WSN Integrated Cloud Computing for N-Care System (NCS) Using Middleware Services

Chandrakant N, Bijil A P, Puneeth P, Deepa Shenoy P, Venugopal K R, L M Patnaik

Abstract: The number of wireless devices with powerful sensing capabilities is constantly growing. A mobile phone is an example of a device that is packed with several powerful sensors. Cloud computing is another area that been in focus over the last decade. Cloud computing can be defined as an architectural abstraction that provides scalability and reliability based on requirement. The challenge lies in the fact that sensors for different purposes are heterogeneous in nature. We propose a framework called the N-Care System that utilizes heterogeneous wireless networks to collect data, cloud services to provide additional computational capabilities and provides information for different types of end users. A wireless sensor network consisting of sensors that possess both sensing and transmitting capabilities forms a communication back-bone that can capture a wide variety of data. Multiple sensors are grouped in to a cluster that consists of an internet capable computing device called cluster head that collects data from the constituent sensor nodes and pushes it in to a cloud based database. End users can log in and access data from sensors that fall under the user's domain.

Keywords: Middleware, WSN, MANET, NCS, Cluster, Cloud

I. INTRODUCTION

Today, the number of devices with powerful sensing capabilities is consistently growing. Devices like smart phones are an amalgamation of multiple sensors. Mobile phones, sensors and electronic devices today capable of connecting to the internet and sharing information. However the sensors are heterogeneous in nature and there is a paucity of frameworks that are robust enough to handle the heterogeneity of the sensors. The sensors made by different vendors for different purposes differ not in functionalities but also in semantics, which makes integration heterogeneous sensors a challenge.

In this paper we focus on developing an N Care System (NCS) that can handle heterogeneous sensors and is capable of catering to different types of end users. For instance, smart cities of the future will have a distributed network of myriads of sensor nodes that measure different parameters for efficiently managing a city.

Such cities will have different end users, who are interested only in the data related to their domain. Traffic police officers will supervise the data useful for effectively managing vehicular traffic. Citizens can monitor electricity usage and keep an eye on the elderly and children back at home, while they work. Cloud computing is a technology that uses the internet and remote servers to maintain data and applications. Cloud computing provides a platform for delivering applications and other services over the internet along with scalability and virtualization. Cloud computing has three possible delivery modes. They are namely, Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Cloud Computing provides the perfect environment for integrating heterogeneous sensor nodes and data storage. The scalability offered by cloud ensures that networks can be truly ad-hoc and the framework can handle increase in load without exhibiting any drastic effects that dampen its performance. Virtualization provides scope for outsourcing the processing from sensors, which have limited resources compared to the cloud that has almost unlimited resources. Cloud also offers large amounts of storage space and services such as database management systems for managing the large volumes of data. The data transmitted by the sensors can be stored on the cloud, for both analytic and archiving purposes. The NCS will allow end users to view real time status of sensors they are in charge of. Each end user will be authenticated using their login credentials. The end users will be shown information strictly related to their domain. For example, when a traffic policeman logs in, he will have access to data from all the traffic related sensors in his circle.

II. RELATED WORK

Of late, Cloud Computing has become the topic of extensive research. There is a considerable amount research on Cloud based WSNs and MANETs, which basically integrates mobile wireless device networks and Cloud based services. Some of the existing works will be discussed in this section.

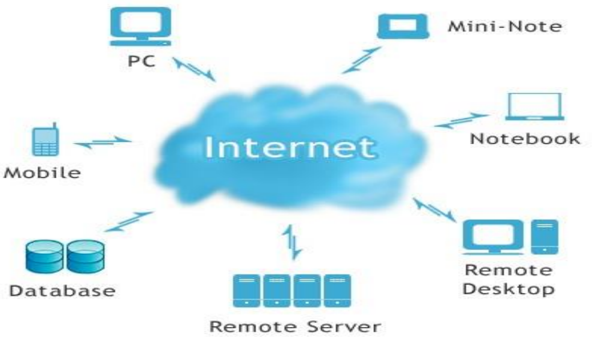


Figure 1. Overview of Cloud Computing

Manuscript published on 30 June 2013.
***Correspondence Author(s)**

Chandrakant N, Department of Computer Science and Engineering University Visvesvaraya College of Engineering Bangalore University, India.

Bijil A P, Department of Computer Science and Engineering University Visvesvaraya College of Engineering Bangalore University, India.

Puneeth P, Department of Computer Science and Engineering University Visvesvaraya College of Engineering Bangalore University, India.

Deepa Shenoy P, Department of Computer Science and Engineering University Visvesvaraya College of Engineering Bangalore University, India.

Venugopal K R, Department of Computer Science and Engineering University Visvesvaraya College of Engineering Bangalore University, India.

L M Patnaik, Honorary ProfessorIndian Institute of Science, Bangalore, India

Cloud Computing Architecture [8] introduces cloud computing at a grass root level. It explains the cloud computing architecture and software architecture for cloud.

The Sensor-Cloud Infrastructure [18] visualizes a physical sensor as a virtual sensor on the cloud computing. Dynamic grouped virtual sensors on cloud computing can be automatic provisioned when the users need them. This approach enables the sensor management capability on cloud computing. Since the resource and capability of physical sensor devices is limited, the cloud computing can improve the performance of physical sensors.

Mobile Cloud Computing (MCC)[13] is a well accepted concept that aims at using cloud computing techniques for storage and processing of data on mobile devices, thereby reducing their limitations.

Hyrax [11] allows client applications to conveniently utilize data and execute computing jobs on networks of smart phones and heterogeneous networks of phones and servers. By scaling with the number of devices and tolerating node departure, Hyrax allows applications to use distributed resources abstractly, oblivious to the physical nature of the cloud. Securing Elastic Applications on Mobile Devices for Cloud Computing [15] aims to build elastic applications which augment resource-constrained platforms, such as mobile phones, with elastic computing resources from clouds. An elastic application consists of one or more weblets, each of which can be launched on a device or cloud, and can be migrated between them according to dynamic changes of the computing environment or user preferences on the device. MobiCloud [6] provides computational intensive services to mobile users. It also monitors and collects mobile ad hoc networking status by considering mobile devices as service nodes. It also presents a new class of applications that can be developed using the extended processing power and more effective communication of MobiCloud. The paper [1] provides a definition of what a cloud database management (CDBMS) is and how it differs from the traditional DBMS. The article [2] explains how the concepts of WSN can be used in conjunction with a Cloud system to create a powerful and highly efficient WSN. A Comparison of Application Models [10] surveys the existing work in mobile computing through the prism of cloud computing principles. It gives a definition of mobile cloud computing and provides an overview of the models of mobile cloud applications. It also highlights the challenges in the area of mobile cloud computing. Mobile Cloud Computing [14] explains a middleware that provides a personal service mashup platform for the mobile client. Finally, the middleware can be deployed on Cloud Platforms, like Google App Engine and Amazon EC2, to enhance the scalability and reliability. The experiments evaluate the optimization/adaptation, overhead of the middleware, middleware pushing via email, and performance of Cloud Platforms. Calling the cloud [5] presents a middleware platform that can automatically distribute different layers of an application between the phone and the server, and optimize a variety of objective functions (latency, data transferred, cost, etc.). It builds on existing technology for distributed module management and does not require new infrastructures. Secured Wireless Sensor Network-integrated Cloud computing for u-Life Care (SC3)[16] monitors human health, activities, and shares information among doctors, care-givers, clinics, and pharmacies in the Cloud, so that users can have better care with low cost. SC3 incorporates various technologies with novel ideas including; sensor networks, Cloud computing security, and activities recognition. Mobile Cloud Computing as a future of mobile multimedia database [9] focuses on the key issues of Mobile Multimedia Database Systems and the issues that need to be taken into consideration in the future. Cloud computing has the potential to save mobile client energy but the savings from offloading the computation need to exceed the energy cost of the additional communication. Energy efficiency of mobile clients in cloud computing [12] provides an analysis of the critical factors affecting the energy consumption of mobile clients in cloud computing. It provides measurements about the central characteristics of contemporary mobile hand held devices that define the basic balance between local and remote computing. In [7] and [17], described about Emergency Communication and Information System for Catastrophic Natural Disasters, numerous disasters are that mobile communication system is vulnerable and the loss of communication system may have a catastrophic consequence that analyzes the causes that paralyzed the entire communication systems in Jixi Earthquake and proposes a P2Pnet that uses notebook PCs to construct a MANET based emergency communication and information system. Brief system requirements and system design are presented. A prototype of Disastrous Earthquake Rescue Information System is presented in this paper. Middleware Service Oriented Rescue and Crime Information System (RCIS) Using Heterogeneous Fixed Nodes in WSNs [3] consists of a Wireless Sensor Network that consists of heterogeneous nodes that have different functionality. These nodes are distributed across the environment and they send the outcome of events that occur in nature. In this network each node has to send a request/response to a Base Node (BN) where all requests/responses are evaluated and the required values are computed for the end user. The noticeable out come values are passed on to a special system called Rescue and Crime Information System (RCIS) where all filtered and computed values are displayed in this system with severity of events and required information. Rescue and Crime Information on Cloud (RCIC) using heterogeneous nodes in WSNs [4] consists of sensor nodes that together form a cloud based MANET. Such a network of sensors depends on the cloud for additional computational power, storage and management of data. The data obtained by the sensors are sent to the cloud, where it is processed, analysed and normalized by the Middleware. RCIC will aims at assisting in detection of occurrences of natural and man-made disasters along with criminal activities. RCIC will help concerned authorities to keep tabs on such events and respond in times of an emergency.

