithm used in this paper is based on fixed-width clustering where the width of all the clusters is prefixed. In the first stage, an initial centroid value is selected and distance from every other point to this point is calculated. If the calculated distance is less than the fixed width w, the point is added to the present cluster. Otherwise, a new cluster with this data point as the cluster center is formed. Several iterations are performed until a stable point is achieved. The next stage consists of transmitting the cluster summaries to parent nodes up to the gateway node. The third stage involves detecting the anomaly clusters by using the K-nearest neighbor algorithm on the set of clusters obtained at the gateway node. In this way, a distributed approach has been used to identify anomalies in the sensor network data.

The simulation experiments presented in the paper uses the cluster width, w as the parameter for analyzing the false positive rate, FPR (occurs when a normal observation is identified as anomalous by the algorithm), and the false negative rate, FNR (occurs when an anomalous observation is identified as normal by the algorithm). In this case, the K value (of the K-nearest neighbor algorithm) is kept constant while varying the width, w of the clustering algorithm. The test data was obtained from the Great Duck Island project.

The evaluation performed in the paper showed that the proposed distributed approach achieves comparable performance with the centralized case.

Although the proposal made in this paper provides a foundation for distributed anomaly detection in the case of wireless sensor networks, the choice of the width, w can greatly affect the performance analysis. Also, the hierarchical topology used in this paper still requires the nodes to communicate with a single gateway node in a certain way. This carries with it reasonable communication costs. A more desirable approach would be to separate the sensor nodes into independent groups and provide an intermediate set of higher capacity nodes, which can act as leaders for each group of nodes. These nodes will then report the outlier values from a set of nodes in a small region around it such that the time required for processing is reduced, thereby reducing computation costs further.

2.5 Distributed Deviation Detection in Sensor Networks

Palpanas et. al. present yet another proposal for distributed deviation detection in a sensor network in [13]. The paper deals with the problem of distributed deviation detection in streaming data. The authors address this issue by using a technique based on kernel density estimators. The authors claim that their technique processes as much data as possible in a decentralized fashion so as to avoid any unnecessary communication and computational effort.

The authors are interested in values that significantly deviate from the norm and term such values as 'deviations.' This definition of deviation in a data set is of special importance in a sensor network setting because it can be used to detect the faulty sensors and to filter spurious reports from different sensors. The major motivation behind this

work is the fact that sensors have limited resource capabilities and that multiple data streams from various sensors have to be dynamically processed. In such cases, it becomes very important to minimize the processing and communication overhead of the sensors.

The architecture used by the authors consists of two different sets of nodes. The first category is a set of low capacity nodes deployed in large numbers. The second category is a set of high capacity nodes, which are more powerful and more sophisticated and are deployed in much lesser numbers as compared with the low capacity nodes. In this way, all the low capacity sensors can be divided into groups and each group can be assigned to one of the high capacity nodes. The paper aims to report outlying values coming from unknown data distributions. This means that data models are built as the data values come in and an approximation of the data distribution is made. In such a scenario, outliers are those values that significantly differ from the model.

The authors divide the task of finding deviations into two subtasks: the first one deals with modeling the data distribution, and the second one deals with managing and combining these models in the network of sensors. The authors also address the issue of timeliness guarantees on the delivery of streaming data.

The first stage deals with building the approximate data distribution model. The model used in the paper to estimate the distribution of the values generated by the sensors is based on kernel density estimators. The model is built based on a static relation T, which stores the data values, t. The requirement here is that these values must fall in the range of [0, 1]. A kernel function $k(x)$ is then defined according to a certain criterion. The underlying distribution $f(x)$ is approximated later according to which the values in T are generated using a certain function as described in the paper. A fact that is also considered

here is that it suffices to consider just the values in a sliding window of size N. Outliers are then detected based on the number of values that are in the neighborhood of a particular value. This neighborhood is estimated based on the distribution function $f(x)$.

The second stage described in the paper deals with integrating the deviation detection functionality in a distributed environment. At the lowest level, i.e. at the low capacity sensors, local outliers are identified. At the next level, i.e. at the high capacity sensors, the outliers reported are with respect to all the sensors in the particular region. Thus, a model combining technique is adopted in order to obtain a single model from two different local models. The combined model is later assigned to a high capacity node. Since kernel density estimators are used, the sample set, S and the bandwidth of the kernel function, B, can be combined simply by taking their unions. This type of combining the data models gives a coarse view of the network at the high capacity sensors level. The details of the data values are masked away and can be obtained by querying the individual sensors directly.

Although the paper aims at addressing the issue of distributed deviation detection in the sensor network environment, the modeling of data distribution requires the setting up of initial parameters. Also, kernel density estimator technique needs to be constantly adjusted with respect to sample and standard deviation of the data values. Along with this adjustment, a self-correcting mechanism, which is capable of fine tuning other parameters of the model, such as the sample size, needs to be provided. Also, a major concern would be quantifying the accuracy of the combined model. The accuracy of the combined model greatly depends on the frequency of updates of the parameters from the underlying models.

2.6 Other Distributed Anomaly Detection Systems

Branch et. al. present their work on in-network outlier detection in wireless sensor networks in [14]. The authors address the issue of unsupervised outlier detection by developing an algorithm that works in-network. The authors claim that the algorithm results in the communication load being proportional to the outcome and reveals its outcome to all the sensors. This algorithm introduces reasonable communication load and power consumption. In this paper, the authors define outliers as events with extremely small probability of occurrence. The distributed system architecture described in this paper consists of a system of peers each holding a set of data points. An outlier detection algorithm is run on each peer, and each peer has knowledge of an outlier ranking function. Based on this information, peers determine the outliers in their data. The authors perform analysis on a real data set and also on simulated values. According to them, the algorithm shows promising results when compared with a centralized approach.

Another attempt to detect outliers in wireless sensor networks is made by Janakiram et. al. in [1]. The authors base their method on building Bayesian Belief Networks. They assume that a sensor node observation can either be detected as an outlier or not based on its neighbors' readings as well as the readings of the node itself. Based on this assumption, and the fact that conditional dependencies exist among various attributes of the sensor readings, the authors use Bayesian Belief Networks (BBN) to detect outliers in a sensor stream data. If a particular observation falls outside the range of a class, it is termed as an outlier. The paper bases its outlier detection method on several criteria as is explicitly described in it. The proposed method not only handles outlier data but also estimates missing data based on the Bayesian network built. The authors describe

three phases in the detection of outliers: training phase, testing phase, and inference phase. They later describe their attempt to integrate their outlier detection algorithm as a component of a middleware called COMis (Component Oriented Middleware for Sensor Networks). The authors claim is to improve the accuracy in detecting outliers and missing values in the sensor data.

Yet another attempt is made to address the issue of online outlier detection in [15] by Subramaniam et. al. The authors propose a technique to provide online outlier detection in sensor data using non-parametric models. The paper proposes a framework that computes in a distributed fashion an approximation of multi-dimensional data distributions in order to accommodate complex applications in resource-constrained sensor networks. The main goal of this paper lies in its proposition to identify either distance- or density-based outliers in a single pass over the data and with limited memory requirements. The model described in the paper operates on approximations of the sensor data distributions. The authors base their technique on the idea of approximating the data distribution in order to achieve considerable energy savings in the network. The proposed model does not require apriori knowledge of the data distribution. The paper assumes the sensor network to be organized using overlapping virtual grids or cells. The grid network consists of an additional set of nodes called the leader or parent nodes, which are responsible for processing measurements of all the sensors in a particular cell. Moving up the tiered system, the leader node of a cell collects values from the leader nodes of all its sub-cells in the lower level. The authors present two different definitions for outliers on which further analysis of the data can be based. The first definition is of a distance-based outlier in which a point p in a data set T is a (D, r)-outlier if at most D of the points in T

lie within a distance r from p. The second definition is of local metrics-based outliers which detects outliers, based on the metric Multi Granularity Deviation Factor (MDEF). According to the paper, for any given value p, MDEF is a measure of how the neighborhood count of p compares with that of the values in its sampling neighborhood. A value is termed as an outlier if its MDEF is significantly different from that of the local averages [15]. In this paper, the problem of finding outliers is addressed by computing an accurate approximation of the data distribution. The authors claim that, using an approximation for the data distribution allows for combining data from multiple sensors efficiently, thereby minimizing the communication costs. Also, the approximation is determined in-network itself, which will further help in the computation of other queries in the network too. The probability density function is estimated using kernel estimators whose basic step is to produce a uniform random sample. The authors give a detailed description for the calculation of the data distribution in a sliding window, distributed computation of estimators, fault tolerance produced, and complexity analysis. A very elaborate description of distributed detection of distance-based outliers and outlier detection using multi-granular local metrics is made. Also, applications, such as online query processing, are considered and the use of the proposed algorithm is studied in greater detail. An extensive experimental evaluation is made with respect to the accuracy of data distribution estimation and accuracy of outlier detection mechanism. Although this paper provides a comprehensive description of an online outlier detection algorithm, it is based on building a data distribution model for the sensor values. This increases the complexity of the system. And so, it is critical to find a simple system architecture that

deals with simple data processing tasks in order to detect outliers in a less complex way along with preserving the accuracy of the detection process.

As seen from the above discussion, a fair amount of literature deals with using clustering techniques for the detection of outliers in sensor data. However, the traditional *k*-means clustering algorithm has not been used anywhere in the literature for addressing the problem of outlier detection in sensor networks.

