

Implementation of SCADA Through Cloud Based IoT Devices – Initial Design Steps

Raman Dugyala,
Computer Science & Engineering
Dept.
Vardhaman College of
Engineering
Hyderabad, India.
raman.vsd@gmail.com

N Hanuman Reddy,
Computer Science & Engineering
Dept.
Vardhaman College of
Engineering
Hyderabad, India.
raju9009@gmail.com

Shrawan Kumar,
Computer Science & Engineering
Dept.
Vardhaman College of
Engineering
Hyderabad, India.
shrawan.kumar@vardhaman.org

Abstract—The aim of this work is to develop Internet of Things (IoT) device in place of remote terminal unit (RTU) and integrating it with cloud based SCADA. SCADA systems are used for monitoring the power utilities. The data that is monitored is spanned across a geographical area. The data capture is done by the RTU which are physically located at places where the meters are available. Data on real-time basis will be updated in structured databases for the use of operators. The operators will then use this information and take required actions at the control center. Existing SCADA systems use dedicated Local Area Network (LAN) as computing environment for servers along with dedicated communication network and protocols for Data Exchange. In addition they make use of computer based RTUs for data collection from field devices. In this paper we present a similar architecture where physical RTUs and data exchange centers are replaced by IoT devices and various other AWS services thereby improving the efficiency of the SCADA systems. This work sets a floor for implementing SCADA using IoT and cloud services.

Keywords— IoT, SCADA, Cloud, RTUs, Smart Grid, Power, Energy

I. INTRODUCTION

SCADA[1], Supervisory control and Data Acquisition systems are useful for real time monitoring of systems that are geographically widely distributed from a central location. Sensors and actuators of the system are located in the field and are accessed by the server using a data communication network.

For example, in a power grid many independent companies operate. The companies can be involved in GENERATION TRANSMISSION or DISTRIBUTION of electrical energy and usually coordinated by a GRID[2] operator. These networks are very large and operate cooperatively for mutual advantage. Generally, a 15-minute schedule is prepared for the day for each generating unit (power to be generated) on the basis of the projected load demand by all distribution companies by the grid operator. Transmission companies provide necessary infrastructure capacity (transmission lines) for carrying the power. Distribution companies provide the necessary customer support to users and also responsible for billing and

collection. In turn the generation and distribution companies get paid by them for their services.[1]

Main concerns in this operation are continuous balancing of actual generation and load at all times and adhering to power quality parameters such as maintaining voltage and frequency, ensuring stability of the network for possible contingencies like faults leading to sudden change in load, loss of generation or transmission lines. To this end each company (in particular grid operator, transmission companies and distribution companies) establishes a system for continuous monitoring of distributed data. Each of them will acquire their own data as well as data from other systems as needed. Naturally a need arises to have robust communication network which is secure, reliable, scalable, cost effective and reconfigurable.

II. EXISTING ARCHITECTURE

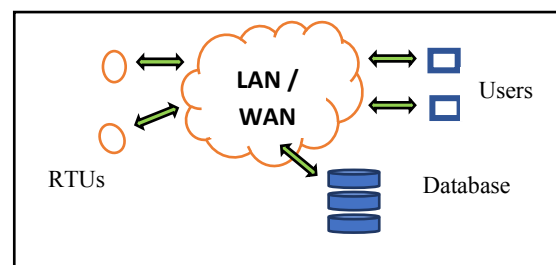


Figure 1: Existing architecture

Existing SCADA systems use

- Dedicated Local Area Network (LAN) as computing environment for servers.
- Dedicated communication network and protocols for Data Exchange
- Computer based RTUs for data collection from field devices.

III. PROPOSED ARCHITECTURE

A similar architecture to the existing architecture is used where physical RTUs and data exchange centers are replaced by IoT devices. Lan is replaced by Cloud and various other AWS services for database functions are used aiming to improve efficiency of the SCADA systems.