III. CLOUD COMPUTING AND CLOUD MIDDLEWARE

Cloud Computing System:

A cloud computing system can roughly be bifurcated into two sections: the front end and the back end. The front end is the client and back end includes the cloud services offered by the system. The front end includes the client’s computer (or computer network) and the application required to access the cloud computing system.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

The back end of the system consists of various computers, servers and data storage systems that create the "cloud" of computing services. Theoretically, a cloud computing system could include practically any computer program you can imagine, from data processing to video games. Ideally, each application is assigned its own dedicated server.

A central server administers the system, monitoring traffic and client demands to ensure everything runs smoothly. It follows a set of rules called protocols and uses a special kind of software called middleware. Middleware allows networked computers to communicate with each other. Usually servers do not run at full capacity. That means there's unused processing power going to waste. The technique is called server virtualization. A cloud computing system uses a technique called server virtualization to reduce the need for more physical machines. Virtualization is an illusion created by a server of having multiple servers, each having its own independent operating system.

IV. CLOUD MIDDLEWARE:

Cloud middleware is a software that enables seamless integration between various services, to harness the maximum potential of the available resources. It is the integration software that is hosted on a network to be readily available for interconnection of various software components or applications that are a part of the cloud. The cloud middleware is a major component of cloud computing as complex applications on a cloud need to work in unison. This software facilitates the re-use of applications and web components that are hosted anywhere on the cloud.

Cloud middleware consists of two components, the user-level middleware and the core middleware. The user-level middleware provides tools and environments for cloud programming such as mashups, workflows, libraries and web 2.0 interfaces. Core middleware offers platforms such as Virtual Machines, VM management and deployment. The resource management layer is used to manage the resources. The resource management layer also coordinates the resource sharing based on application needs, passed through the upper layers. Services provided by upper layers may need some resource sharing support, which is encapsulated in the communication layer. As an application uses such a service, the corresponding layer asks for the communication layer to manage the access control of the required resources. Indeed, the resource management layer commands the allocation and adaptation of resources, such that the QoS requirements specified by the applications can be met.

Figure 2 shows the cloud middleware. It shows how the users at the top most level can access different services offered by the cloud. The IaaS middleware offers the users access to the virtualisation layer, the physical layers, the networking layer and to the data centre. The PaaS middleware includes the services offered by IaaS and provides additional access to operating systems and application server platform. The SaaS middleware encompasses all the services offered by IaaS and PaaS and it provides direct access to the UI, BPM, SOA and other similar software objects.

Figure 3 shows the general cloud architecture. The cloud can be segregated into seven major blocks: infrastructure, cloud runtime, service, application, client infrastructure, management and security.

V.PROBLEM DEFINITION

There has been immense growth in sensor technologies. We are surrounded by devices that sensing ability. A mobile phone is a collection of multiple sensors. Cars today come equipped with multiple sensors such as collision warning sensors. There are different types of sensors for different purposes. Each kind of sensor is heterogeneous in nature. They differ in functional- ity, semantics and protocols used. They also differ in their processing capacities and battery life. Because of the differences in sensors, integration of heterogeneous sensors is a challenge.

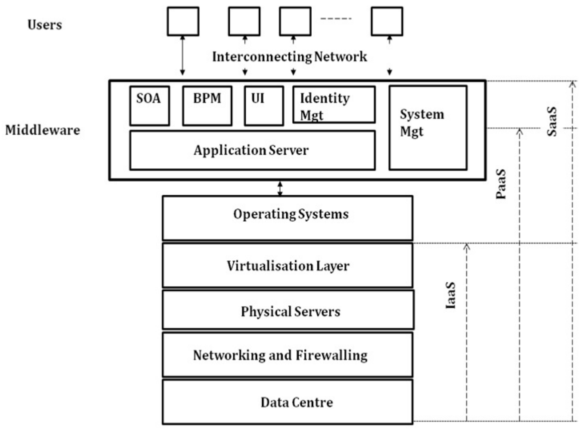


Figure 2. Cloud Middleware.

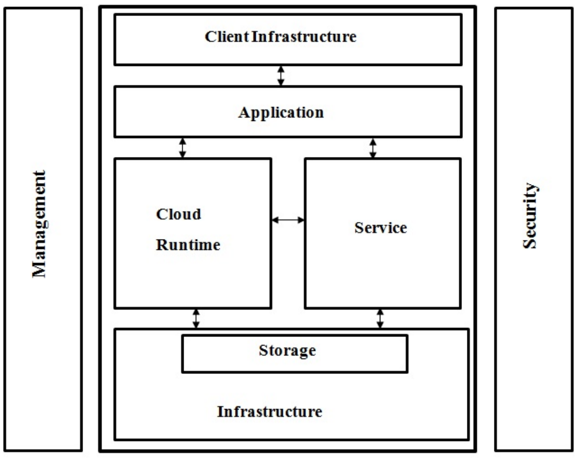


Figure 3. General Cloud Architecture.

Sensors can greatly aid us in our daily lives. For this purpose integration of heterogeneous sensors becomes essential. Additionally, multiple types of end users may need to share a subset of the available sensor data. Integration of multiple sensors will produce large amounts of data. Managing, accessing and filtering of data is another challenge.

Often sensors are minuscule in size and have limited processing power and battery life. But there are applications where additional processing capabilities are needed. Presently there is a paucity of frameworks that can handle heterogeneous sensors and possess ability to handle variable load. Over the last few years, Cloud Computing has become a subject of extensive research. Cloud computing offers computing as a service.

Cloud computing provides the perfect platform for developing frameworks that are robust enough to handle heterogeneous clients (read WSN), handle differing volumes of traffic and provide large volumes of on demand storage spaces.

Using Cloud computing we can develop frameworks that maintain data from a wide variety of large number of sensors that have different purposes. We can ensure that this large volume of data is handled properly and segregated for different type of sensors. Such a framework can become the building blocks of future super cities that rely on a communication back bone of WSNs that communicate with each other, gather data and send it back for further processing, analysis or directly to the end users. Such a framework will require on demand access to additional servers for additional processing capabilities, on demand storage for storage the enamors volumes of data collected from the sensors and ability to deploy applications that can process the required data on demand. Additionally the framework should also be scalable to accept variable amount of sensors.

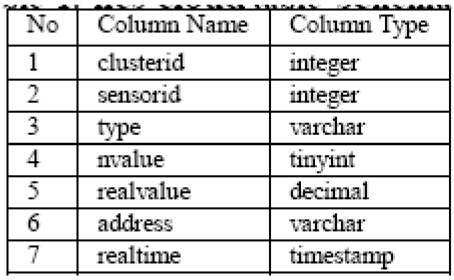
4.1 Assumptions

We have assumed the following parameters for this paper:

1. A WSN can have heterogeneous nodes, meaning each node consists of dissimilar or diverse functionality.

2. Each node can send/receive packets to/from other nodes or neighbors other than base/header node.

Table 1. ncs cloud table schema



No	Column Name	Column Type
1	clusterid	integer
2	sensorid	integer
3	type	varchar
4	nvalue	tinyint
5	realvalue	decimal
6	address	varchar
7	realtime	timestamp

Table 2. Severity and its Attention Rate

No	Severity Rate(%)	Attention Type
1	0-25	Low
2	26-50	Medium
3	51-75	High
4	76-100	Very High

3. This network is dependent on Cloud middleware virtualization and normalization services for WSNs.

4. The sensor nodes in the network are not necessarily fixed in the area hence there could be topology changes over a time.

5. The sensor nodes in the network send back values to a base station that is connected to a cluster head.

6. A cluster head is assumed to have limited storage and processing capacity. A cluster head is connected to the internet and has a database capable of storing the real time sensor values.

7. Few utilities/applications of Image Processing(IP)/Video Processing(VP)/Audio Processing(AP) have been treated as upcoming techniques and evaluated accordingly.

VI. ALGORITHM

This section focuses on the basic algorithms used for implementation.

6.1 Cluster Daemon

```
Algorithm 1 SetSeverityRateCloud(type)
1:  connectToCloudDatabase(NCS)
2:  query ← select * from ncs where nvalue is NULL
        and type=type
3:  resultset ← execute(query)
4:  while resultset.hasnext() do
5:      cid ← resultset.get('clusterid')
6:      sid ← resultset.get('sensorid')
7:      rtime ← resultset.get('realtime')
8:      rval ← resultset.get('realvalue')
9:      nval ← normalise(rval,type)
10:     query ← select * from ncs where nvalue is
            NULL and type=type
11:     execute(uquery)
12: end while
13: disconnectFromCloudDatabase(NCS)
```

The result is parsed to obtain the array of tuples. Since the number of fields is dynamic, the fields have to be set based on the descriptor. Since the transformation is not yet done, it is important that the type of the column or field is passed, as the transformation is handled by another module.