CHAPTER III

SYSTEM DESIGN AND APPROACH

3.1 Overview

The approach proposed in this thesis is used to address the issue of cluster-based outlier detection in wireless sensor networks while reducing the communication overhead and processing time. The algorithm is based on the popular *k*-means clustering paradigm to separate a set of sensor data into groups of similar data points. 'Similar data points' in the above statement refers to data points that are relatively close to each other. The closeness between a given pair of data points is determined in terms of the Euclidean distance measure. The final anomaly detection is based on the traditional *K*-nearest neighbor algorithm, a popular data mining technique for the detection of distance-based outliers.

The choice of the system architecture or topology of the network made in this thesis is in order to provide a distributed environment for performing the processing of data. The sensor nodes themselves are clustered into separate groups each headed by a superpeer node. This is helpful in performing in-network computations on the data. The concept of superpeers presents a kind of heterogeneity in the network. It, however, allows P2P networks to perform more efficiently without compromising their decentralized nature.

The topology of the network consists of a large number of low-capacity sensor nodes termed as 'clients' and a relatively small number of high capacity nodes termed as 'superpeers.' A set of clients are conveniently assigned to a superpeer based on some

predefined criterion. This describes the overall system architecture. A detailed description of the sensor network topology is given in Section 3.3 below.

Every node in the network performs a *k*-means clustering operation on its own local data and produces a summary of the clusters generated. The summary information is then transmitted over the network to each node's respective superpeer. The superpeer combines its own clusters with the clusters from all its clients and merges them, thus, producing a final set of clusters. Summary information of the clusters obtained at the superpeers is maintained at the superpeer itself. A general description of the *k*-means clustering algorithm, and the summary information used by the nodes in the network is given in Section 3.4.

The superpeers finally run an outlier detection algorithm based on *K*-nearest neighbor technique (as explained in [3]) on its clusters and detect the outlier clusters. The outlier clusters from all superpeers are thus reported for the data analyst to use them for any further analysis. A general description of the *K*-nearest neighbor based outlier detection algorithm used in this thesis is given in Section 3.5 below.

A flowchart for the overall algorithm is provided in Section 3.6.

3.2 System Description

This section provides an overall description of the system and details of the notations used in this thesis. The sensor network considered in this thesis is assumed to be composed of a set of n nodes represented as follows,

$$N = \{n_i: i=1, 2\ldots n\}$$

The topology of the sensor network consists of a large number of client nodes and a relatively small number of superpeer nodes, the details of which are discussed in the following section.

After every time interval of say Δ_t, each sensor node n_i measures a set of features such as temperature, humidity, wind, etc., in the form of a feature vector say x^i_t . The feature vector can thus be represented as,

$$x^i_t = \{v^i_{tj} : j = 1, 2.....,d\}$$

In the above statement, d determines the dimension of the data set, i.e. the number of features measured by the sensor. Since only the current values of the data measured are sufficient to perform the operation of outlier detection, it is enough to consider a sliding window of m measurements. This means that at the end of every window of m measurements, each sensor node measures a set of features on which further data processing can be performed. Thus, at the end of each window, every sensor node would have collected a set of measurements given by,

$$X_i = \{x^i_t : t = 1, 2, ..., m\}$$

The objective of this thesis is to find the set of outliers in the sensor network data in such a way that the process takes place at the node level itself. The proposed method for detecting outliers in the network itself aims to minimize the communication overhead in terms of drastic reduction in the time required to complete the operation. The problem of detecting outliers in the data is to find a set, $O \subset X$ where,

$$X = \mathrm{U}_{i=1....n} X_i$$

A simple overview of the methodology used in this thesis is given below. It is assumed that each node in the network knows its respective superpeer at the start of the

algorithm itself. This implies that the system architecture is determined prior to the start of the algorithm. At the end of each window of m measurements, the following operations are performed in the network:

- Each node $n_i \in N$ partitions its local data X_i into a fixed number of k clusters and produces a set of clusters, $C_i = \{c_r^i : r = 1...k\}$. Here, the number of clusters k is a parameter whose value is determined initially and provided as input to the algorithm.
- Each client node then communicates with its respective superpeer and sends information only regarding its clusters C_i in the form of a simple cluster summary (and not the entire set of observations it measures). This is done because each cluster $c_r^i \in C_i$ can be sufficiently represented by its centroid value and the number of data vectors it contains. This concept has been proven in the literature [16]. Let a cluster c_r^i contain $N_r^i \leq m$ number of data vectors and a set of data vectors, $X_r^i = \{x_q^i : q = 1...N_r^i\}$. Then, the centroid values and the number of data vectors of each cluster in the set of clusters at node n_i will represent the summary information that the node transmits to its superpeer. Thus, the sufficient statistics that need to be determined by each node are given below,

$$Centroid(c_r^i) = \frac{\sum_{q=1}^{N_r^i} x_q^i}{N_r^i}$$

$$Num\_Data\_Vectors(c_r^i) = N_r^i$$

- Each superpeer node S_p combines and merges its own clusters C_p with the clusters it receives from all its clients $C = \bigcup_{i \in clients(s_p)} C_i$ and forms a combined set of clusters $C_c = C \cup C_p$. It then runs the clustering algorithm on this combined set of clusters again and produces a new set of k clusters and their summaries for the aggregated clusters.
- Each superpeer node then runs an outlier detection algorithm based on calculating the inter-cluster distances ICD_{c_i} for all the clusters it holds. It then chooses the shortest K inter-cluster distances for each cluster to find its average inter-cluster distance $AICD_{c_i}$.
- Finally, the outliers in the network are detected at the superpeer nodes by using a conditional criterion and then reported.

3.3 Network Topology

This section provides a detailed description of the network topology and the reasons for choosing it for this thesis.

Sensor networks are a typical example of peer-to-peer (P2P) type of networks. The topology of the network considered in this thesis consists of a large number of low capacity sensor nodes termed as 'clients' and a relatively small number of sensor nodes termed as 'superpeers'. This is an example of a distributed type of a network, wherein the nodes talk to their peers in a certain way. But the specific type of topology that is assumed for this work is in order to improve the system performance in comparison to a centralized case. The centralized scheme as mentioned earlier requires that all nodes

necessarily have to send the entire unprocessed data to a central node for any kind of processing. Thus, a distributed system aids in processing the data in the network itself, thereby, improving the system performance especially in terms of reduction in communication overhead.

The superpeer nodes serve as leaders for a group of low capacity client nodes. This grouping of low capacity nodes under each superpeer is decided based on a predefined selection criterion. Simply put, a superpeer operates as a server node for a group of client nodes. The system topology is assumed to be set up already so that it can be used directly by the outlier detection algorithm proposed in this thesis. It is to be noted that this kind of a topology was chosen in order to accommodate the large scalability of modern sensor network systems. The figure below shows an example of the superpeer topology used.

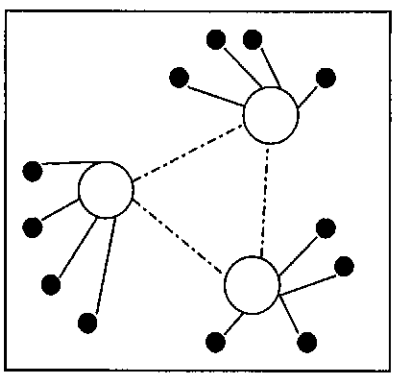

Figure 3. Superpeer Topology

In Figure 3 above, the small black circles represent the client nodes and the large white circles represent the superpeer nodes.

3.4 Clustering Algorithm

The technique used in this thesis to detect outliers in sensor network data is based on the principles of the popular *k*-means clustering paradigm. This clustering algorithm is initially run on each of the nodes in the system including the superpeer nodes. It is later run again on the superpeer nodes alone. The clustering algorithm divides the given data set into *k* number of partitions of similar data. Here, similarity is measured in terms of Euclidean distance between two data points in the data set. As mentioned earlier, the sensor network data points considered in this work are typically data vectors of a certain dimension that represent the number of features or attributes depending on the application requirements.

k initial cluster centers (centroids) are determined from the set of input data points based on some heuristic. Each of the remaining points is assigned to its closest cluster based on the distance between the point and the initial cluster centroids. New values for cluster centroids are then calculated. This process is repeated until the criterion function converges. The criterion function that is used generally is the squared-error criterion. The squared error criterion [5] is defined below,

$$SSE = \sum_{i=1}^{k} \sum_{p \in c_r^i} \left| p - Centroid(c_r^i) \right|^2$$

Here, *SSE* = sum of square-error for all points in the data set

p = any data point

The *SSE* criterion makes the resulting *k* clusters as compact and separate as possible. This helps in determining the outlier values in a data set.

3.5 Outlier Detection Algorithm

The outlier detection algorithm used in this thesis is based on the popular *K*-nearest neighbor algorithm to identify the *K* nearest neighbor clusters and use their average inter-cluster distances to detect the outlier clusters. This outlier detection algorithm is run on the already obtained final set of clusters at the superpeer nodes. The metric used in this method for identifying outlier clusters is the average inter-cluster distance between a cluster and its *K* nearest neighbor clusters [6]. Thus, the anomalous clusters are accurately detected and reported.

The outlier detection algorithm used in this work is described below:

At each superpeer node:

- For each cluster in the cluster set, the inter-cluster distances to every other cluster is determined. The set of inter-cluster distances is represented as,

$$ICD_{c_i} = \{d(c_i, c_j) : j = 1,...(|N_c| - 1), j \neq i\}$$

  Here, $d(c_i, c_j)$ - Euclidean distance between centroids of clusters c_i and c_j, and

  $|N_c|$ - Number of clusters in the final set of clusters at the superpeer

- For each cluster c_i, the closest *K* distances are selected from its ICD_{c_i} cluster set and their average is determined as follows,

$$AICD_{c_i} = \frac{1}{K} \sum_{j=1, \neq i}^{K} d(c_i, c_j)$$

  The assumption made here is that $K \leq |N_c| - 1$, i.e. at the most, *K* shortest distances can be selected for each cluster set at each superpeer node.