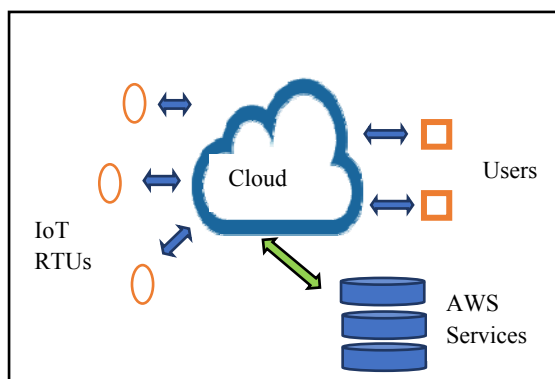


Figure 2: Proposed architecture

The four salient aspects of the architecture include

- ReconfigurableIoT[3][4] devices that interacts with cloud services on one side and with the machine on the other side
- An online cloud-database called DynamoDB is used to make data transactions. Data is stored in the database when published by a user and retrieved by other users having secured access
- Configuring IoT device using Amazon Web Services security credentials and certificates to have sync with the IoT device. Also configuring AWS services to have sync with one service to another
- Building separate user applications to publish and subscribe data on DynamoDB over the other AWS services

A. RTUs

The illustration of various RTU where the data is taken from is shown below

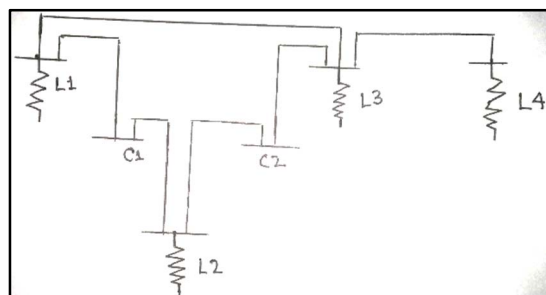


Figure 3: RTUs illustration

Here L values correspond to the respective load values for each bus. These values are published by the user in JSON format so as to populate the DynamoDB table.

B. Users

Users are nothing but the user access devices that help in

- Displaying the captured data
- Configuring the IoT devices&

- Controlling the IoT devices

C. Cloud & AWS Services:

These components include

- DynamoDB[2] Table which is an important cloud resource is a DynamoDB Table - 'data table', which holds the data acquired from the field devices. DynamoDB Table provides for efficient storage on cloud and access to retrieve data based on a combined partition key and range key.
- IoT topic for messages&
- Archive data to be implemented

IV. CLOUD SERVICES USED

A. AWS IoT:

AWS IoT enables Internet-connected devices to connect to the AWS Cloud and lets applications in the cloud interact with Internet-connected devices.

Devices report their state by publishing messages, in JSON format, on MQTT topics. When a message is published on an MQTT topic, the message is sent to the AWS IoT MQTT message broker, which is responsible for sending all messages published on an MQTT topic to all clients subscribed to that topic.

Communication between a device and AWS IoT is protected through the use of X.509 certificates.

You can create rules that define one or more actions to perform based on the data in a message. Rules use expressions to filter messages. When a rule matches a message, the rules engine invokes the action using the selected properties. Rules also contain an IAM role that grants AWS IoT permission to the AWS resources used to perform the action.

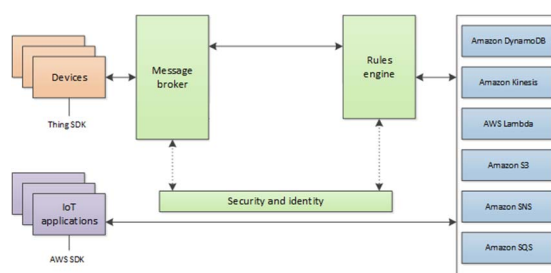


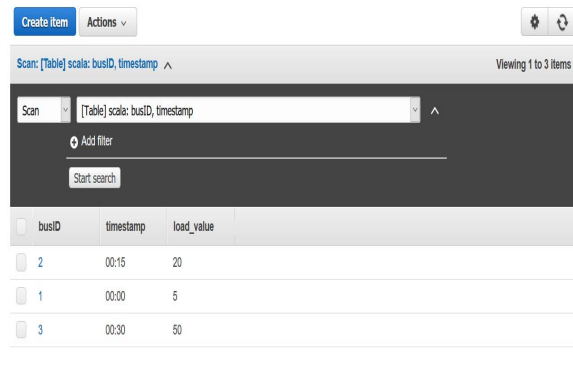
Figure 4: AWS IoT flow

B. AWS DynamoDB:

Amazon DynamoDB is a fast and flexible NoSQL database service for all applications that need consistent, single-digit millisecond latency at any scale. It is a fully managed cloud database and supports both document and key-value store models.