Algorithm 5 is used to parse the XML and create a configObject that is used by the cluster daemon for processing. This processing involves obtaining the client and the table and field elements along with other attributes. Algorithm 6 constructs the query string. This construction ensures that no other field details are extracted unnecessarily. XML parser is one of the critical piece of code 5 that retrieves the information about the cluster daemon. In the first step we find the path to the configuration file and set the XML document parser provided by java. In step 2 to 10 of algorithm5 we read and set up the cloud configuration. The steps 11 to 20 are repeated for every table tag present in the configuration file. The field tag loop from steps 21 to 27 are repeated for every table loop and all the details are retrieved and set. The transformation module process the fetched records from the data extractor. the transformation works only on numerical data. this is taken care with the help of the type attribute. The algorithm then processes the individual fields per sensor per cluster and normalizes them. this process can be done on the cloud end but if the client is powerful then this processing can reduce the load on the cloud. Data extraction is done by the client module of the cluster daemon and is most process intensive. In the step 1 of the algorithm 6 the configuration object is sent to the Connector class which returns suitable driver for the required sensor. The next few steps 2 through 8 involves,

```
Algorithm 2 GetAttentionType(severityRate)
Require: Initialize attentionType ← low, i ← 0
1:  if severityRate ! = null then
2:      if severityRate >= 0 AND severityRate <25
then
3:          attentionType ← Low
4:      end if
5:  else
6:      if severityRate >=25 AND
severityRate <50
then
7:          attentionType ←
```

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

```
Medium
8:    end if
9:  else
10:   if severityRate >=50 AND severityRate <75 then
11:     attentionType ← High
12:   end if
13:  else
14:   if severityRate >=75 then
15:     attentionType ← VeryHigh
16:   end if
17:   return attentionType
18:  end if
```

construction of the query string. This is done with the help of TableInfo object that has the information of the number of fields per sensor. Once the query is constructed the result is obtained and the data is stored to individual tuples. The tuples have metadata along with the field values step 12 to 16 of the algorithm6. This module returns an array of tuples that must be sent to the cloud.

The minimum and the maximum threshold per sensor is obtained from the descriptor and this help in normalization. Apart from this there are other parameters like min-threshold and max threshold which provide us with permissible values.

Data cleaning is a critical task as sensors obtain a large volume of data. it becomes crucial that these records are cleared. Since the cloud stores all the information the backup copy of these data only increases the resources handled by the cluster.

DataCleaner is a client module that cleans the sensor table so that there cluster doesn't have redundant information. This module is executed only if the data has been successfully written into the cloud. Step 1 of algorithm 7 is to obtain the tuples that have been sent. The following steps involve obtaining the connection to the sensor database and deleting the particular record. DataSender is a server module in the cluster daemon. The cluster daemon receives the tuples from DataSender 6. Step 2 involves obtaining the connection to the cloud. This is done with the help of server Connector. Since the sensor Schema and the cloud Schema are very different it so happens that we need to transform it. Steps 4 to 14 explain how the transformation takes place. Some extra fields are added so that the cloud can understand the details of the cluster. Step 15 is the final step that writes the transformed tuple into the cloud database.

Algorithm 3 CloudToEndUser(usertype)

```
Require: Displaying on Browser
1:  connectToCloudDatabase(NCS)
2:  typelist ← getListofTypes(usertype)
3:  query ← select * from ncs n where type in (typelist) and nvalue is not null and
      realtime=(select max(realtime) from ncs n2 where n.clusterid=n2.clusterid and
      n.sensorid=n2.sensorid group by n2.sensorid,n2.clusterid)
4:  resultset ← execute(query)
5:  while resultset.hasnext() do
6:      cid ← resultset.get('clusterid')
7:      sid ← resultset.get('sensorid')
8:      rtime ← resultset.get('realtime')
9:      rval ← resultset.get('realvalue')
10:     nval ← resultset.get('nvalue')
11:     atype ← getAttentionType(nvalue)
12:     type ← resultset.get('type')
13:     address ← resultset.get('address')
14:     display(cid,sid,address,type,atype,rtime)
15: end while
16: disconnectFromCloudDatabase(NCS)
```

In this paper we use a combination of real sensors and simulated sensors. RandomGenerator() is an inbuilt function provided by the programming language. Sensor readings are seldom erratic. For instance, the pulse of a person never jumps from 70 to 140, falls to 45 and then jumps to 96 in a time frame of three minutes. Usually it stays in a range of between 60-90. increased physical activity will cause it to rise above 90. But it would remain in this state for something, even if the physical activity decreases. A sensor will always have maximum and minimum values it can measure. This is represented by the maxval and minval constants. A quantity will also have optimum range. For pulse it is between 60-90. Here the range is represented by minthres to maxthres. In order to ensure there no erratic variation in the readings we calculate the odds that the reading will fall within the optimum range or vice versa. A random number is generated and based on the value generated, the sensor reading is randomly generated to fall within in the optimum range or below it or above it.

Algorithm 4 DataTransformation

```
1:  Obtain the tuples that have to be transformed.
2:  for t ∈ records.getRecords() do
3:    Tuple te = new Tuple(t);
4:    for i = 0 → myTable.getFields().size() do
5:      if myTable.getFields().get(i).getType() ≤ Tuple.integer then
6:        min ← myTable.getFields().get(i).getMin()
7:        max ← myTable.getFields().get(i).getMax()
8:        x ← (t.value[i] − min) ∗ 100.0f/max
9:        te.value[i] ← x;
10:     end if
11:   end for
12:   tRecords.getRecords().add(te);
13: end for
14: ret tup
```

6.2 Cloud

Table 1 shows the schema for the master table ncs that is stored on the cloud. It stores details of all sensor nodes of all clusters that are a part of the NCS. Each tuple in this table contains a cluster id, sensor id, type of sensor, real value (or reading) of the sensor, address where the sensor is located and the time when that particular reading was taken. Each tuple represents a reading of a sensor. Each tuple is uniquely identified using a combination of cluster id, sensor id and the time stamp of the reading.

Algorithm 1 is used to set the severity rate for each reading in the NCS table. The algorithm takes type, the type of sensors that the particular program caters to as the sole parameter. A connection to the cloud database is first established. When the cluster daemon writes into the NCS table it leaves n value attribute value blank.

Algorithm 1 scans the latest entries in the table for all tuples that have the n value attribute null. Based on the type of sensor, a normalize function is called to convert the value of the sensor's value to 0-100 scale, called the severity rate. The corresponding row is updated and the nvalue is set to the severity rate.

Algorithm 2 assigns an attention type to an event based on the severity rate according to table 2. A low or medium attention type indicates that the readings from the sensor is normal. A high attention type indicates that the sensor reading has crossed the threshold value and some action is required to restore normalcy. A very high attention type indicates that the sensor readings are well beyond the threshold values and some immediate action is required.

Algorithm 5 XML Parser

```
Create a Document parser and parse the XML file document.
root ← document.getRootElement()
clouddbName ← root.getAttribute(clouddnnameAttr)
password ← root.getAttribute(passwordAttr)
userName ← root.getAttribute(userNameTag)
connectionStr ← root.getAttribute(connectionStrAttr)
myConf.setClouddbName(clouddbName)
myConf.setConnectionStr(connectionStr)
myConf.setUserName(userName)
myConf.setPassword(password)
Get all the table nodes to dbs.
while i = 0 →dbs.size() do
    name ← table.getAttribute(nameAttr);
    password ← table.getAttribute(passwordAttr)
    userName ← table.getAttribute(userNameTag)
    connectionStr ← table.getAttribute(connectionStrAttr)
    dbName ← table.getAttribute(dbNameAttr)
    interval ← table.getAttribute(intervalAttr)
    dataset ← table.getAttribute(datasetAttr)
    Get all the field nodes to fds.
    while i = 0 →dbs.size() do
        fname ← field.getAttribute(nameAttr)
        fmin ← field.getAttribute(minAttr)
        fmax ← field.getAttribute(maxAttr)
        ftype ← field.getAttribute(typeAttr)
        sensor ← field.getAttribute(sensorAttr)
    end while
end while
```

Algorithm 3 fetches the data of those sensors, whose type falls under the domain of that particular user from the ncs master table. This algorithm takes the user type as the input parameter. Based on the user type, list of sensor types is generated, which is basically all the types of sensors that fall under the user's domain. Since the ncs has multiple tuples with the same sensor id and cluster id, that are differentiable by their time stamp, the reading with the latest time stamp is chosen and displayed.

Algorithm 6 DataExtraction

```
Parse the XML descriptor and obtain the ClientDBConfig
mytable ← dbConfig.getTableInfo()
query ← "select "
Tuples ← Tuples [result.size]
j ← 0
while i = 0tomytable.getLength() do
    query ← mytable.getFields().get(i).getFieldName()
end while
connection ← getConnection(mytable)
result ← connection.executeQuery(query)
while result.hasnext() do
    t ← new Tuple()
    t.col ← String[mytable.getFields().size]
    t.type ← String[mytable.getFields().size]
    t.value ← Object[mytable.getFields().size]
    t.id ← Boolean[mytable.getFields().size]
    while i = 0tomytable.getLength() do
        t.col[i] ← mytable.getFields().get(i).getFieldName()
        t.type[i] ← String[mytable.getFields().size]
        t.value[i] ← result.getObject(t.col[i])
        t.id[i] ← mytable.getFields().get(i).isSensor()
    end while
    Tuples[j + +] ← t
end while
return Tuples
```

VII. SYSTEM DESIGN

The overview of system design is shown in 4. The NCS contains of four main parts, i.e., Sensors/Detectors unit, Clusters ,Cloud unit and the End User Clients.

Sensors Unit: Sensor unit consists of sensors and detectors. Sensors and detectors are electronic devices that are designed to detect/ sense or quantitatively determine physical factors such as temperature, pressure, position or velocity over a measuring assortment. Many devices require accurate measurement sensors to calculate pressure, position, speed, acceleration or volume. These values are conveyed to gauges or to a computer microprocessor to either give a visual readout of the measured value or to input the computer program to direct a change in state or to activate another mechanical device. With the rapid development of science and technology, many sensors, such as temperature sensors, image sensors, and magnetic sensors, have been widely applied in a variety of fields such as auto-mobile industry and the telecommunications. Transducers are devices which function generally to translate an input of one form into an output of another form or magnitude.