- The set of average inter-cluster distances for all the clusters is determined,

$$ICD = \{AICD_{c_i} : i = 1, 2, \ldots \mid N_c \mid\}$$

And finally, an outlier cluster is identified as that cluster, which has an average inter-cluster distance $AICD_{c_i}$ more than a standard deviation of the inter-cluster distance *std_deviation*(*ICD*) from the mean of the inter-cluster distance *avg*(*ICD*). This implies that a cluster is declared as an outlier cluster if it satisfies the below condition,

$$AICD_{c_i} > avg\ (ICD) + std\_deviation\ (ICD)$$

3.6 Distributed Cluster-Based Outlier Detection

This thesis proposes to distribute the function of outlier detection to all nodes in the network. The flowchart for the overall methodology used in this thesis is described in Figures 4 to 9 below.

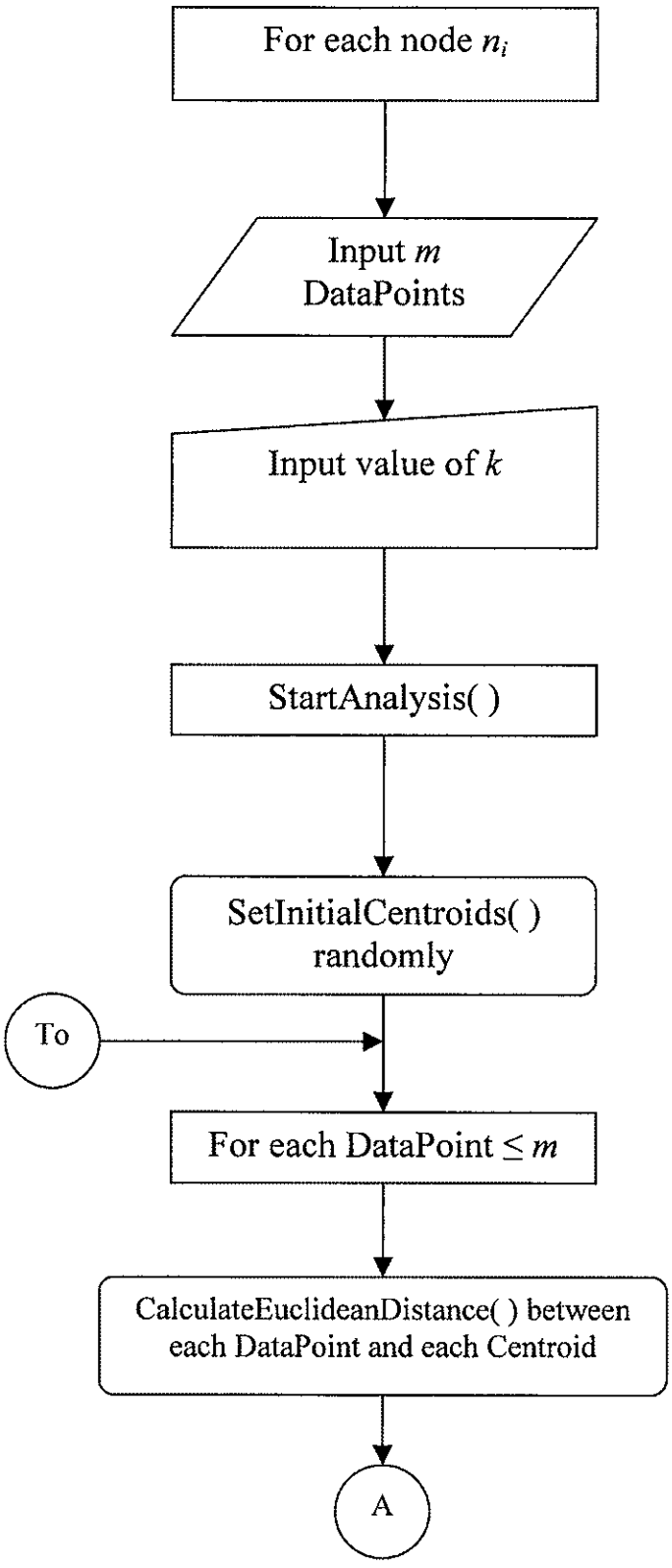

Figure 4. Flowchart for clustering algorithm run on all nodes in the network

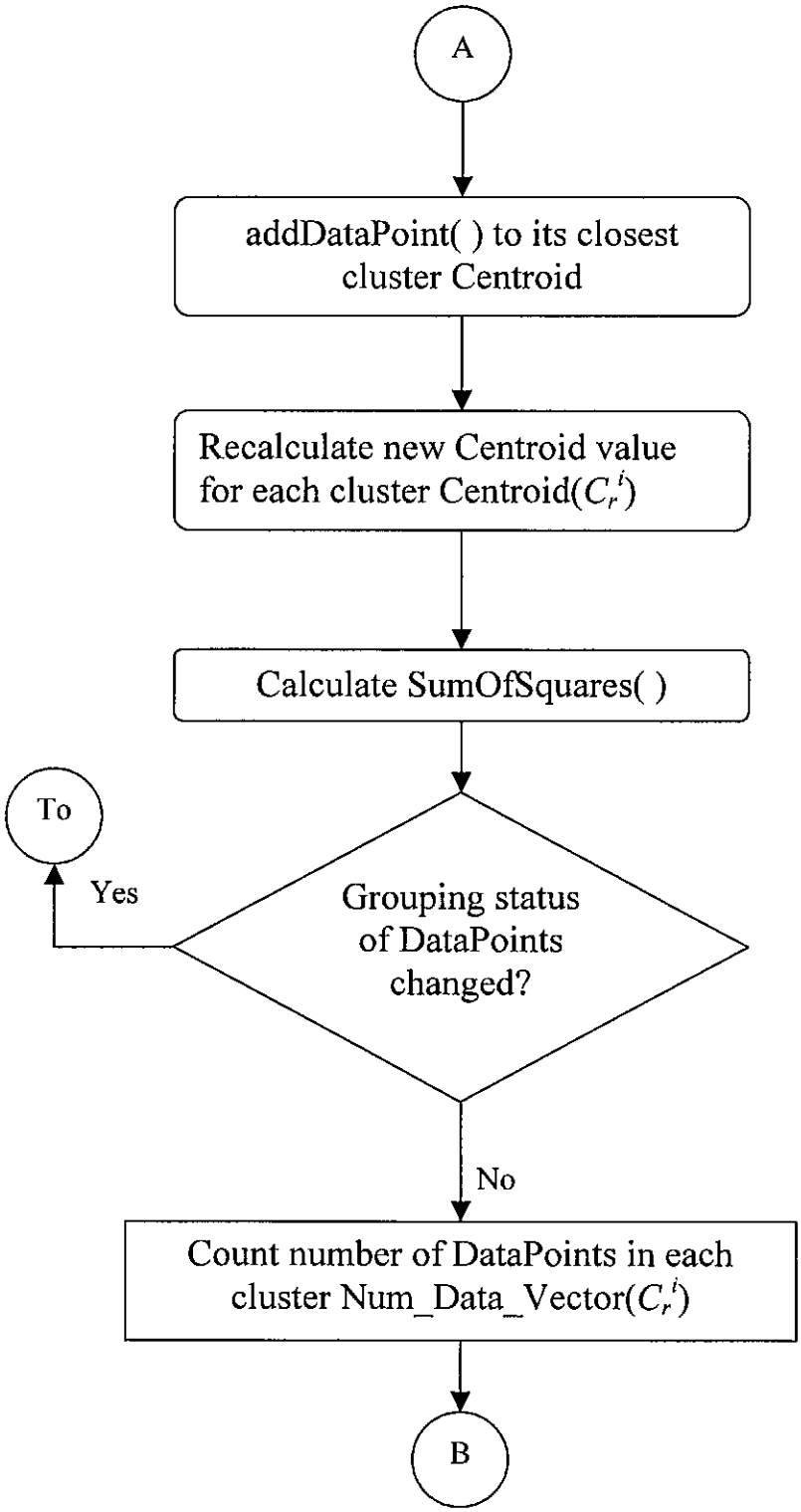

Figure 5. Flowchart for clustering algorithm run on all nodes in the network (contd.)