The single most important part of using DynamoDB begins before you ever put data into it:

primary keys and composed of one attribute, called a hash key, or a compound key called the hash and range key. In our case, the primary key is the “BusID” and the range key is the “timestamp”. Using timestamp as the range key helps us to select ranges of time intervals for either editing or displaying data. Other than thesetwo keys, our table consists of another attribute called “info” which has lineflow values i.e, it contains multiple values for the same corresponding key value.



| busID | timestamp | load_value |
|-------|-----------|------------|
| 2 | 00:15 | 20 |
| 1 | 00:00 | 5 |
| 3 | 00:30 | 50 |

Figure 5: DynamoDB Table

C. AWS IAM :

AWS Identity and Access Management (IAM) is a web service for securely controlling access to AWS services. With IAM, you can centrally manage users, security credentials such as access keys, and permissions that control which AWS resources users and applications can access. IAM provides the infrastructure necessary to control authentication and authorization for your account.

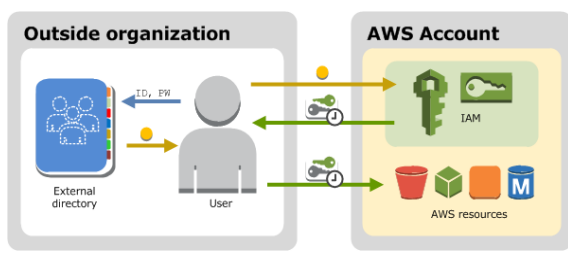


Figure 6: AWS IAM architecture

D. AWS IoT SDK :

The AWS IoT Device SDKs help you to easily and quickly connect your devices to AWS IoT. The AWS IoT Device SDKs include open-source libraries, developer guides with samples, and porting guides so that you can build innovative IoT products or solutions on your choice of hardware platforms. In our project, we use AWS IoT SDK for Python. The SDK is built on top of modified Paho MQTT Python client library. The type of connection used for connecting to the AWS IoT is

MQTT (over TLS 1.2) with X.509 certificate-based mutual authentication.

V. DESIGN DESCRIPTION OF PROGRAM MODULES

A. Creating DynamoDB table:

This will help add a new table to the account. It was named “SCADA”. In an AWS account, table names must be unique with each region. Creating a table can be done in two ways- using the AWS console or through programmatic approach. We used the console to create and configure the DynamoDB table.

Key Schema – This specifies the attributes that make up the primary key for the table or an index. Each keyschema element is composed of:

- **AttributeName** - The name of this key attribute.
- **KeyType** - The role that the key attribute will assume:
 - **HASH** - partition key
 - **RANGE** - sort key

We have two primary keys. The “busID” as the HASH/partition key and “timestamp” as the RANGE/sort key. There will also be another attribute called “info” which contains the “load_values” and “lineflow”. But this is not a primary key.

B. Configuring IoT Device

AWS IoT provides a thing registry that helps you manage your things. A thing is the representation of a device or logical entity. Things are identified by a name. The name of our thing here is “IoTdevice” and the type is “rtu”[5][6][7].

Once we have created our thing, it gives us 4 files to download and securely store in the machine- Device certificate, a primary key, a public key and a rootCA certificate. These certificates and keys are for authentication purpose.

After this, we have to create a policy for our thing. This essentially involves the permissions or actions that can be performed by the device or to the device. The policy that is created needs to be attached to the certificates

