Anomaly Detection in Cloud Systems using Machine Learning Project Report

Manish Choudhary Shailesh Lohia Sahil Chadha Aditya Upreti

ABSTRACT

Virtualized infrastructures are designed to be scalable and highly available and yet they suffer from chronic instabilities leading to inconsistent results, performance issues and downtime, making such infrastructure unreliable. The most common reason for this is system and network anomalies. The goal of our project is to collect measurement data corresponding to the various system and network level performance parameters from all the monitored machines in the cloud infrastructure and classify these events as anomalies or expected behaviour based on the optimal performance level of each machine as learnt by our system. Our system is based on a simple client server model, where multiple clients send system and network parameter values to the server which then learn their optimal performance using self-organizing maps. Once the parameters are learnt, the system use new incoming information to detect if there are any anomalies that are expected to occur. We believe that by detecting such anomalies and providing it to the administrators, we will be able to help avoid performance issues and downtimes and thus make the cloud infrastructure more reliable.

1. INTRODUCTION

In this era of computer systems, cloud based systems and data centres have become the backbone for most of the online computations and businesses. Cloud based services enjoy the benefits such as high availability and scalability. Multi-tenancy helps in providing the services in an economical way. Cloud systems can be used as IaaS, PaaS, SaaS or NaaS, and thus, have become one of the prime focuses of research and innovation. Because of the large number of nodes, mixed bandwidth interconnected networks and geographical spread, the cloud infrastructure faces chronic instabilities leading to the unreliable system. It has become a question of “when failure happens” instead of “if failure happens”. Inconsistent results, performance problems and downtime are the day-to-day problems faced by the cloud users as well as administrators. System and network anomalies either due to accidents or malicious behaviour are the key contributors to the unreliability of the cloud systems. We have developed an anomaly detection system (ADS) using unsupervised machine learning techniques in an attempt to make the cloud dependable. Our system has a distributed client-server architecture involving virtual machines to be monitored as clients, detection server pool and database cluster. For prototyping, we have implemented two detection servers both acting as primary servers for half of the clients and as secondary for the other half of the clients. We periodically monitor the clients by collecting data for various performance parameters and send this data to the detection servers. Detection servers take this data and store into the database cluster. Our system learns the optimal performance for each of the clients monitored over a period of time and then tries to detect anomalies in monitored virtual machines based on this. We have also focused on the scalability, failover, fault tolerance and security aspects of our system making it reliable. We believe that using such an anomaly detection system to monitor the cloud infrastructure would help in making the cloud based systems reliable by giving the real time updates about the behaviour based anomalies.

2. RELATED WORK

Anomaly detection in the cloud infrastructure is very important from the perspective of the consumers in adopting the cloud as it ensure the meeting of service level agreements and hence there is a lot of research that has been. [1] is a survey on the various kinds of anomalies that can occur in the cloud infrastructure and the various machine learning algorithms that can be used in order predict these anomalies. The author discusses that though there are a number of supervised learning techniques that are available for the detecting anomalies or pattern, these require labelled data that is representative of the overall system performance. Since this is difficult, unsupervised learning is usually the approach followed for the development of these systems. In [2] the authors follow a supervised learning approach for anomaly detection. In this, they make use of the principal component and most relevant principal components for the purpose of anomaly detection. In this system, the author collect the initial dataset and assume that the dataset contains only normal information, thus bypassing the need for labelled data. [3] makes use of the unsupervised learning - self-organizing maps as the method for the anomaly detection and use a threshold method in order to detect whether a certain input is an anomaly. Our system, is inspired from the work done in [3], though we would like to try other learning algorithms from [1] as well.

3. SYSTEM ARCHITECTURE

ADS has a client server architecture backed by MySQL database cluster for persistency (as shown in figure 1). The system consists of following key components.

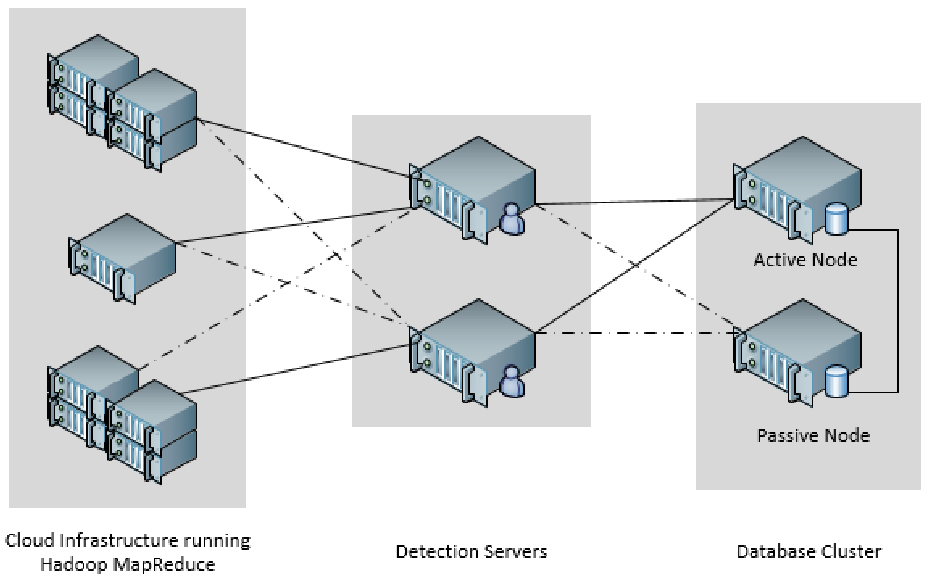


Figure 1: Client Server Architecture Backed by Database Cluster

3.1. Clients

The virtual machines which should be monitored for anomalies are the clients in our system. From implementation perspective, we have considered some client machines running Hadoop map-reduce computations which is one of the common operations performed in the cloud environment.