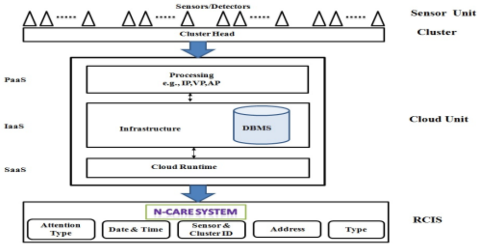


Figure 4. Overview of System Design

Algorithm 7 DataCleaner

```
Obtain the tuples that have to be transformed.
connection ← getConnection(mytable)
for t ∈records.getRecords() do
    query ←"DELETE FROM dbname WHERE record = t"
end for
```

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

A transducer usually comprises of a diaphragm which moves or vibrates in response to a some form of energy, such as sound. Loudspeakers, thermometers, microphones, position and pressure sensors are some examples of transducers with diaphragms.

The Sensor Unit provides input for NCS. It is a heterogeneous network of nodes. It contains sensors that vary tremendously in utility and application. Here we have used sensors, real and simulated, for fields like room care, agriculture, medical field.

Clusters: A cluster is a combination of a base station connected to a computing device, that is capable of connecting to the internet and wireless sensor nodes, that form an ad-hoc network with the base station. Every cluster has a local database that stores data sent by the sensors to the base station.

Cluster daemons that run on the supporting computing device to which the base station is connected, continuously uploads the data from the local database to the cloud database. It also performs clean up of the data present in the local database. Local clusters have considered limited memory, and in order to conserve space, old entries in the table are deleted.

The cluster daemon implemented in this paper has the ability to manage clusters, optionally transform data and provide reporting tools on the uploaded data. The cluster daemon is divided into the following parts

1. Extractor (Cluster Daemon)
2. Transformer
3. Reporting

1. Extractor

Extractor or the cluster daemon is used to retrieve the data from a cluster node and send it to the master database and do a cleanup. The cluster daemon written in java has the ability to access any type of database using the descriptor or metadata of the sensor database.

Architecturally the extractor is divided into three parts.

Algorithm 8 DataSender

```
Obtain the tuples that have to be transformed.
connection ← getConnection(mytable)
for t ϵrecords.getRecords() do
    rtp ←TransformedTuple(cdb, tbd)
    for i = 0 →t.value.length do
        if t.type[i]equalsTuple.TIMESTAMP then
            rtp.value[Tuple.TIMERECORDED] ←t.value[i]
        else if t.type[i]equalsTuple.INTEGER then
            rtp.value[Tuple.VALUE] ←t.value[i]
        else
            if t.sensorid[i]equalstrue then
                rtp.value[Tuple.SENSORID]
                    ←t.value[i].toString();
            end if
        end if
    end for
    Send the record t to the cloud.
end for
```

(a) Data xtractor

(b) Data Sender

(c) Data Cleanup

(a) Data Extractor

The data extractor is part of the cluster daemon that processes or extracts information from the cluster node about the information about the sensors. The following is the syntax for the descriptor used by the cluster daemon.

```
<table name=”test” dataset=”fire ”
        connectionstr=”localhost:3306 ”
username=”root” password=”root”
interval=”5”>
<field name = ”col1” type=”int”
min=”10” max=”15” ></field>
<field name = ”col2” type=”string”>
</field>
<field name = ”col3” type=”long”
min=”−1000” max=”1500” >
</field></table>
<table name=”test2” dataset=”temp”
dbname=”mydb” connectionstr
=”localhost:3306” username=”root”password=
”root”>
<field name = ”tb1col1” type=”int”
min=”11” max=”16” ></field>
<field name = ”tb1col2”sensor=”true” type
=”string” ></field>
<field name = ”tb1col3” type=”long”
min=”−2000” max=”2500” ></field>
</table>
```

The above XML has a table tag, which is used to extract the data from the local or the cluster node table. The attributes are described below.

- Name: name of the sensor. This is the name that will be linked to the sensor that will be used to reverse track.
- Dataset: to which dataset the information extracted is to be placed. A dataset is used for multiple reasons, the critical being able to partially view the information on the cloud. It is used for security reasons.
- Dbname: the name of the database the sensor writes the information into.
- ConnectionStr: a connection string is a string that is used to connect to a database with the help of a JDBC connector. There are a variety of drivers written by the database makers to help connect to the database.
- Username: the name of the user that can access the information or the database mentioned above. It is important that the user has read and write permissions.
- Password: password for the user mentioned above. This field can be omitted if no password is needed.

The field tag is described in detail below

- Name: name of the column by which the sensor stores the value into.
- Type: the type of the data the sensor stores, it could vary from string, integer to timestamp.
- Min: this is the minimum value of the field that is used by the transformer to transform the data.
- Max: this is the maximum value of the field that is used by the transformer to transform the data.

A cluster daemon can process more than one database and can manage multiple fields. The definition of individual fields are done as shown, only the fields with the min and max are transformed.

The fields that are mentioned are transferred and queried upon.

b) Data Sender

This module is used to send the contents of the cluster daemon to the main database. The details of the database are configured into the configuration manager, which is illustrated below.

```
<client id=”1” dbname=”test”
connectionstr=”localhost:3306”
username=”root” password=”root”>
</client>
```

The attributes of the client is explained below

- dbname: name of the master database.
- connectionStr: connection string used to connect to database
- Username: name of the user.
- Password: password for logging into the db.

The cluster daemon is configured to write only into one master database. This is not a drawback as centralization should be done from the cluster there by reducing the efforts done be data-sender to maintain redundancy issues.

(c) Data Cleaning

This module involves clearing data from the cluster node. It makes sure that the data in the cluster is deleted and moved to master database and no redundant data exists. This way the cluster is cleaner and easier to maintain and most of the heavy lifting is done on the master database.

As for security reasons keeping the data collected in multiple places is good if you want backup but bad if you want the data to be secure. There is a lot that goes into security, which is expensive both in terms of money and human resource. The better system than storing or keeping the old values of the cluster is to backup the main database. This way the data is not lost forever and also secure.

2. Transformer

This module helps in transforming the data extracted by the extractor and integrates it. The approach used in this case is to create a database with a schema.

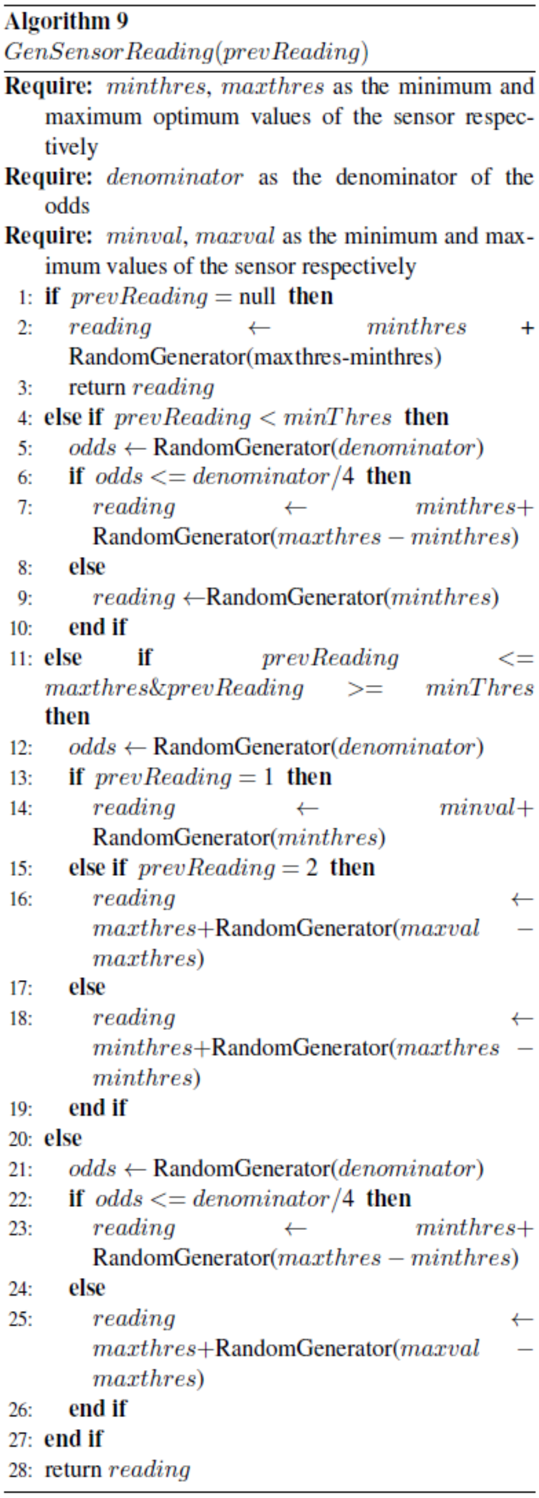
This design ensures any type of sensor and any type of data is handled and transformed.

The transformation is done with the help of metadata present in the descriptor, which has the min and max thresholds, the data is scaled down to 1 to
100 and they are stored in the cloud database.

This way the data in the master table is same for different sensors. This way data mining algorithm can run or process the data easily. Data present in the master database can be shared across to different personal and departments. This way true centralization of the data can be achieved heterogeneous sensors. Transformation is necessary because the data reported by the sensors can be in different scales and they themselves can be in different locations. The transformation of data will help us understand the data in a common scale.