and the thing which is an option available in the IoT core console.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "arn:aws:iot:us-east-2:426974861569:thing/IoTdevice"
    },
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "arn:aws:iot:us-east-2:426974861569:topic/rtu"
    },
    {
      "Effect": "Allow",
      "Action": "*",
      "Resource": "arn:aws:iot:us-east-2:426974861569:client/IoTdevice"
    }
  ]
}
```

Figure 7: Policy document used for the thing

After creating and attaching the policy, we have to create a “rule” which acts as a bridge for connecting IoT device to the DynamoDB table. DynamoDB rules allow us to take information from an incoming MQTT message and write it to a DynamoDB table.

Creating rule will have a name field and a description field which contains the information of what the rule does. After that, it will ask for the SQL version and the “attributes” from the message that have to be inserted into the table. For us, the attributes would be busID, timestamp[8][9] and info. Then there is an “action” column to decide what it should do with the recognised data. We used the “Insert a message into DynamoDB table” action. This will take the attributes and their data tuples from the published message and populate it into the DynamoDB table.

C. Creating an IAM user

It is recommended not to use the AWS account root user for any task where it's not required. Instead, we can have an IAM user for root account that requires administrator access. We can create an IAM user directly from the console. The name of our IAM user is “user1”

As we have the user created, we should attach policies for the user to give it permit for accessing any AWS resource or perform actions. These policies are AWS managed policies or AdministratorAccess.

| Policy name | Policy type |
|-------------------------|--------------------|
| Attached directly | |
| AWSIoTThingRegistration | AWS managed policy |
| AWSIoTLogging | AWS managed policy |
| AWSIoTDeviceAccess | AWS managed policy |
| AWSIoTDeviceAccess | AWS managed policy |
| AWSIoTDeviceAccess | AWS managed policy |
| AWSIoTDeviceAccess | AWS managed policy |
| AWSIoTDeviceAccess | AWS managed policy |
| AWSIoTDeviceAccess | AWS managed policy |

Figure 8: Policies attached to user1

There is an option to create a role which is a secure way to grant permissions to entities. This role is accessed or used by the IoT device to give it permissions same as that of the user. The role we created is “iot_role” and this allows IoT to call AWS services on our behalf.

D. Simulator program

All the configurations related to the AWS console have been done. Now the next step is to write programs that publish/subscribe/display data onto the cloud or the database.

The pseudo code for publishing data in json format would be like:

```
config (iotrtu)
Loop:   read   data   (   busID,
timestamp, load_values, lineflows)
        construct data_object
s = encode (data_object)
publish to "rtu" (topic)
#this will publish the data
forever in a loop for an assigned
time
```

Simulation of this program will create changes in the DynamoDB table. This can be done in two ways: (i) by using the rule engine which inputs the data into the DynamoDB table when the message published contains the required attributes (ii) after subscribing the data, the object can be decoded into a json string, which can then be directly inserted into the DynamoDB table by writing a simple DynamoDB python interface program that contains boto3 commands.

```

1 from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient
2 import logging
3 import time
4 import datetime
5 import json
6
7 AllowedActions = ['publish', 'subscribe']
8
9 # Custom MQTT message callback
10 def customCallback(client, userdata, message):
11     print('Received a new message:')
12     print('-----')
13     print('from topic: %s' % message.topic)
14     print('payload: %s' % message.payload)
15     print('-----')
16
17 # For certificate based connection
18 myMQTTClient = AWSIoTMQTTClient("MyBus")
19 myMQTTClient.configureEndpoint("iot.us-east-2.amazonaws.com", 8883)
20 myMQTTClient.configureCredentials("Desktop/Root-CA.pem", "Documents/76a0d32a86-private.pem.key", "Documents/76a0d32a86-certificate.pem.crt")
21 myMQTTClient.configureOfflinePublishQueueing(-1) # infinite offline Publish queueing
22 myMQTTClient.configurePingFrequency(2) # ping every 2 sec
23 myMQTTClient.configureConnectDisconnectTimeout(10) # 10 sec
24 myMQTTClient.configureMQTTOperationTimeout(5) # 5 sec
25
26 myMQTTClient.connect()
27 print('Connected')
28 obj = {"timestamp": "00:00", "info": {"load_value": "5"}}
29 print(obj)
30
31 data_obj = datetime.datetime.utcnow().isoformat()
32 myMQTTClient.publish("my_data", data_obj)
33 myMQTTClient.subscribe("my", 1, customCallback)
34 myMQTTClient.unsubscribe("mytopic")
35 myMQTTClient.disconnect()
36 print('Disconnected')

```

Figure 9 : Program displaying typical publishing and subscribing commands via a client

E. Data display program:

The data that is stored in the DynamoDB table can be retrieved or displayed using a query method. A partition key is a must in the query where as the sort key is optional.

The commands in the data display program will require the Boto3 python SDK. The Boto 3 SDK constructs a ConditionExpression for you when you use the Key and Attr functions imported from boto3.dynamodb.conditions. You can also specify a ConditionExpression as a string.