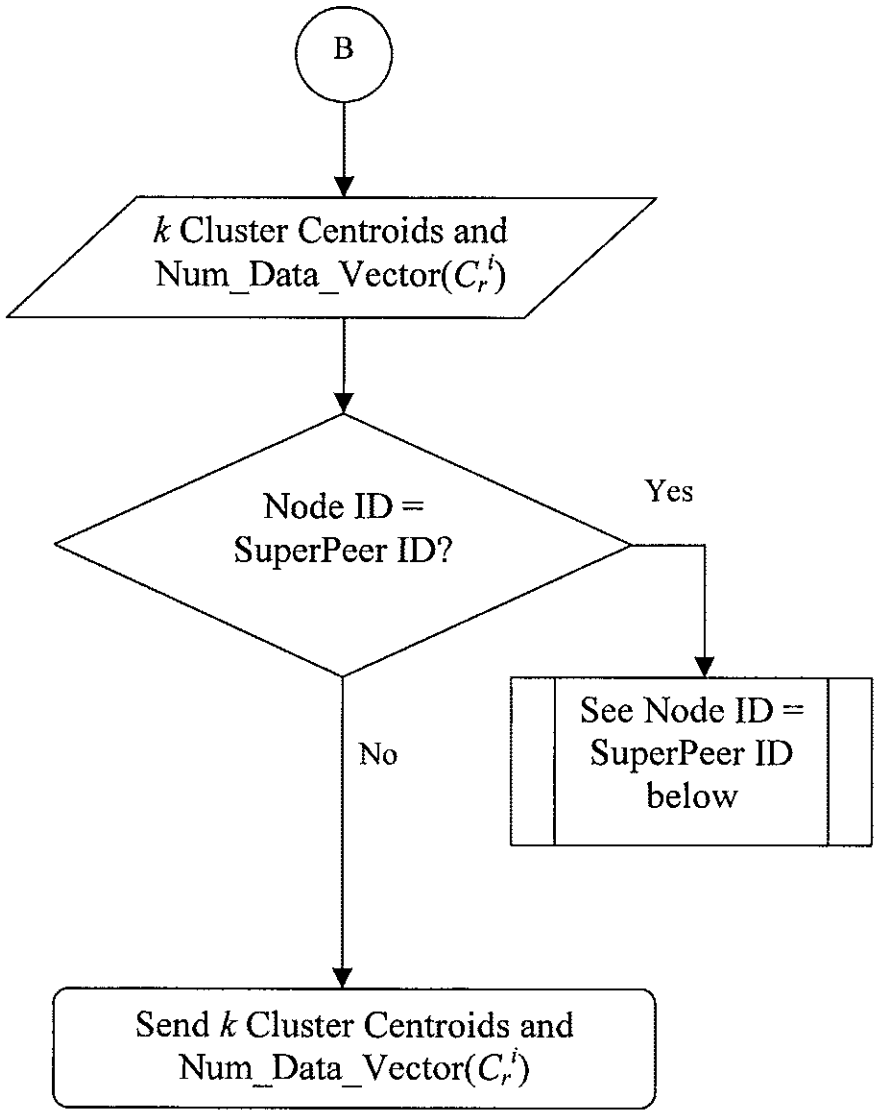

Figure 6. Flowchart for clustering algorithm run on all nodes in the network (contd.)

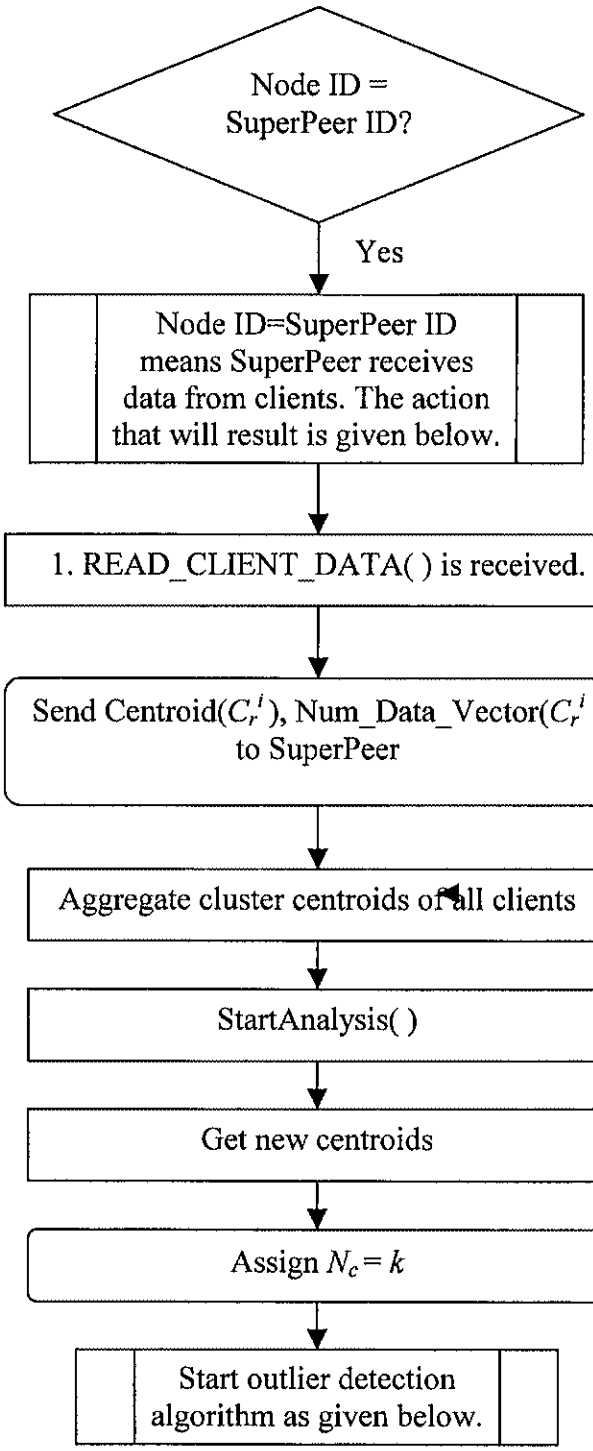

Figure 7. Flowchart for data transmission and clustering operation at superpeers

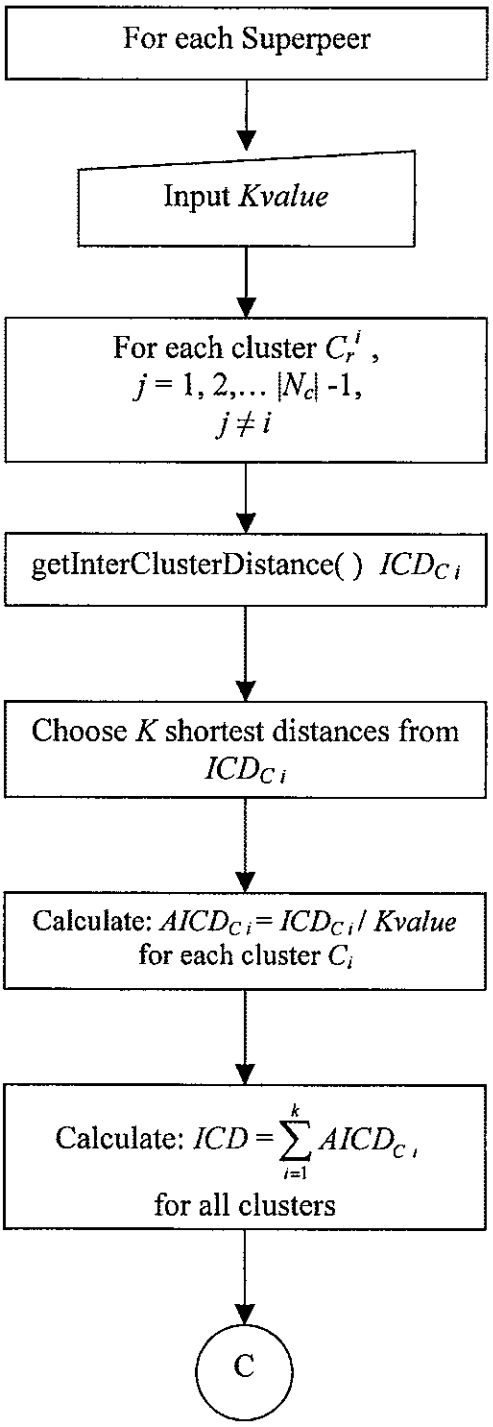

Figure 8. Flowchart for outlier detection operation on superpeers

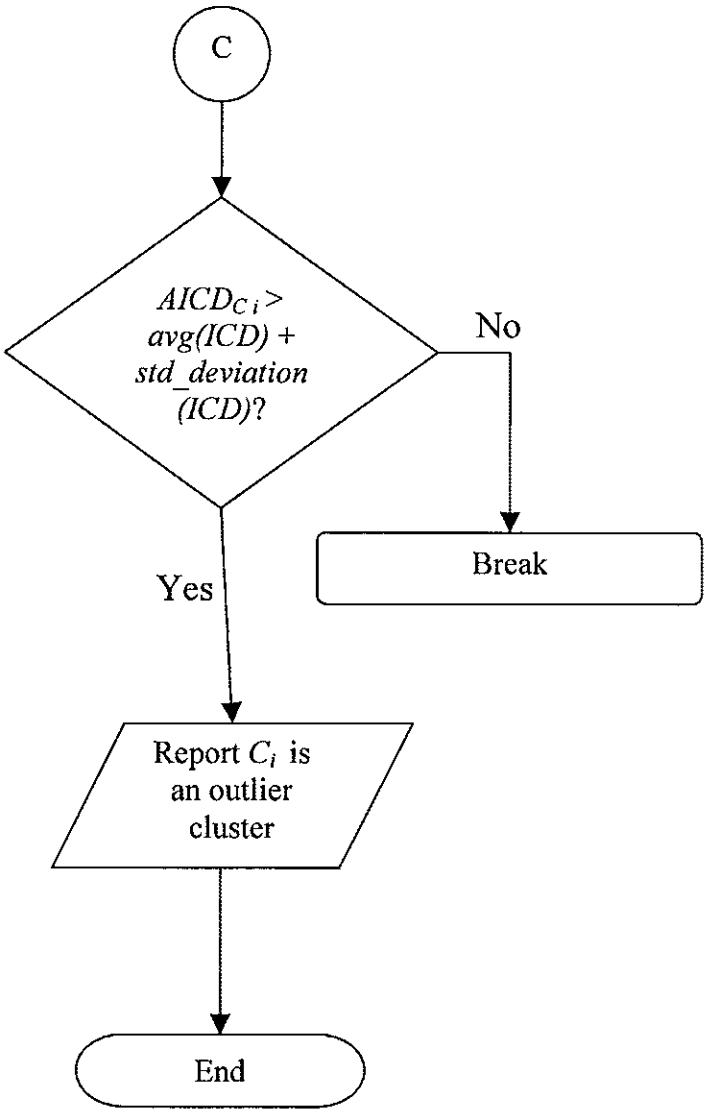

Figure 9. Flowchart for outlier detection operation on superpeers (contd.)