Monitoring organizations can make a better use of the centralized data. This data can be fetched and reported to the necessary departments. Since all the operations are carried out real-time the actions taken can be on time and more efficient in emergency cases.

The process of monitoring will change drastically with this framework, since all the data is put on the cloud and are put under a sunset called dataset. It will be a lot easier and more scalable than the traditional approach which involved continuous polling of the database. SaaS cloud phenomenon has made it possible to use software like MySQL on cloud making it less architecture and network dependent and more scalable and efficient.



```
Algorithm 9
GenSensorReading(prevReading)
Require: minthres, maxthres as the minimum and maximum optimum values of the sensor respectively
Require: denominator as the denominator of the odds
Require: minval, maxval as the minimum and maximum values of the sensor respectively
if $prevReading = null$ then
    $reading \leftarrow minthres$ + RandomGenerator(maxthres-minthres)
    return reading
else if $prevReading < minThres$ then
    $odds \leftarrow$ RandomGenerator($denominator$)
    if $odds <= denominator/4$ then
        $reading \leftarrow minthres+$RandomGenerator($maxthres - minthres$)
    else
        $reading \leftarrow$RandomGenerator($minthres$)
    end if
else if $prevReading <= maxthres\&prevReading >= minThres$ then
    $odds \leftarrow$ RandomGenerator($denominator$)
    if $prevReading = 1$ then
        $reading \leftarrow minval+$RandomGenerator($minthres$)
    else if $prevReading = 2$ then
        $reading \leftarrow maxthres+$RandomGenerator($maxval - maxthres$)
    else
        $reading \leftarrow minthres+$RandomGenerator($maxthres - minthres$)
    end if
else
    $odds \leftarrow$ RandomGenerator($denominator$)
    if $odds <= denominator/4$ then
        $reading \leftarrow minthres+$RandomGenerator($maxthres - minthres$)
    else
        $reading \leftarrow maxthres+$RandomGenerator($maxval - maxthres$)
    end if
end if
return reading
```

3. Reporting

Ideally reporting is the most critical functionality of this system. This tools or framework may be used by a number of organizations and their rights to view information may be different. It is important that these things are handled as part of the framework.

The dataset attribute is used to differentiate the different type of information that is coming into the system. In some cases it becomes important to reconstruct the actual values of the sensors rather than the transformed values. These are addressed with the help of a web app adhering to this framework that follows a certain authentication policy to prevent security risks. The use of graphing tools makes it very interactive yet informative reporting.

Cloud Computing System: Cloud computing is the delivery of computing as a service. Resources such as hardware, software and information are shared over a network. In a cloud computing system, there's a significant shift in the workload. Local computers no longer have to do all the heavy lifting when it comes to running applications. The network of computers that make up the cloud handles them instead. Hardware and software requirements on the user's side decrease. A user's computer just needs to be able to run the cloud computing systems interface software, such as a Web browser, and the cloud computing system takes care of the rest.

In NCS, the Cloud complements the sensors, which have limited resources by providing additional computational power, resources and storage capacity. For example, virtualization on the cloud provides hardware for activities such as Image Processing (IP) and Video Processing (VP). The cloud also provides infrastructure services such as a Database Management System (DBMS) for storing and managing the data obtained from the sensors. Users can deploy their software on the cloud, using the Software as a Service (SaaS) feature. The values obtained from the sensors or after processing it are normalised on a scale of 1-100 using SaaS. Each type of sensor will have different software associated with it for performing the normalisation.

In NCS, both the end users and the clusters are clients of the cloud.

End User Client: A registered end user can log into the system using any web browser. An end user can access his or her domain specific data. The end user UI displays a variety about the different Middleware Services NCS requires two kinds of middleware services, one between the WSN and the cluster daemon and the other between the cluster daemon and the end user. The middleware service between WSN and cluster daemon offers format independence. It enables the cluster daemon to manage sensors irrespective of the specification, functionality, data format and make of the sensor. The cloud middleware offers services such as computation, processing, normalization, scalable database management system, additional storage capacity and virtualization.

VIII. IMPLEMENTATION

This section gives the overview of implementation of NCS. This work has been carried out using Java as the programming language and JSP for creating the dynamic web pages and to display crucial information. Google SQL has been used as the cloud based DBMS. A combination of MySQL and Postgre databases were used as cluster DBMS.The Google App Engine was used to deploy the SaaS components.

7.1 Implementation Details

7.1.1 Wireless Sensor Network

For the experiment we have used a combination of actual sensors and sensors simulated in Java. The description of the real sensors, data acquisition board and base station is given below.

- IRIS It is a wireless measurement system developed by MEMSIC. The IRIS is a 2.4 GHz Mote module used for enabling low-power, wireless sensor networks. It uses a IEEE 802.15.4 compliant RF transceiver. A variety of sensor and data acquisition boards can be mounted on the IRIS Mote via the standard 51 pin expansion connector.
- MDA300CA The MDA300CA is an versatile data acquisition board that also includes an on board temperature/humidity sensor. With its multifunction direct user interface, the MDA300CA can be used for environmental and habitat monitoring as well as many other custom sensing applications.
- MIB520CB The MIB520CB provides USB connectivity to the IRIS and MICA family of Motes for communication and in-system programming. Any IRIS node can function as a base station when connected with the MIB520CB USB interface board. A base station obtains details from all the nodes that are part of the network. In addition to data transfer, the MIB520CB also provides a USB programming interface.The MIB520CB offers two separate ports: one dedicated to in-system Mote programming and a second for data communication over USB. The MIB520CB has an onboard processor that programs Mote Processor Radio Boards. USB Bus power eliminates the need for an external power source. For the sensors to work, they motes need to be programmed. The motes are programmed by mounting the MIB520CB USB connector. The MIB520CB utilises two ports, the lower port for programming and the higher port is for data transfer.

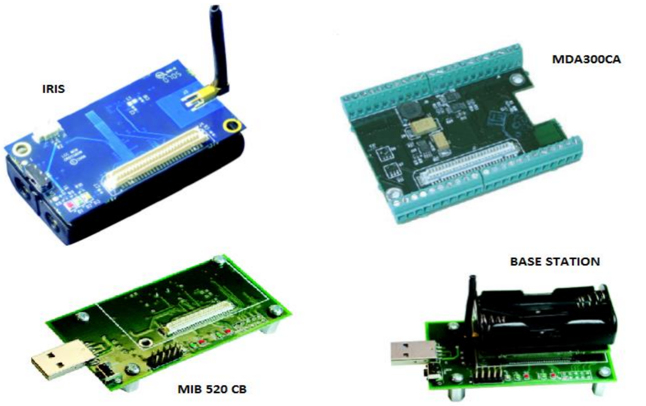


Figure 5. Sensors, Data Acquisition Board and Base Station used for the experiment

The Crossbow Moteview 2.0F application is used for burning the program in to the moteview and for choosing other customization. In this paper we have used Xmesh Low Power protocol for the sensors. The XMesh protocol stack is a powerful open-architecture, flexible embedded wireless networking and control platform built on top of the TinyOS operating system. XMesh combines high-performance with interoperability through the support of open standards including IEEE 802.15.4/ZigBee.

Xmesh is a low-power, multi-hop routing protocol that offers a solution to routing by creating self healing, ad-hoc groups of motes that tunnel sensor data to a base node. Xmesh packets contain the address of the origination node in addition to the sender and destination.

This protocol enables messages to hope from node to node in a network rather than rely on a point to point connection that is susceptible to rapid changes. In case a mote malfunctions, the other motes automatically reroute around the damaged link ensuring that data flow is not interrupted.

The node ids are assigned to the motes during the burning process. The mote that will serve as the base station the node id is 0. Once all the motes are programmed, they are ready to be deployed. The base node is mounted on MIB520CB and connected to the cluster head. Moteview application uses a Postgre database to store the values from the sensors. The values are stored in xbw da100 results in task database. Since the number we had were limited, in order to demonstrate the scalability of the sensors we simulated the remaining sensors in Java. Though these sensors are virtual in nature, they randomly generate values similar to that of the real sensors and their values are written into a database. In order to account for heterogeneity of the sensors, we have considered three broad categories of sensors. The MEMSIC sensors are classified as room care sensors and are updated real time. Agrocare(sensors that measure soil pH, mositure and temperature) and medcare (sensors that measure patient's body temperature, blood pressure and pulse) are the other two types of sensors. The classes under the agroSense package simulate the behaviour of various agriculture related sensors while classes under medCare package simulate the behaviour of sensors used in hospitals. The roomcare sensor values are stored in a Postgre database while roomcare and medcare sensor values are stored in to MySQL databases. The Random Generator provided by Java is used for randomly generating values. However the problem here is that the random generator tends to produce values that are too erratic. For example, the blood pressure of a human may not always be erratic. Seldom does the bp of the person jump from 60 to 150, fall to 150 to 45 and jump to 225 in five seconds. However using the random generator can produce such readings. As a result the programs have been written in such a way that the values generated are not generated. Once a reading is generated, it is compared with the previous reading. If the new reading is deviates too much, it is re-generated. However sometimes, the condition of a person can deteriorate all of a sudden. This is again decided based on another random number.

7.1.2 Clusters

The cluster daemon has been implemented using Java. It is divided into three logical blocks. These blocks have different kind of operations.