```

1 from __future__ import print_function # Python 2/3 compatibility
2 import boto3
3 import json
4 import decimal
5 from boto3.dynamodb.conditions import Key, Attr
6
7 # Helper class to convert a DynamoDB item to JSON.
8 class DecimalEncoder(json.JSONEncoder):
9     def default(self, o):
10         if isinstance(o, decimal.Decimal):
11             if o % 1 > 0:
12                 return float(o)
13             else:
14                 return int(o)
15         return super(DecimalEncoder, self).default(o)
16
17 dynamodb = boto3.resource('dynamodb', region_name='us-east-2')
18
19 table = dynamodb.Table('SCADA')
20
21 print("Info from 0:15")
22
23 response = table.query(
24     KeyConditionExpression=Key('timestamp').eq('00:15')
25 )
26
27 for i in response['info']:
28     print(i['timestamp'], ":", i['busID'])

```

Figure 10: Example of a data display program

VI. ADVANTAGES OF THE PROPOSED ARCHITECTURE

A. Security:

The data is published and subscribed through detailed specifications for different levels of security and encryption standards. It is

thoroughly investigated to enable a multi-tiered security model for SCADA.

B. Scalability :

Although it involves real-time data, the computing ability would not degrade or get complicated because data storing and processing is done in the cloud

C. Cost:

Giving secured access to various subscribers for the same data is efficient in money terms because it rather reduces the cost of maintaining offline databases which will account to more time and labour wastages

D. Availability :

It does not involve any centralized servers for controlling the flow of data but only needs an AWS account and secured access to the stored data

E. Reconfigurability :

The service can be reconfigured by the user any number of times based on the requirement.

VII. CONCLUSION & FUTURE WORK

We have presented a novel SCADA architecture where physical RTUs and data exchange centers are replaced by IoT devices and various other AWS services thereby improving the efficiency of the SCADA systems. The following can be implemented in future in addition to the current work

- SCADA exposed as a web service deployed on the cloud
- Study of XML-based protocols defined by Smart Grid[10] standards
- Use of shadow client service (IoT) for internet connection loss for the IoT devices
- Archiving real-time data (day long) for data exchange applications
- IoT RTU to act as a user agent and accept commands from SCADA center.
- Kumar et al. [11-14] proposed different watermarking protocol to provide secure and private transaction between the communicating parties by using cloud.
- Raman et al.[15-20] used information flow analysis to provide security in cloud and IoT as well those techniques are used in SCADA.

REFERENCES

- [1] Use of SCADA Data for Failure Detection in Wind Turbines K. Kim, G. Parthasarathy, O. Uluyol, and W. Foslien Honeywell S. Sheng and P. Fleming National Renewable Energy Laboratory Presented at the 2011 Energy Sustainability Conference and Fuel Cell Conference Washington, D.C. August 7-10, 2011
- [2] Bruhadeswar Bezawada, Kishore k. Raman Dugyala, Rui Li "Symmetric Key based Secure Resource Sharing" Fifth International Symposium on Security in computing and communications (SSCC-17) ISSN:1865:0929 paper was published in Springer-CCIS Journal.2017.
- [3] Research and Application of a SCADA System for a Microgrid Shuangshuang Li, Baochen Jiang *, Xiaoli Wang and Lubei Dong School of Mechanical, Electrical & Information Engineering, Shandong University, Weihai 264209, China; lishuangshuang_sdu@163.com (S.L.); wxl@sdu.edu.cn (X.W.); donglubei@mail.sdu.edu.cn (L.D.) * Correspondence: jbc@sdu.edu.cn; Tel.: +86-186-6935-5366 Academic Editor: Manoj Gupta Received: 8 February 2017; Accepted: 29 March 2017; Published: 31 March 2017