CHAPTER IV

EXPERIMENTS, RESULTS AND ANALYSIS

This chapter deals with all the experiments conducted for this thesis. The distributed approach of outlier detection proposed in this thesis is implemented in a JAVA simulation based on a combination of both real data and simulated data. This implies that a sample of real data was taken and based on this data, a more suitable data set was obtained using a random number generator. Details of all the experiments, as well as their corresponding results, are presented in this chapter. Comparisons between the distributed approach used in this thesis (requires every node to participate in the outlier detection) and a centralized approach (requires the entire raw data to be transmitted to a central base station, thereby performs the outlier detection) are made wherever possible. Detailed explanation of the experiments based on the analysis of the data is given.

4.1 Data Set and Network Size

The experiments in this thesis were implemented in a JAVA simulation based on real data obtained from the website at http://www.weather.com/. A sample data set was collected for two days from 2nd October to 4th October 2007 for every one hour interval for a period of 48 hours. The data set was a three-dimensional one with temperature (in degrees Fahrenheit), humidity (as a percentage), and wind speed (in miles per hour, mph) as the three features of each data vector. Thus, all experiments were conducted on a three-dimensional data set and corresponding results obtained. Based on this real data set, two sets of outlier data sets with relatively high values and relatively low values were

generated using a random number generator in JAVA. These three data sets were each stored in text files called NorfolkWeatherData.txt, NorfolkWeatherDataBogusHigh.txt and NorfolkWeatherDataBogusLow.txt. The three data sets were used for conducting various experiments for this thesis and producing results for further analysis. A default number of 24 data vectors per each node in the network in each sliding window (i.e. a window with m=24) is assumed for most of the experiments. This value is, however, varied for few experiments in order to analyze the behavior of certain other parameters in the network.

For the experiments, the network topology is assumed to consist of a default number of 100 nodes with five superpeers included for the distributed case and a set of 100 nodes with only one base station included for the centralized case. For most of the experiments, the number of nodes in the network is maintained constant.

4.2 Simulation Setup

Simulations were run for experiments with different scenarios, and results obtained were based on an initial setup designed for the overall system. This section provides details of some initial settings made during the development of the methodology and during the simulation runs. It is to be noted that this thesis deals with the data in the sensor network, which is part of the application layer. And hence, most of the topology and routing issues are based on some standard assumptions. It is also important to note that several experiments that were conducted required some default values for various parameters as well as some additional set up, which is clearly stated under each experiment scenario in the later sections.

4.2.1. Network Setup

The topology of the network is preset based on the ideology of superpeers. Also, a static topology is considered wherein no addition or removal of nodes occurs. This implies that, once the topology is set up, it remains the same throughout the experiment. Although not explicitly discussed in this thesis, the sensor nodes are themselves clustered into groups of client nodes with each group headed by a high capacity superpeer node as its leader. Thus, each group of client nodes communicates directly only with its respective superpeer. This type of a hierarchical system was used in order to achieve considerable decrease in the processing time. The proposed approach requires that the topology selected is a hierarchical one. Also, to make the system imitate a real-time scenario as closely as possible, it is assumed that each sensor node has its own sleep and wake cycles. This implies that not all sensors measure the values at the same time. The superpeers are then required to wait until each of their clients transmits its data.

4.2.2. Other Settings

For the outlier detection algorithm based on finding the K nearest neighbor clusters, the K value was fixed based on some sample runs performed prior to the analysis. A trial and error method was used and sample runs performed for determining an optimal value for K, and this value was assumed to hold good for all the experiments that were conducted. For experiments involving the centralized case, the topology was assumed to be preset also. A static topology is assumed in which each node knows the central base station. The default network was assumed to consist of 100 nodes, which a single base station can conveniently handle. This implies that a node directly

communicates with the base station in a single hop and that it is not required to know its peers in order to communicate with the base station. Dynamism in the network is not considered in this thesis because outlier value detection and mobility are seen as two completely separate areas of research. Since the main emphasis is on the data processing issues in a sensor network environment, most of the networking and topology issues are assumed apriori.

4.3 Simulation Environment

The simulation environment designed for this thesis aims to provide options to work with different parameters related to the methodology presented. It thereby provides an interface to be able to view the outlier clusters that have been detected in the sensor network. The overall simulation environment was designed in such a way as to display the values of initial set of cluster centroids that each node transmits to its respective superpeer node and the number of data vectors each of these clusters contains. Also, the clusters formed at the superpeer nodes and their respective centroid values are displayed. The code developed was often modified to test the system performance for varied set of parameters.

4.4 Experiments and Data Analysis

The first set of experiments that were conducted consisted of varying the number of nodes in the network keeping the percentage of the superpeer nodes and the percentage of the nodes carrying outlier values constant. A set of fifteen simulations were run for each network size. The number of nodes in the network was varied from 80 to 160, and

the average time required for detecting the outliers by all the superpeer nodes was measured. Figure 10 below displays a graph of the average time required by the network to detect all the outlier clusters in the network while varying the total number of nodes in the network. The average time calculated is the sum of the time taken by each of the nodes to produce their local clusters, time taken by the client nodes to send the cluster information to their respective superpeers, the time taken by the superpeer to perform another clustering operation on the combined cluster set, and finally the outlier detection algorithm. A 95% confidence interval for each case of sample mean was calculated and the difference was observed for each network size. The largest interval of 7.528 +/- 0.0358 seconds was observed for the 160 node network size, as shown in the figure below.

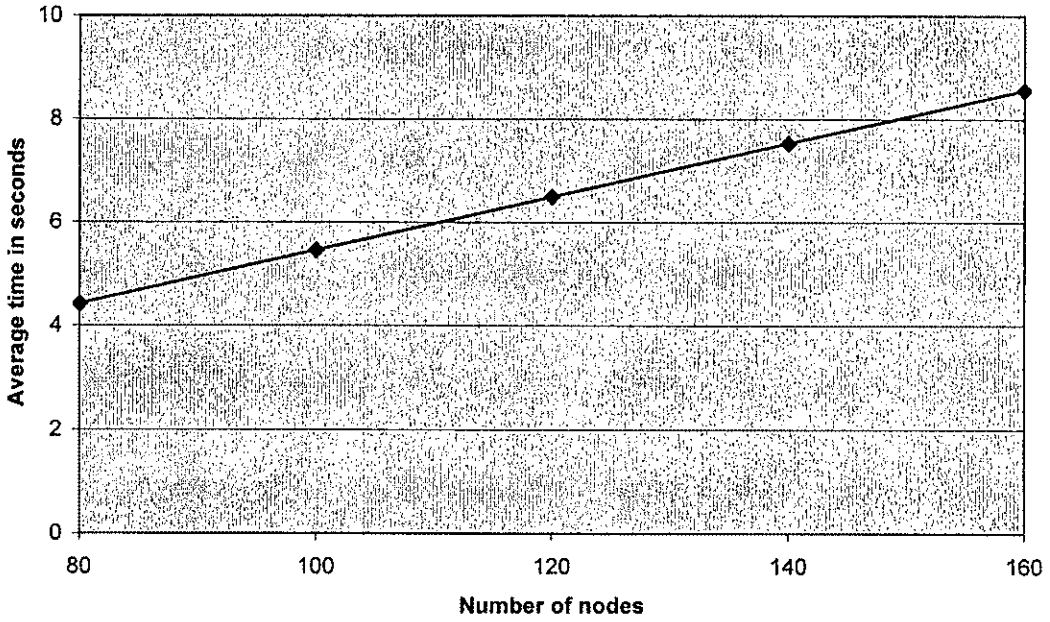

Figure 10. Average time vs Number of nodes for the proposed distributed case

It is observed from the figure above that, as the nodes are increased in the network, the time taken by the network to detect the outliers is also increasing. A point to

be noted here is that the average time scales almost linearly with the network size. This clearly indicates the performance of the algorithm with respect to network size. It can thus be concluded that the proposed approach varies linearly with time.

Another observation that can be made from the graph above is that each superpeer node actually waits for all its client nodes to transmit their respective cluster summaries before it can start further processing of data. This is attributed to the fact that all the nodes in the network send their cluster summaries only at the end of a sliding window of m (here, m=24) measurements, and all the superpeers will start data processing only after receiving data from their respective clients. Also, it is to be noted that each node has its own sleep and wake cycles. This means that, although the nodes transmit a set of 24 measurements at a time, they do not all necessarily transmit data at the same instant. This requires the superpeer nodes to wait for the entire process of data transmission to complete for them to start any advanced operations.

Figure 11 below shows a comparison between the distributed and centralized case by changing the network size and observing the average time taken for detecting outliers in the network. As is seen from the figure, the time increase with the increase in the network size in the distributed case is less as compared with the centralized case. This is obvious because of an increase in the number of data vectors along with an increase in the number of nodes in the network. This increase in the number of data vectors causes a corresponding increase in the average time required to transmit all these data vectors to the base station and also in the time that the base station waits before performing further processing of the data. This process results in an overall increase in the average time required to detect the outliers for the centralized case.