1. Client
2. Server
3. DAO
4. Parser

1. Client The client package takes care of the processing of the information on the cluster daemon. This involves connecting to the local database and extracting the information. The object used for storing is DAO.

2. Connector: This module helps the client to create JDBC connection. JDBC is a java wrapper that makes connection to databases operating system independent. But it is important that the driver for the database should be added to DriverManager a java library. This is done with the help of class.ForName which loads the supported libraries.

Since the client should support multiple databases it becomes important that this class covers most of the traditional databases.

3. Query Manager: The query manager is the process of managing the SQL queries on the databases. There are a number of operations that could be performed here including data cleaning etc. In the implementation the fetchRecords is implemented which fetches the details from the database of a table field specified in the configuration.

Delete records is used to delete all the records for this iteration. i.e. once the information is processed completely.

4. Transformer: The client or the cluster if has enough computational capability then transformation can be done with the help of the Transformation module. This class is designed to normalize the information. Normalization is an important task as this representation of the data can be used in data mining and machine learning mechanisms or process. This module takes in the data type a Tuple and returns TransformedTuple.

5. Cleaner: This module helps clean the database; with the help of query manager each of the transformedTuple is deleted once it has been written to the cloud.

6. ClientRetriever: This module combines all of the above functions of a client into a thread. A thread in java is an independent execution unit. This thread is created for each table tag and the information is sent to the data sender module.

7. Server: The server packages handle the data that has to be sent to the server and also convert the local schema to the global schema. The server thread isnt run in parallel with the client package but created by it. Server package has to make a call to the data cleaner else the information remains on the server

8. ClientSender: This is the module that is responsible to convert the local sensor schema that may be different for individual sensor to global schema. There are other information that is sent along with this sensor data like min-threshold and maximumthreshold etc.

9. QueryManager: This class is used to handle the queries on the cloud. The function implemented is to send an array of transformed tuples. Any errors in writing should be communicated with the data cleaner module so that those records arent erased.

10. Connector: This connector has to maintain the connection to the cloud. It becomes important that the cloud providers support JDBC connection. The connection driver provided by the GoogleCloudSql works on the internet too.

11. Dao: The client daemon has to handle large amount of data and it should do so quickly. Dao uses ClientDBConfig, TableInfo, Tuple, FieldInfo and TransformedTuple data-structures to store information and communicate.

12. ClientDBConfig: This class is used to describe the cluster daemon. It is created from the configuration. It contains the connection information of the sensors and also the cloud along with the user credentials and also the details of the number of sensors connected along with the number of fields and some meta data about the database they are connected to.

13. TableInfo: This class contains the details of the individual sensors connected to the cluster daemon; this also contains the credentials to connect to the local database along with the information about the interval at which the data must be fetched from the sensors table.

It also contains the details about the dataset to which the sensor belongs to; a dataset is a field that we use to find out to whom the information is available to.

14. FieldInfo: This class contains the details about the individual fields that a sensor. Contains the information about the column names in the database, the type of the data and also if the field represents the sensor id.

15. Tuple: This data structure contains a single record of a sensor. It is designed to hold any type of value and also contains the column names, types, if the field is a sensor-id and the value of the sensed entity.

16. TransformedTuple: This is an object that is constructed from a tuple. It is very similar except that the tuple semantics are changed to match with the cloud schema. It contains information about which column contains max threshold, min threshold, sensor-id, timestamp and other Meta information.

17. PropertyParser: The property parser is an interface to read the configuration file. This configuration file can be of variety of exchange documents formats varying from JSON which is a javascript object notation used to represent java script objects in string format and XML commonly known as the markup language.

18. PropParserXML: This is class that implements the PropertyParser interface and serves the ClientDBConfig object from an XML document. The format for which have been covered before. It is critical that the markup standards are followed.the following is the DTD the parser is implemented for

```
<!ELEMENT client ( table+ ) >
<!ATTLIST client clouddbname NMTOKEN
#REQUIRED >
<!ATTLIST client connectionstr CDATA
#REQUIRED >
<!ATTLIST client id NMTOKEN
#REQUIRED >
<!ATTLIST client password NMTOKEN
#REQUIRED >
<!ATTLIST client username NMTOKEN
#REQUIRED >
<!ELEMENT field EMPTY >
<!ATTLIST field max NMTOKEN
#IMPLIED >
<!ATTLIST field min NMTOKEN
#IMPLIED >
<!ATTLIST field name NMTOKEN
#REQUIRED >
<!ATTLIST field sensor NMTOKEN
#IMPLIED >
<!ATTLIST field type NMTOKEN
#REQUIRED >
<!ELEMENT table ( field+ ) >
<!ATTLIST table connectionstr CDATA
#REQUIRED >
<!ATTLIST table dataset NMTOKEN
#REQUIRED >
<!ATTLIST table dbname NMTOKEN
#REQUIRED >
<!ATTLIST table interval NMTOKEN
#REQUIRED >
<!ATTLIST table name NMTOKEN
#REQUIRED >
<!ATTLIST table password NMTOKEN
#REQUIRED >
<!ATTLIST table username NMTOKEN
#REQUIRED >
```

7.1.3 Cloud Environment

Google App Engine has been used as the cloud environment. Google App Engine) is a platform as a service (PaaS) cloud computing platform for developing and hosting web applications in Google-managed data centers. Applications are sandboxed and run across multiple servers. App Engine offers automatic scaling for web applications as the number of requests increases

for an application, App Engine automatically allocates more resources for the web application to handle the additional demand. Google App Engine supports only Java, Python and Go programming languages. We have used Java.

The Google App Engine based Google SQL has been used as the cloud based database management system. The Google SQL is basically a platform as a service. Google Cloud SQL is web service based client that allows creation, configuration, and use of relational databases with App Engine applications. It offers the capabilities of a MySQL database, and facilitates easy movement of data, applications, and services in and out of the cloud. For this experiment we have used a D0 instance type of Google Cloud SQL service. It includes one Google Cloud SQL instance with limited amount of RAM (¡512 MB) and 0.5 GB storage space. To effectively utilise the limited resources available with this instance, only two tables have been maintained on the cloud. The ncs stores all the readings from all the sensors and login to authenticate end users. JDBC connections have been used to connect to the cloud database and retrieve data. This is done by including the Google Cloud SQL API to the build path of the project and authenticating this tool by allowing to access the instance.

The values of each sensor at a given instant of time is sent to the Google SQL cloud service. A database named ncs with application id ncaresystem has been used for this purpose. Each time a cluster daemon sends a new batch of sensor values, it is added to the table ncs as tuples. In this way the entire history of the sensor readings are maintained.

The classes under the NCS project represents cloud middleware. There are different classes for handling different type of sensors. For example, AnalyseBPSensor class represents the software needed to processor values from a particular blood pressure sensor. The NCS has been been deployed on the Google App Engine.

Each class under the project continuously keeps scanning new tuples, whose type under its domain. The real values are normalized/processed and the corresponding tuples are updated with their normalized values.

7.1.4 End User

The end user client page has been designed using JSP, Html, CSS and Javascript. Each user has to login with a user name and password.Based on the user name, the end user will be directed to his or her home page, which shows the status of the various sensors assigned to him. For example, a doctor will be able see the status of all medcare sensors that are present in a particular hospital.

7.2 Experiment Details and Results

For demonstrating the advantages of using a cloud based framework, two experiments have been performed, the first with a traditional DBMS and second with a cloud based master database. The total response time was calculated in both cases using the formula illustrated below:

Total Response ime $= \alpha + \beta + \gamma + \theta$

where,

α is time taken to transmit from sensor to cluster,β is time taken by cluster daemon to read from the cluster database and send to cloud database,

γ is time taken by the SaaS componentθ is time taken by UI to read from the cloud databaseIn the first experiment, a traditional MySQL database was used as the master database. Also the normalization was done locally on the system in which the master database was located. The experiment set up included a set of simulated sensors, cluster daemons, SaaS component and a master database. This set up represents the traditional client server model, where no cloud services are used. In this experiment test cases with 1 to 4 clusters were . Each cluster had 100 simulated sensors each. In order to support cluster daemon a combination of 2 host systems and 2 virtual machines were used.

Fig 7 shows a graph of response time vs number of clusters. It can be seen that there a large increase in response time with increase in the number of clusters. This is because this set up does not lacks the ability to scale with rapid increase in demand.

In the second experiment, a traditional MySQL database was replaced by a Google Cloud SQL database. The SaaS component was deployed on the Google App Engine. The experiment set up included a set of simulated sensors, cluster daemons, cloud based SaaS components and a cloud based master database. This set up represents the NCS model, where cloud services are utilized the robustness and scalability of the framework. In this experiment too, test cases with 1 to

4 clusters were . Each cluster had 100 simulated sensors each. In order to support cluster daemon a combination of 2 host systems and 2 virtual machines were used.

Fig 8 shows a graph of response time vs number of clusters. It can be seen that there a small increase in response time when the number of clusters is increased from 1 to 2. When the number of clusters is increased to 3, there is a fall in the response time. Again there is a small jump in the response time when number of clusters used is 4. The overall change in response time is minimal when this framework is used.