- [4] Shen, Z.Q.; Deng, W.; Pei, W.; Mu, L.S.; Ouyang, H. Design and implementation of microgrid SCADA platform. *Adv. Mater. Res.* 2013, 732–733, 1358–1364. [CrossRef]
- [5] K Vinay Kumar, D Raman, "Secure Overlay Cloud Storage With Access Control And Assure Deletion", *International Journal of Engineering & Science Research*, Vol 4, Issue7, @ e-ISSN 2277-2685, p-ISSN 2320-976 July 2014.
- [6] Palma-Behnke, R.; Ortiz, D.; Reyes, L.; Jiménez-Estévez, G.; Garrido, N. A social SCADA approach for a renewable based microgrid—The Huatacondo project. In *Proceedings of the IEEE Power and Energy Society General Meeting*, Detroit, MI, USA, 24–28 July 2011; p. 7.
- [7] Mehta, B.R.; Reddy, Y.J. *Industrial Process Automation Systems: Design and Implementation*; Elsevier Publishers: Amsterdam, The Netherlands, 2015; Volume 7, pp. 237–300.
- [8] D Raman, Bojja Vamshi Krishna "Ensuring Security Services for Data Storing and Data Sharing in Cloud Computing", *International Journal of Science and Research (IJSR)*, ISSN: 2319-07064, Volume 2 Issue 2, February 2013.
- [9] Pampashree; Ansari, M.F. Design and implementation of SCADA based induction motor control. *Int. J. Eng. Res. Appl.* 2014, 4, 5–18.
- [10] Cao, K. SCADA system research of the grid. *China New Technol. Prod.* 2013, 5, 23. (In Chinese).
- [11] Kumar, Ashwani, S. P. Ghrera, and Vipin Tyagi. "An ID-based Secure and Flexible Buyer-seller Watermarking Protocol for Copyright Protection." *Pertanika Journal of Science & Technology* 25.1 (2017).
- [12] Ashwani Kumar, Satya Prakesh Ghrera, & Vipin Tyagi, "Modified Buyer Seller Wa-termarking Protocol based on Discrete Wavelet Transform and Principal Component Analysis", *Indian Journal of Science and Technology*, 8(35), 1-9, 2015.
- [13] Ashwani Kumar, Satya Prakesh Ghrera, & Vipin Tyagi, "Implementation of wavelet based modified buyer-seller watermarking protocol", *WSEAS Trans. Signal Process.* 10, 212-220, 2014.
- [14] Kumar, Ashwani. "Design of Secure Image Fusion Technique Using Cloud for Privacy-Preserving and Copyright Protection" *International Journal of Cloud Applications and Computing (IJCAC)* 9.3 (2019): 22-36.
- [15] Raman Dugyala, Bruhadeshwar Bezawada, Rajini Kanth V. Thatiparthi, Sai Sathyanarayan "Information Flow Tracking Approach for Vulnerability Signature Generation in Web Applications", Vol 10, Issue14, @ P- ISSN 0973-4562 e- ISSN 1087-1090, Jan 2015.
- [16] Raman Dugyala, N Hanuman Reddy, N Chandra Shaker reddy, J Phani prasad "A Roadmap to Security in IoT" *International Journal of Applied Engineering Research* ISSN 0973-4562 Volume 12, Number 19 (2017) pp. 8270-8272
- [17] N. Chandra Sekhar Reddy, B.Rama, B. Madhuravani, Raman Dugyala, N. Rajasekhar "MOBILE NETWORK SERVICES FROM THE PRESENCE CLOUD" *International Journal of Pure and Applied Mathematics* ISSN: 1314-3395 (on-line version) Volume 119 No. 14 2018, pp.95-100
- [18] Raman Dugyala, N Hanuman Reddy, Raghuram G "DECENTRALIZED SECURE ONLINE DIGITAL DATA REGISTRATIONS" *International Conference on Advanced Machine Learning and Soft Computing (ICMLSC 2018)* *International Journal of Engineering and Technology (UAE)* ISSN: 2227-524X
- [19] Raman Dugyala, Bruhadeshwar Bezawada, Rajini Kanth V. Thatiparthi, Sai Sathyanarayan "Static Program Behavior Tracing for Program Similarity Quantification" *Proceedings of the First International Conference on Computational Intelligence and Informatics* Volume 507 of the series *Advances in Intelligent Systems and Computing* pp 321-330 DOI 10.1007/978-981-10-2471-9_31 Print ISBN 978-981-10-2470-2 Online ISBN 978-981-10-2471-9, May 2016
- [20] A Monika, D Raman, "Justified Cross-Site Scripting Attacks Prevention from Client-Side" *International Journal of Advanced Technology & Engineering Research*, Vol6, Issue7, @ ISSN : 0975-3397, July 2014.