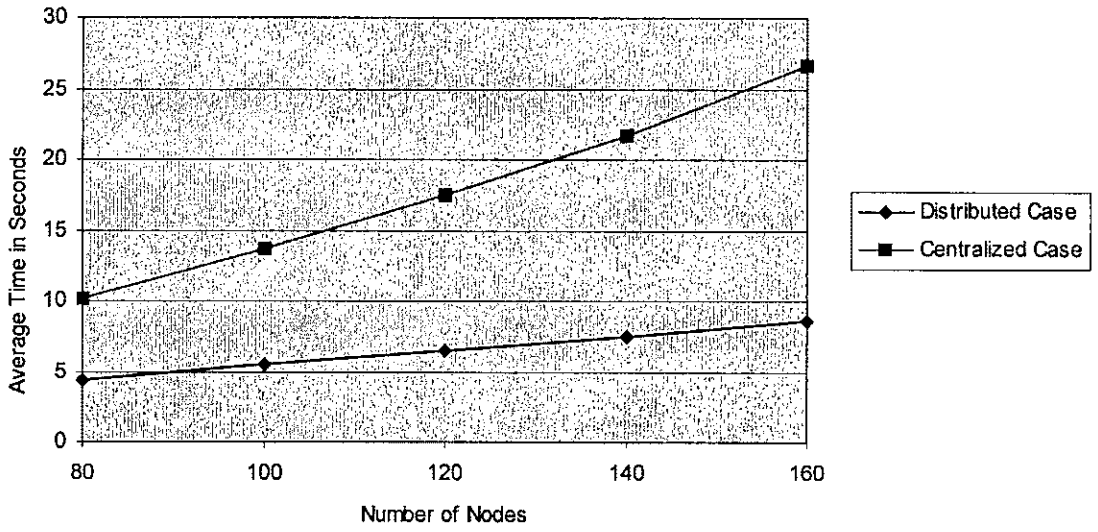

Figure 11. Average time vs Number of nodes for both distributed and centralized cases

A 95% confidence interval was calculated for each case for the average time required for processing the data and the largest interval of 26.626 +/- 0.489 seconds was observed in the centralized case for a 160 node network and of 7.528 +/- 0.0358 seconds was observed in the distributed case for a 140 node network.

The next set of simulation runs were performed for determining the number of messages transmitted per unit time across the network. The number of messages transmitted per second from all client nodes to their respective superpeer nodes was calculated by varying the number of data vectors in the network. The number of data vectors was increased by increasing the number of measurements transmitted by each sensor node in each window. One of the several advantages of using a cluster-based distributed approach for outlier detection in a sensor network is clearly demonstrated by this set of experiments. The number of messages transmitted across the network remains constant in the case of the distributed approach. This is because the data sent by each

client node to its respective superpeer consists of a constant number of messages. This is attributed to the fixed number of clusters considered for the analysis of the algorithm proposed in this thesis. The simulations were run by considering the transmission of a single floating point number as one message. Thus, the number of messages transmitted across the network can be determined by using the following equation:

$$N_{Total\ Messages} = (Number\ of\ SP) * (Number\ of\ Clients\ per\ SP)*(2 + k\,(d+1))$$

Here, *SP* - Superpeers

k - Number of Clusters

d - Dimensionality of a data vector

Two messages in the above equation are accounted for the superpeer ID and the client node ID in the message packet sent by the client nodes to their respective superpeers. The default values of the above mentioned parameters used for the purpose of this set of experiments are given in Table 1 below:

Table 1. Table showing default values

Parameter	Value
Number of SP	5
Number of clients per SP	19
Number of clusters, k	10
Number of features, d	3

The number of messages transmitted by all the client nodes across the network to their respective superpeers in the distributed approach proposed in this thesis can be

calculated using the equation above and substituting the default values of the parameters as follows:

Total number of messages transmitted, ($N_{Total\ Messages}$) = 5 * 19 * (1 + (10*3) + 10 + 1)
= 95 * 42
= 3990

The next step is to measure the mean time ($T_{MessageTransmission}$) taken for data transmission from the clients to their respective superpeers for each of the data set size. The dataset is increased from an initial value of 24 data points to a final value of 120 data points in uniform intervals of 24. A 95% confidence interval is calculated for each case of sample mean. It was observed that the average time taken for any data set size remains almost constant. The message transmission rate was then calculated by using the equation below:

Message Transmission Rate = (*Total Number of Messages Transmitted*) / (*Mean Time Taken for Message Transmission*)
= $N_{Total\ Messages}$ / $T_{MessageTransmission}$

Table 2 below shows the values of number of data points per nodes in the first column and their corresponding message transmission rates in the second column. It can be inferred from the table below that the time taken remains almost constant for any number of data vectors in the network as long as all the other parameters, such as the number of clusters k and the network size, are kept constant. This is due to the fixed number of clusters generated.

Table 2. Values of number of data points per node and message rate

Number of data points per node	Message rate (messages per sec)
24	312
48	307
72	301
96	299
120	303

The average message transmission rates were also calculated for the centralized case where each node transmits all its data vectors to a central base station. It was observed that the number of messages transmitted across the network from the individual nodes to a central base station increased linearly with the increase in the number of data points. It was also observed that the distributed approach of outlier detection takes far less time as compared to the centralized approach and also remains constant with a constant value of *k*. Figure 13 below displays the results of the message transmission rates for both the distributed and centralized cases by varying the size of the data set.

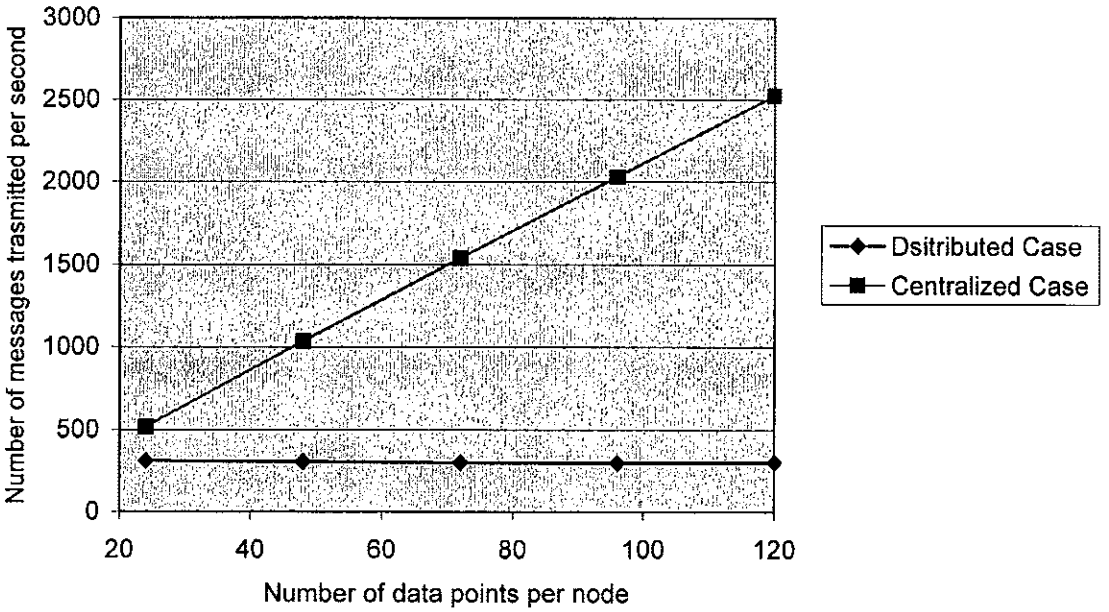

Figure 13. Message transmission rate vs Number of data points per node for distributed and centralized cases

Figure 14 below displays a graph obtained by calculating the percentage reduction in the communication overhead obtained in the distributed approach of outlier detection proposed in this thesis as compared with that of the centralized case by varying the number of clusters k into which the dataset is partitioned. Reduction in communication overhead is determined using the following equation:

Reduction in communication overhead = (*Number of data vectors transmitted in the centralized case – Number of clusters transmitted in the distributed case*) / (*Number of data vectors transmitted in the centralized case*)

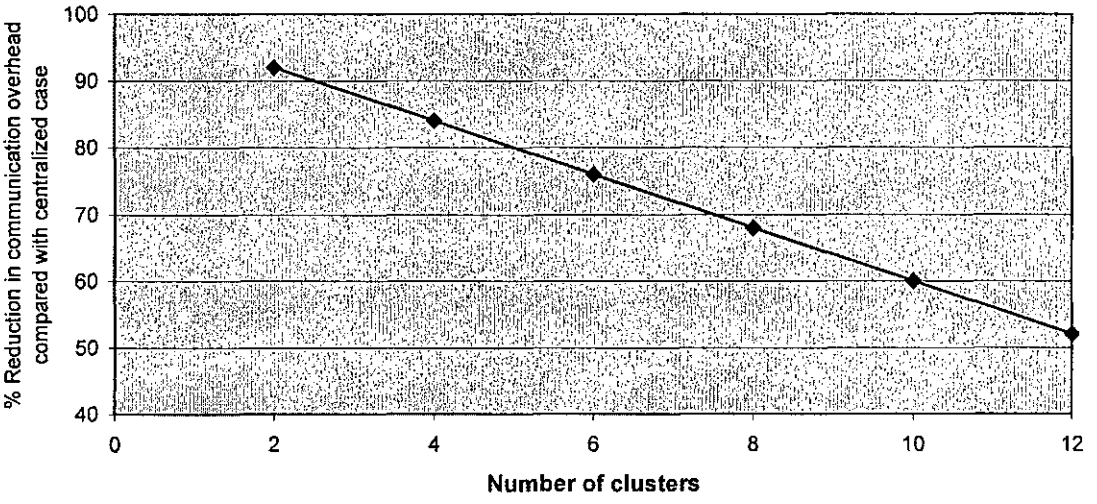

Figure 14. %Reduction in the communication overhead in the network vs Number of clusters

Figure 14 above illustrates the fact that, as the number of clusters into which data is partitioned increases, the number of centroid values that each client node transmits to its respective superpeer increases. Thus, there is a linear increase in the number of data values transmitted to superpeers from clients. This increase in the number of data values being transmitted leads to a linear decrease in the percentage reduction of communication overhead because the number of data vectors transmitted to the central base station in the centralized case remains constant at 2376, given that 24 number of data vectors are being measured and transmitted by each of the 99 nodes. Thus, it is observed that the less the number of clusters, the more is the percentage reduction in the communication overhead in the network.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

The objective of this research was to present a novel approach for cluster-based outlier detection in wireless sensor networks. The performance of the algorithm was studied and verified by conducting a number of experiments. A distributed approach was used to detect the outliers in the network itself as opposed to a centralized approach, wherein all the data from individual nodes is transferred to a central base station. The popular *k*-means clustering algorithm was used by every node to initially cluster its own local data. A summary of the clusters, and not the entire data set, is then transferred to each node's respective superpeer. It is at the superpeers that the *k*-means clustering operation is performed again, and finally an outlier detection algorithm based on the *K*-nearest neighbor algorithm is run. Each sensor node was required to maintain information regarding its respective superpeer and each superpeer was required to maintain information regarding its client nodes. The superpeer node was required to be a high capacity node, which is capable of performing more complex data processing tasks as compared with the clients.