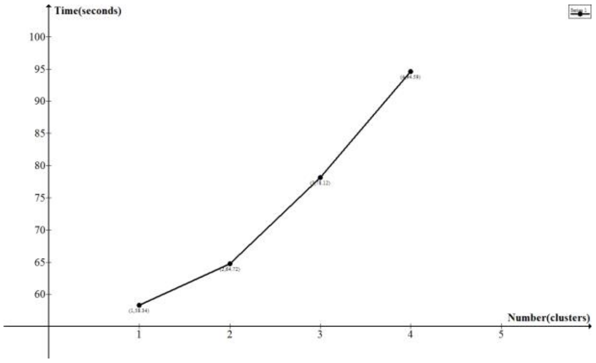


Figure 7. Total Response Time for framework without cloud support

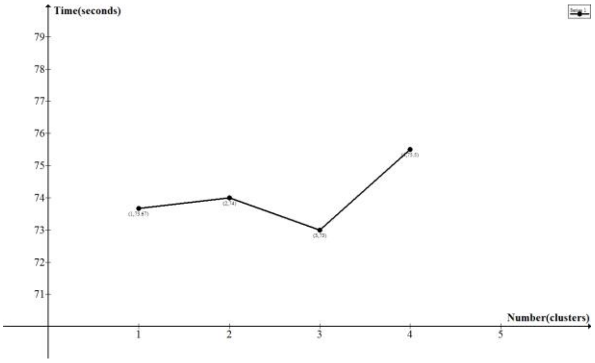


Figure 8. Total Response Time for NCS (cloud support enabled)

From Fig 7 it can be noted that without using cloud services the total change in response time as the number of clusters increased from 1 to 4 is around 36 seconds or 62%. In contrast it is evident from Fig 8 that the total change in response time as the number of clusters from 1 to 4 is roughly only 2 seconds or 3.4%. This is very minimal compared to the traditional model. This can be attributed to the scalability provided by the cloud services. The main advantage of using the cloud, apart from massive availability of resources is the fact it can scale effectively to meet increase in load.

7.2.1 Statistics from Google APIs Console

Fig 10 shows the number of queries made to the database on the cloud per minute. This graph is snapshot of a part of the total duration the experiment was performed (4 minutes). At the peak, an average of 348 queries were made to the database per minute. It is inclusive of the queries by both the end user clients and cluster daemons. The peak was obtained when all four cluster were executing in parallel.

Fig 11 shows a snap shot over 15 minutes and variations in the number of writes made to the database. The writes to the database involve inserting values into the database. This was done by the cluster daemons. At the peak, when four clusters were executing, 360 writes were made to the database, by the 4 clusters in parallel.

Fig 12 shows the number of open connections to the database. At the peak (when all 4 clusters were executing in parallel) there 600 open connections. The clusters used in the database were a combination of clusters with critical sensors, that need real time updating and non critical sensors that send by values only at periodic intervals. The number of open connections is almost the double of the number reads. This is because the cluster daemons for non critical sensors do connect to the cloud database but since sensors values dont send values frequently, they write data only when the value from the sensors change.

Fig 13 shows the number of reads and writes made to the database. Cluster daemons write to the database while the end user client's console reads from the database. The number of reads is always constant since the clients are configured to read from the database at regular intervals of time. The number of sensors or clusters do not make a difference. However, the number of writes vary according to the number of sensors/clusters that are active As you can see, the graph for number of reads and number of reads and writes are identical, except that the number of read and write is is constantly

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)
© Copyright: All rights reserved.

20 more than number of writes . The graph form is identical. 20 reads are made by the clients per minute.

IX. USE CASES

NCS is a live system that collects different data for different end users. The end users login to their accounts with their user name and password. The end user is then shown the status of all the sensors assigned to him or her. The end user can view details such as sensor id, cluster id, type of the sensor, address where it is located and the attention type of the event. The end user can take appropriate action according to the attention type.

A sensor node keeps a track of all the events that occur around it by measuring a physical quantity. Each sensor is designed to measure a different physical quantity using suitable unit of measurement. The readings of each sensor are sent to a cluster head in the form of electric signals which are stored in a local database after some basic processing. The cluster daemon running on the cluster head sends the data to a master table on the cloud. The sensor information is analysed by specialised software for each type of sensors in the cloud.

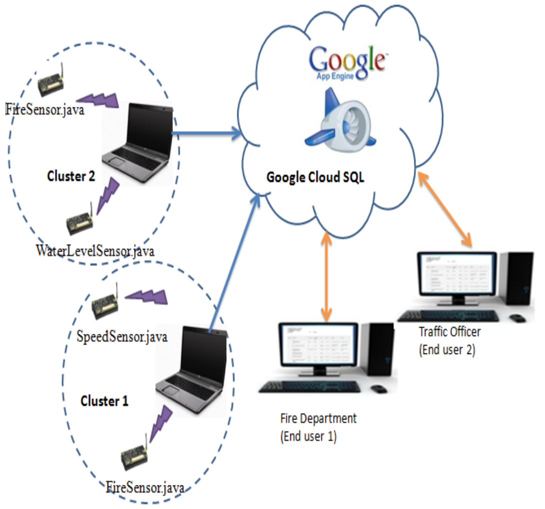


Fig 9: Experiment SetUp.

In order to maintain uniformity and for ease of calculation, each software then calculates the severity rate in a scale of 0-100. Depending on the severity rate, it assigns the attention type for each event. This information is displayed by the user interface of NCS. The user can interact with NCS using the user interface, keep tabs on what is happening by looking at the present status and also view the entire history of that particular sensor. Using this information provided by NCS, the user can closely monitor a situation and effective action can be taken in case of any emergency. Depending on the priority of the sensor (real time or moderate or low priority), the information from each sensor node is updated after regular intervals of time and any change in attention type of any of the sensor nodes can be observed by the end user. For real time sensors the interval will be almost zero.

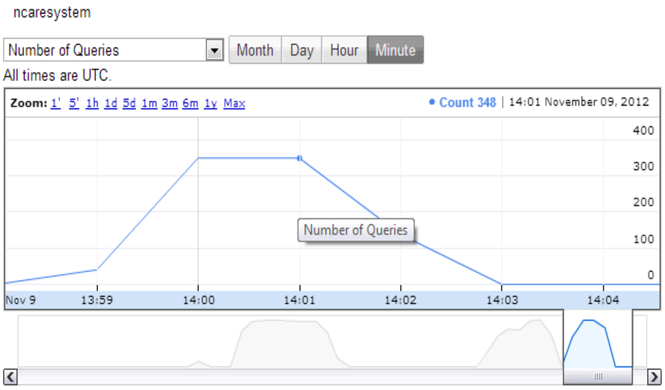


Figure 10. Number of Queries

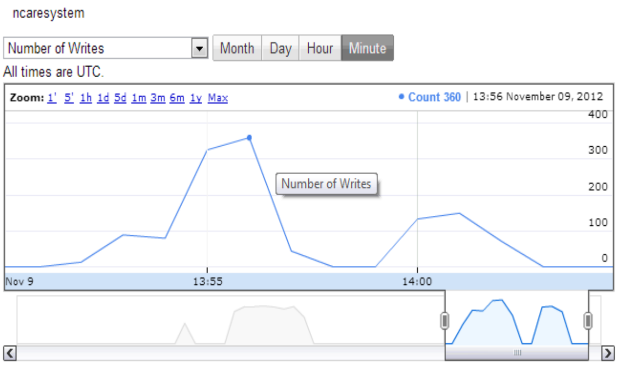


Figure 11. Number of Writes

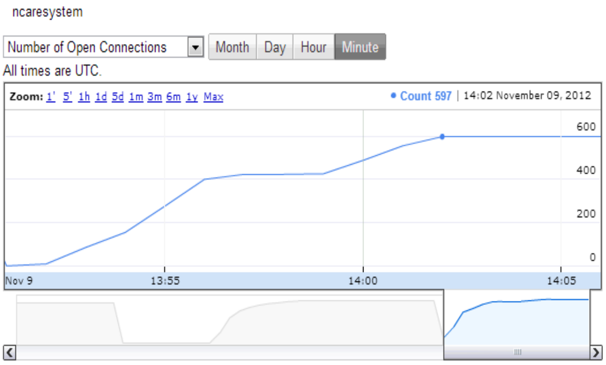


Figure 12. Number of Open Connections

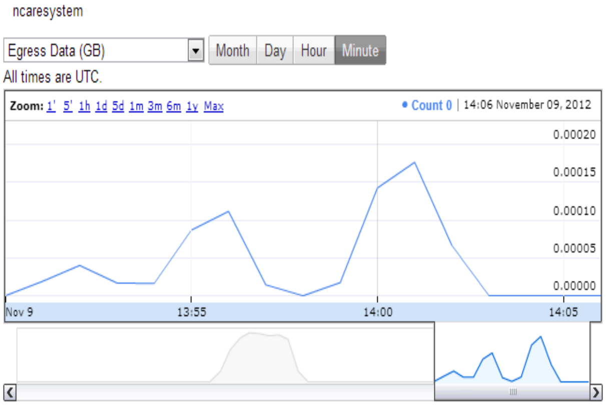


Figure 13. Number of Reads and Writes

In order to maintain uniformity and for ease of calculation, each software then calculates the severity rate in a scale of 0-100. Depending on the severity rate, it assigns the attention type for each event.

This information is displayed by the user interface of NCS. The user can interact with NCS using the user interface, keep tabs on what is happening by looking at the present status and also view the entire history of that particular sensor. Using this information provided by NCS, the user can closely monitor a situation and effective action can be taken in case of any emergency. Depending on the priority of the sensor (real time or moderate or low priority), the information from each sensor node is updated after regular intervals of time and any change in attention type of any of the sensor nodes can be observed by the end user. For real time sensors the interval will be almost zero.