Previous implementations of clustering sensor data based on *k*-means clustering paradigm did not deal with the outlier detection aspect. Also previous implementations of the *K*-nearest neighbor algorithm for outlier detection did not use the *k*-means clustering algorithm for partitioning the sensor data. Thus, this thesis proposes a new methodology to use a combination of the *k*-means and the *K*-nearest neighbor techniques in a sensor network environment for detecting outliers in the network data. The major challenge in

this thesis was that it was required to determine an optimal k value for clustering the data. This required the use of a trial and error method to determine the optimal k value for clustering. An initial system setup, with respect to the topology of the network, was needed for the successful running and verification of the approach.

The test data used in this thesis was based on real data obtained from weather.com website and using a random number generator to introduce outliers in the network. Experiments were conducted by varying parameters like the network size, the data set size, the value of k, etc., and the corresponding processing times, message rates, and detection rates were studied. Also, the distributed approach proposed in this thesis was compared with a centralized approach for cluster-based outlier detection based on several experiments. The entire system was implemented in a JAVA simulation. The purpose of this thesis was to accurately detect the outlier clusters in the sensor network and compare the performance of the distributed approach with a centralized system. Also, the proposed approach of outlier detection aims to reduce the message transmission rates or communication overhead as compared with the centralized approach.

Although the system proposed in this thesis performs well in most of the test cases, it is based on some very critical initial settings made. An effort can be made in the future to make the system more robust by considering the dynamism in the network, i.e. the addition and removal of nodes in the network. Also, the superpeers can be made to talk to each other, to provide a more sophisticated peer-to-peer network scenario. Another area that can be delved into would be clustering the superpeers so that they can form an additional tier in the network topology.

Several applications require detection of outliers in the sensor data. Outliers detected in the sensor data can be used to identify any malicious activity, intrusions, or misbehaviors in the network. The abnormal values detected need not necessarily be regarded as errors and discarded. This is because, in certain cases, they can be used to identify interesting events in the environment in which the sensors are deployed. The abnormal values detected can also be used for further analysis by a data analyst while trying to make interesting observations about the system. Thus, outlier detection in sensor networks is a growing area of research because of several critical applications that use the sensor information. Any contribution made to this field will go a long way in improving the system of data collection and processing.

REFERENCES

[1] D. Janakiram, A. M. Reddy and A. V. U. P. Kumar, "Outlier Detection in Wireless Sensor Networks using Bayesian Belief Networks," First International Conference on Communication System Software and Middleware, 2006, pp. 1-6.

[2] D. M. Hawkins, "Identification of Outliers," Monographs on Applied Probability and Statistics, Chapman and Hall, 1979.

[3] S. Rajasegarar, C. Leckie, M. Palaniswami and J. C. Bezdek, "Distributed Anomaly Detection in Wireless Sensor Networks," 10th IEEE International Conference on Communication Systems, 2006, pp. 1-5.

[4] S. Bandyopadhyay, C. Giannella, U. Maulik, H. Kargupta, K. Liu and S. Datta, "Clustering Distributed Data Streams in Peer-to-Peer Environments," International Journal on Information Sciences, vol. 176, issue 14, 2006, pp. 1952-1985.

[5] J. Han and M. Kamber, "Data Mining: Concepts and Techniques," San Francisco; Morgan Kaufmann Publishers, 2001.

[6] V. Barnett and T. Lewis, "Outliers in Statistical Data," 3rd Edition, Wiley Series in Probability and Mathematical Statistics, John Wiley and Sons, 1994.

[7] M. I. Petrovskiy, "Outlier Detection Algorithms in Data Mining Systems," Journal on Programming and Computer Software," vol. 29, issue 4, 2003, pp. 228-237.

[8] E. M. Knorr and R. T. Ng, "Algorithms for Mining Distance-Based Outliers in Large Datasets," Proceedings of the 24th Very Large Data Bases Conference, 1998, pp. 392-403.

[9] S. Ramaswamy, R. Rastogi and K. Shim, "Efficient algorithms for Mining Outliers from Large Data Sets," Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000, pp. 427-438.

[10] A. Montresor, "A Robust Protocol for Building Superpeer Overlay Topologies," in Proceedings of the Fourth International Conference on Peer-to-Peer Computing, 2004, pp. 202-209.

[11] http://www.weather.com/outlook/driving/interstate/hourbyhour/23508?from=36hr_fcstHourLink_driving

[12] E. Eskin, A. Arnold, M. Prerau, L. Portnoy and S. Stolfo, "A Geometric Framework for Unsupervised Anomaly Detection: Detecting Intrusions in Unlabeled Data," Journal on Applications of Data Mining in Computer Security, 2002, pp. 77-101.

[13] T. Palpanas, D. Papadopoulos, V. Kalogeraki and D. Gunopulos, "Distributed Deviation Detection in Sensor Networks," ACM SIGMOD Record, vol. 32, issue 4, 2003, pp. 77-82.

[14] J. Branch, B. Szymanski, C. Giannella, R. Wolff and H. Kargupta, "In-Network Outlier Detection in Wireless Sensor Networks," 26th IEEE International Conference on Distributed Computing Systems, 2006, pp. 51-58.

[15] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki and D. Gunopulos, "Online Outlier Detection in Sensor Data Using Non-parametric Models," Proceedings of the 32nd International Conference on Very Large Databases, 2006, pp. 187-198.

[16] T. Zhang, R. Ramakrishnan and M. Livny, "Birch: A New Data Clustering Method for Very Large Databases," in Proceedings of the ACM SIGMOD International Conference on Management of Data, 1996, pp. 103-114.

[17] I. Onat and A. Miri, "An Intrusion Detection System for Wireless Sensor Networks," in Wireless and Mobile Computing Networking and Communications, 2005, vol. 3, pp. 253-259.

[18] A. P. R. da Silva, A. A. F. Loureiro, M. H. T. Martins, L. B. Ruiz, B. P. S. Rocha, H. C. Wong, "Decentralized Intrusion Detection in Wireless Sensor Networks," in Proceedings of the 1st ACM international workshop on Quality of service and security in wireless and mobile networks, 2005, pp. 16-23.

[19] C. E. Loo and M. Y. Ng, C. Leckie and M. Palaniswami, "Intrusion Detection for Routing Attacks in Sensor Networks," International Journal of Distributed Sensor Networks, vol. 2, issue 4, 2006, pp. 313-332.

APPENDIX

A. Code for running k-means algorithm on all nodes

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Random;
import java.util.Vector;
import java.util.Iterator;

public class PrgMain {
   static int [] faultNodes = new int [] {5, 15, 25, 35, 45, 55, 65, 75, 85, 95};
        static int firstRandom = 72;
        static int secondRandom = 73;
        static int thirdRandom = 74;
        static Random rand = new Random();

        static int nextNumberType() {
                return Math.abs(rand.nextInt()) % 2;
        }

        static int nextNumber() {
                return Math.abs(((rand.nextInt() % 5))) + 1;
        }

        static boolean isFaultNode(int number) {
                boolean faultNode = false;
                for (int i = 0; i < faultNodes.length; i++) {
                        if (faultNodes[i] == number) {
                                faultNode = true;
                                break;
                        }
                }
                return faultNode;
        }

   public static void main (String args[]) {
        deleteFiles();
                long startTime = System.currentTimeMillis();
```

```
            for (int nIndex = 0; nIndex < 100; nIndex++) {
          BufferedReader reader = null;
          Vector <DataPoint> dataPoints = new Vector <DataPoint> ();
            int numType = nextNumberType();
          try {
            if (isFaultNode(nIndex)) {
                  if (numType == 0) {
                        reader = new BufferedReader(new
FileReader("C:\\Documents and Settings\\Swetha\\My
Documents\\analysis\\distributed\\NorfolkWeatherDataBogusLow.txt"));
                  }
                  else {
                        reader = new BufferedReader(new
FileReader("C:\\Documents and Settings\\Swetha\\My
Documents\\analysis\\distributed\\NorfolkWeatherDataBogusHigh.txt"));
                  }
            }
            else {
                  reader = new BufferedReader(new FileReader("C:\\Documents and
Settings\\Swetha\\My Documents\\analysis\\distributed\\NorfolkWeatherData.txt"));
            }
                for (int i = 0; i < 24; i++) {
                              String line = reader.readLine();
                              String [] splits = line.split("\t");
                              if (splits.length > 0)
                              {
                                    numType = nextNumberType();
                                    firstRandom =
Integer.parseInt(splits[0].trim());
                                    secondRandom =
Integer.parseInt(splits[1].trim());
                                    thirdRandom =
Integer.parseInt(splits[2].trim());