8.1 Medical Care Information

Consider a doctor A who works at hospital X. Hospital X has two kinds of patients: those are presently admitted there and those who are under remote supervision. Doctor X can log in to the NCS client using the username and password provided by his hospital. He can then view the status of all the patients who are assigned to him, whether they are presently admitted or are under remote supervision. Each patient is equipped with different types of sensors for measuring parameters such as body temperation, pulse and blood pressure. The sensors regularly transmit back their reading to a cluster head, which could be a computer at home or some other internet capable device in the hospital. Cluster daemon present in these devices push the data into the cloud database. The doctor's UI then reads from the UI and displays the latest available readings from the sensors. Now for instance, suppose a patient's blood pressure suddenly falls below the threshold value, the event becomes high priority. It is assigned a very high attention type by the SaaS component. The event is shown as having very high attention type. Seeing this, the doctor can rush to a patient's help.

8.2 House Care Information

Consider a house owner B, who lives in a smart house. The house has different types of sensors like room temperature sensors, electricity meter sensor, smoke detectors among other. B's elderly mother C, who is diabetic also lives in the same house. Her health status is monitored by Dr. A of hospital X. In case of a medical emergency both doctor A and B are informed by NCS. C has a blood sugar sensor strapped on. The blood sugar sensor along with the other sensors in the house continuously transmit data back to the central desktop. The cluster daemon then pushes the data into the cloud database. On logging into his account, B can view the status of all the sensors back home and also supervise his mother's health. Doctor A can supervise C's health condition.

X.NCS APPLICATIONS, ADVANTAGES AND DISADVANTAGES

10.1 NCS Applications:

1. A Smart grid is an application of NCS. A smart grid is an electrical grid that uses WSNs to gather and act on information like information about the behaviour of suppliers and consumers automatically to improve the efficiency, reliability and sustainability of the production and distribution of electricity.

2. NCS framework can be applied to develop Smart Hospitals that use WSN to supervise patients health round the clock. Additionally doctors can remotely track the health conditions of the patients from anywhere in the world

3. Smart homes can utilize the NCS frameworks. Homes equipped with heterogeneous wireless sensor nodes can send back data to a different end users. Electric meter sensors will send data to the electric board officials and home care sensors will send data to the house owners

4. The NCS concept can be extended to create smart cities with a distributed network of intelligent sensor nodes which can measure a variety of parameters for efficient management of cities. The data from various sensors can be delivered wirelessly to the concerned authorities and citizens.

10.2 NCS Advantages:

1. Simplicity: NCS is designed to be easy understand and use.

2. Heterogeneity : NCS can be to integrate heterogeneous sensor nodes with realtive ease.

3. Semantic Independence and Robustness: The cluster daemon uses XML to represent the configuration of a particular sensor type. New sensors can effectively be integrated by creating new configuration files

4. Real Time System: NCS gives real time data for real time sensors and can be used by end users can track live events.

5. On demand deployment or Scalability: NCS has provisions for scaling up or down infrastructure based on the need. Additional sensors and hardware can be enabled in times of emergency.

6. Virtualization : For additional computing and sensing power, NCS utilizes Virtual Sensors [18].

7. User Friendly: The NCS data is organised in an user-friendly manner.

9.3 NCS Disadvantages:

1. Lack of accuracy: Sometimes Attention Type may not be 100% accurate.

2. Sensors are prone to failure: Sensors may run out of battery or may wear out due to prolonged usage

3. Performance heavily linked to internet connectivity: The NCS framework is heavily dependent on the internet connectivity and network speed. Response time can be high if the network is too congested or internet speed is slow.

4. Security Issues: A proper security model for cloud computing is not yet developed. For public cloud services, security and privacy is difficult to implement.

5. High Cost: Deployment would be expensive. In case private cloud is used, investment in infrastructure would be needed.

6. Difficultly in migration: Presently it is very difficult to migrate from one cloud service provider to another.

XI. FUTURE WORKS

Sensors will play a key role in our future and it will become more and more important that we use the framework to integrate sensor data. The one thing that we could do to make this framework better is to use a probability model to find out the columns required to be sent to the server without them being manually specified in a descriptor. This concept would involve a lot of automation and make this framework easy to use. Let us take this to consideration; with the help of the metadata present for every table we could obtain column names of every table to which the data is sent. This along with the probability distribution and some machine learning concepts could make it possible to see which row has repeating data and which row is too trivial to be sent to the cloud.

Another improvement that can be done is to provide some sort of control over these cluster daemons to the administrators. This control would help in maintain a large sensor networks where handling the sensors becomes a real issue. A number of side parameters can also be sent about the health of these sensors to their geographical location which could prove useful.

XII. TECHNICAL DIFFICULTIES FACED

1. Availability of real sensors: it is was close to impossible to obtain enough number of sensors to perform experiments. Therefore we simulated the sensors as computer programs, care was taken that the numbers generated behaved close to the real sensors.
2. Availability of clusters: Since the test was performed on mixed set of sensors(few real and a few simulated) it proved the true support to heterogeneity it was also challenging enough to obtain hosts, hence we used virtual machines as hosts for a few sensors.
3. Providing support to heterogeneous sensors: Most of the sensor makers wrote the data in the databases as they are redundant and safer and easier to maintain, but this comes with the challenge that the sensors have individual schema or representation of data amd even more that each sensor of same type had different grade of measurement. it was important to store the information in a standard format and conversion of individual schemas of the sensor to one standard schema was a challenge.
4. Cloud SaaS: The Google cloud engine though very powerful has limited expandability from low usage to high. As this was not the inefficiency of the system but the service providers, some of the cloud SaaS providers now give this feature and may vary the experiment results

XIII. CONCLUSION

The N Care System offers many advantages. The use of cloud infrastructure increases the computational power of the system. In such a system, computation is done using the cloud infrastructure rather than by individual sensor nodes. As a result the power requirements and size of each sensor can be reduced. Smaller sensors are easier to sustain in times of an emergency such as a natural calamity and to conceal for detecting crime. Additionally, NCS offers a high degree of scalability. As a result it can handle increase in number of sensor nodes without much performance overheads. Since the system is dynamic, back-up sensors can be enabled, in case the main sensors fail. NCS can be utilized to collect data from different types of heterogeneous sensors and to provide domain specific sensor data to the end users.

Future works would revolve around using the cloud framework to improve the battery life of the sensors.

REFERENCES

1. R. Bloor. What is a cloud database. Technical report.
2. S. Bose and R. Liu. Cloud computing complements wireless sensor networks to connect the physical world. Technical report.
3. Chandrakant N, Bijil A P, Deepa Shenoy P, Venugopal K R, and L M Patnaik. Middleware service oriented rescue and crime information system (rcis) using heterogeneous fixed nodes in wsns. In ADCONS 2011, December 16-18, 2011, Karnataka, India.
4. Chandrakant N, Bijil A P, Deepa Shenoy P, Venugopal K R, and L M Patnaik. Middleware service oriented rescue and crime information on cloud (rcic) using heterogeneous nodes in wsns. In ADCONS 2011, December 16-18, 2011, Karnataka, India, pages 1–5, 2012.
5. I. Giurgiu, O. Riva, D. Juric, I. Krivulev, and G. Alonso. Calling the cloud: Enabling mobile phones as interfaces to cloud applications. In Proceedings of the 10th International Middleware Conference Middleware’09), November 30 December 4, 2009.
6. D. Huang, X. Zhang, M. Kang, and J. Luo. Mobicloud: Building secure cloud framework for mobile computing and communication. In Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium, pages 27 – 34, June 2010.
7. Hung-Chin Jang, Yao-Nan Lien, and Tzu-Chieh Tsai. Rescue information system for earthquake disasters based on manet emergency communication platform. In IWCMC09, June 21 24, 2009, Leipzig, Germany.
8. G. Kaefar. Cloud computing architecture.
9. A. Khan and K. Ahirwar. Mobile cloud computing as a future of mobile multimedia database. In International Journal of Computer Science and Communication.
10. D. Kovachev, Y. Cao, and R. Klamma. Mobile cloud computing: A comparison of application models. In Service Oriented System Engineering (SOSE), 2010 Fifth IEEE International Symposium.
11. E. E. Marinelli. Hyrax: Cloud computing on mobile devices using mapreduce. September 2009.
12. A. P. Miettinen and J. K. Nurminen. Energy efficiency of mobile clients in cloud computing.
13. C. S., G. Kumar, M. K. K. Dinesh, and A. M.A. Cloud computing for mobile world.
14. Q. A. Wang. Mobile cloud computing.
15. Xinwen Zhang, Joshua Schiffman, S. Gibbs, Anugeetha Kunjithapatham, and Sangoh Jeong. Securing elastic applications on mobile devices for cloud computing.
16. Xuan Hung Le, Sungyoung Lee, Phan Truc, La The Vinh, A. Khattak, Manhyung Han, Dang Viet Hung, M. Hassan, M. Kim, Kyo-Ho Koo, Young-Koo Lee, and Eui-Nam Huh. Secured wsn-integrated cloud computing for u-life care. In Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, pages 1 – 2, January 2010.
17. Yao-Nan Lien, Hung-Chin Jang, and Tzu-Chieh Tsai. A manet based emergency communication and information system for catastrophic natural disasters. In Distributed Computing Systems Workshops, 2009.
18. M. Yuriyama and T. Kushida. Sensor-cloud infrastructure physical sensor management with virtualized sensors on cloud computing. In Network-Based Information Systems (NBiS), 2010 13th International Conference, pages 1 – 8, September 2010.