                                    int nNum = nextNumber();
                                    firstRandom = (numType == 0)?
Math.abs((firstRandom - nNum)) : (firstRandom + nNum);
                                    secondRandom = (numType == 0)?
Math.abs((secondRandom - nNum)) : (secondRandom + nNum);
                                    thirdRandom = (numType == 0)?
Math.abs((thirdRandom - nNum)) : (thirdRandom + nNum);
                              }
                              else {
                                    System.out.println("Splitting on what?");
                              }
```

```
                        dataPoints.add(new DataPoint(firstRandom, secondRandom,
thirdRandom, "" + i + ":00"));
                    }
                    } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                    }
                    finally {
                        try {
                            reader.close();
                        } catch (IOException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                    }

            JCA jca = new JCA(4, 1000, dataPoints);
            jca.startAnalysis();

            Vector[] v = jca.getClusterOutput();
            BufferedWriter writer = null;
            String outFileName = "";
            if (nIndex < 20) {
              outFileName = "AggregatedData20.txt";
            }
            else if (nIndex >= 20 && nIndex < 40) {
              outFileName = "AggregatedData40.txt";
            }
            else if (nIndex >= 40 && nIndex < 60) {
              outFileName = "AggregatedData60.txt";
            }
            else if (nIndex >= 60 && nIndex < 80) {
              outFileName = "AggregatedData80.txt";
            }
            else if (nIndex >= 80 && nIndex < 100) {
              outFileName = "AggregatedData100.txt";
            }
            try {
                        writer = new BufferedWriter(new FileWriter(new
File(outFileName), true));
                        writer.write("Node " + (nIndex + 1) + ": ");
                        for (int i=0; i<v.length; i++) {
                    Vector tempV = v[i];
                    System.out.println("----------Cluster"+i+"---------");
                    Iterator iter = tempV.iterator();
                    int j = 0;
```

```
                        while(iter.hasNext()){
                                j++;
                           DataPoint dpTemp = (DataPoint)iter.next();
                           System.out.println(dpTemp.getObjName() + "[" +
dpTemp.getX() + "," + dpTemp.getY() + "," + dpTemp.getZ() + "]");
                        }
                        System.out.println("Number of data vectors: " + j);
                        writer.write("Number of data vectors: " + j + "\n");
                        Centroid cd = jca.getCluster(i).getCentroid();
                        System.out.println("Centroid: [" + cd.getCx() + ", " + cd.getCy() +
", " + cd.getCz() + "]");
                        writer.write("" + cd.getCx() + ", " + cd.getCy() + ", " + cd.getCz()
+ "\n");
                    }
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        finally {
                                try {
                                        writer.close();
                                } catch (IOException e) {
                                        // TODO Auto-generated catch block
                                        e.printStackTrace();
                                }
                        }
                }
                Thread t1 = new SuperPeerThread(new String[] {"1", "5",
"AggregatedData20.txt"});
                t1.start();
                Thread t2 = new SuperPeerThread(new String[] {"2", "5",
"AggregatedData40.txt"});
                t2.start();
                Thread t3 = new SuperPeerThread(new String[] {"3", "5",
"AggregatedData60.txt"});
                t3.start();
                Thread t4 = new SuperPeerThread(new String[] {"4", "5",
"AggregatedData80.txt"});
                t4.start();
                Thread t5 = new SuperPeerThread(new String[] {"5", "5",
"AggregatedData100.txt"});
                t5.start();
                try {
                        t1.join();
                        t2.join();
                        t3.join();
```

```
                t4.join();
                t5.join();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            long endTime = -1;
            endTime = System.currentTimeMillis();
            System.out.println("Time Difference: " + (endTime - startTime) + " milli
seconds");
    }

    static void deleteFiles() {
            // TODO Auto-generated method stub
            File f1 = new File("AggregatedData20.txt");
            File f2 = new File("AggregatedData40.txt");
            File f3 = new File("AggregatedData60.txt");
            File f4 = new File("AggregatedData80.txt");
            File f5 = new File("AggregatedData100.txt");

            f1.delete();
            f2.delete();
            f3.delete();
            f4.delete();
            f5.delete();
    }
}
```

B. Code for running outlier detection algorithm on superpeers

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.util.Collections;
import java.util.Random;
import java.util.Vector;
import java.util.Iterator;

public class SuperPeer {
```

```
public void superPeerMain (String args[]) throws IOException {
        double firstRandom = 72;
        double secondRandom = 73;
        double thirdRandom = 74;
        Random rand = new Random();

        int superPeerId = Integer.parseInt(args[0]);
        int kValue = Integer.parseInt(args[1]);
      Vector <DataPoint> dataPoints = new Vector <DataPoint> ();
      BufferedReader reader = null;
      BufferedWriter writer = new BufferedWriter(new FileWriter(new File("SuperPeer"
+ superPeerId + ".txt")));

                try {
                        reader = new BufferedReader(new FileReader(args[2]));
                        String line = null;
                        int i = 0;
                        while ((line = reader.readLine()) != null) {
                                String [] splits = line.split(",");
                                if (splits.length > 1)
                                {
                                        firstRandom =
Double.parseDouble(splits[0].trim());
                                        secondRandom =
Double.parseDouble(splits[1].trim());
                                        thirdRandom =
Double.parseDouble(splits[2].trim());
                                dataPoints.add(new DataPoint(firstRandom, secondRandom,
thirdRandom, "" + i + ":00"));
                                }
                }
                } catch (IOException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                finally {
                        try {
                                reader.close();
                        } catch (IOException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }

      JCA jca = new JCA(10, 1000, dataPoints);
```

```
    jca.startAnalysis();

    Vector[] v = jca.getClusterOutput();
    System.out.println("----------For the super peer: " + superPeerId + "----------");
    writer.write("----------For the super peer: " + superPeerId + "----------\n");
    Vector<Double> icds = new Vector<Double>();
    for (int i=0; i<v.length; i++) {
       Vector tempV = v[i];
       System.out.println("Cluster " + i);
       writer.write("Cluster " + i + "\n");
       Iterator iter = tempV.iterator();
       while(iter.hasNext()){
          DataPoint dpTemp = (DataPoint)iter.next();
          System.out.println("[" + dpTemp.getX() + "," + dpTemp.getY() + "," +
dpTemp.getZ() + "]");
          writer.write("[" + dpTemp.getX() + "," + dpTemp.getY() + "," +
dpTemp.getZ() + "]\n");
       }
       Centroid cd = jca.getCluster(i).getCentroid();
       System.out.println("Centroid: [" + cd.getCx() + ", " + cd.getCy() + ", " +
cd.getCz() + "]");
       writer.write("Centroid: [" + cd.getCx() + ", " + cd.getCy() + ", " + cd.getCz() +
"]\n");

       Double tempIcd = new Double(0);
       tempIcd = getInterClusterDistance(jca, i, kValue);
       icds.add(tempIcd);
    }

    double avg_icds = 0;
    for (int i = 0; i < icds.size(); i++) {
       avg_icds += icds.get(i).doubleValue();
    }
    avg_icds /= icds.size();

    double variance_icds = 0;
    for (int i = 0; i < icds.size(); i++) {
       variance_icds += Math.pow((icds.get(i).doubleValue() - avg_icds), 2) /
icds.size();
    }

    double std_deviation_icds = Math.sqrt(variance_icds);

    for (int i = 0; i < v.length; i++) {
       if (icds.get(i).doubleValue() > (avg_icds + std_deviation_icds)) {
             System.out.println("This cluster is an anamoly");
```

```
                writer.write("This cluster is an anamoly\n");
                Vector tempV = v[i];
                System.out.println("Cluster " + i);
                writer.write("Cluster " + i + "\n");
                Iterator iter = tempV.iterator();
                while(iter.hasNext()){
                   DataPoint dpTemp = (DataPoint)iter.next();
                   System.out.println("[" + dpTemp.getX() + "," + dpTemp.getY() + "," +
dpTemp.getZ() + "]");
                   writer.write("[" + dpTemp.getX() + "," + dpTemp.getY() + "," +
dpTemp.getZ() + "]\n");
                }
                Centroid cd = jca.getCluster(i).getCentroid();
                System.out.println("Centroid: [" + cd.getCx() + ", " + cd.getCy() + ", " +
cd.getCz() + "]");
                writer.write("Centroid: [" + cd.getCx() + ", " + cd.getCy() + ", " +
cd.getCz() + "]\n");
            }
        }
        writer.close();
    }

    public double getInterClusterDistance(JCA jca, int i, int kValue) {
        double icd = 0;
        Vector <Double> tempIcds = new Vector<Double>();
        Centroid curCd = jca.getCluster(i).getCentroid();
        DataPoint curDataPt = new DataPoint(curCd.getCx(), curCd.getCy(),
curCd.getCz(), "dummy");
        for (int j = 0; j < jca.getKValue(); j++) {
            if (i != j) {
                Centroid cd = jca.getCluster(j).getCentroid();
                tempIcds.add(curDataPt.testEuclideanDistance(cd));
            }
        }
        Collections.sort(tempIcds);
        for (int j = 0; j < kValue; j++) {
            icd += tempIcds.get(j).doubleValue();
        }
        return icd / kValue;
    }
}
```

VITA

NAME: Swetha Gali

DEGREES:

- Bachelor of Engineering (Electrical and Electronics Engineering), MVSREC, Osmania University, India, June 2005.
- Master of Science (Electrical Engineering), Old Dominion University, Norfolk, Virginia, December 2007.