We have developed an ADS agent that runs on each of the client machines to measure and collect the data for various performance parameters types periodically and sends them to its assigned primary server or the secondary server in case the primary server fails.

Network	**CPU**	**Memory**	**Disk**
Latency	No. of processes	Utilization	Cache Read bytes/min
Throughput	System Time	Page Faults	Buffer Read bytes/min
Packet Loss	User Time		Write bytes/min
Open ports			Service Time
			Total Files

Table 1: Performance Parameters being monitored

The various components of the ads agent (shown in figure 2) running on the clients have been described below

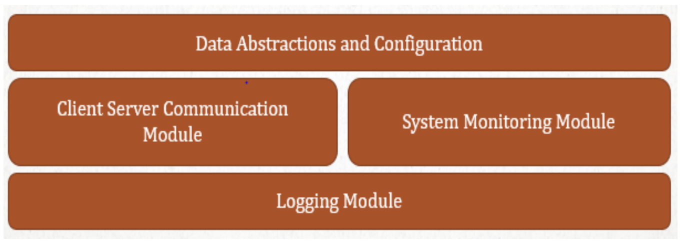
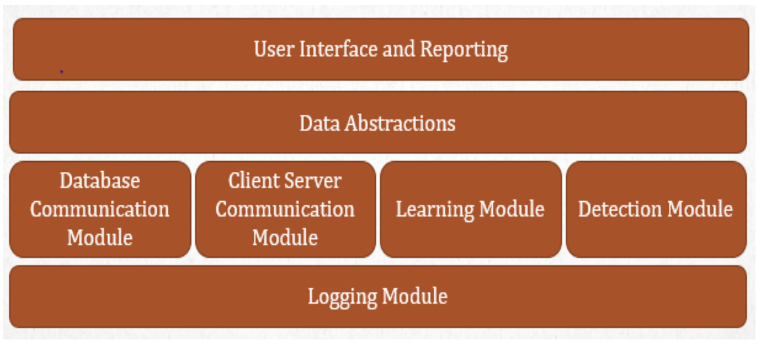

Data Abstractions and Configuration
Client Server Communication Module
System Monitoring Module
Logging Module

Figure 2: Client Agent Modules Abstraction

a) **System Monitoring Module:** It provides the APIs to get the data for the various performance parameter as shown in table 1.
b) **Client Server Communication Module:** It facilitates the transfer of the performance data collected on the client machines to the detection server in an XML format. It also allows to the client to request Unique ID and a repository on the detection server via RPC. In addition, it also provides the functionality to monitor the liveliness of the server.
c) **Logging Module:** It provides an optional logging functionality that can be enabled on clients for auditing purpose.
d) **Data Abstraction and Configuration:** Configuration file enable the set the various configuration parameters as per the requirement avoiding the direct interaction of users with the core code. Data abstractions used are the XML file format for transferring the data and customized multi-level dictionary for keeping the data in-memory.

3.2. Detection Servers

Two detection servers have been implemented for prototyping. In practical scenarios, a pool of detection servers can be implemented. For load balancing, each detection server is responsible for serving half of the clients. Therefore, each server acts as the primary server for half of the clients and as the secondary or backup server for the other half. Detection servers are responsible for hosting RPC service to serve the client requests, storing and updating the data received from the clients to the database cluster as well as for the execution of learning and prediction modules to detect anomalies.


User Interface and Reporting
Data Abstractions
Database Communication Module
Client Server Communication Module
Learning Module
Detection Module
Logging Module
Figure 3: ADS Modules Abstraction on the Detection Server

The various modules that provide the base for the detection server are:

a) **Client Server Communication Module:** It enables the server to host RPC service, allows the client to do server liveliness monitoring and helps the ads agents to push the XML files containing performance data.
b) **Database Communication Module:** It facilitates the servers to communicate with the database server. It provides the various APIs for creating and connecting to database, for inserting the client information and collected data into tables and fetching data from data using various filters as well as functionalities to commit or rollback the transactions.
c) **Learning Module:** This module provides the functionality to learn the optimal performance of various clients with the help of "self-organizing map" machine learning technique. It is run once initially for each client on a configurable sufficient amount of data collection.
d) **Detection Module:** This module executes the prediction phase periodically for the data collected in the last epoch using the optimal performance learnt initially and classifies if data in a particular epoch is anomalous.
e) **Logging Module:** It facilitates verbose logging for auditing purpose in case the administrators want to go through the logs on detection of some anomaly or for debugging purpose.
f) **Data Abstractions:** XML for performance data and SQLAlchemy Objects database related data
g) **User Interface and Reporting:** Interface that facilitates an administrator to get the results based on some criteria.

3.3. MySQL Database Cluster

In the cluster, two mirrored database servers are maintained for persistency and fault tolerance. The database schema is shown in the Figure 4 below. "Clients" table is used to maintain the information (such as client UID, hostname and IP Address) of the clients. A separate table is maintained for each parameter type (as shown in table 2) which are updated every time the server receives data from the some client. "Servers" table is to maintain metadata for the detection servers in the pool. "Anomalies" table is used to store the results of the prediction module in case some anomaly is found.

VM ID	Timestamp	<Param 1>	<Param 2>	…	<Param n>

Table 2: Table structure for storing performance parameters

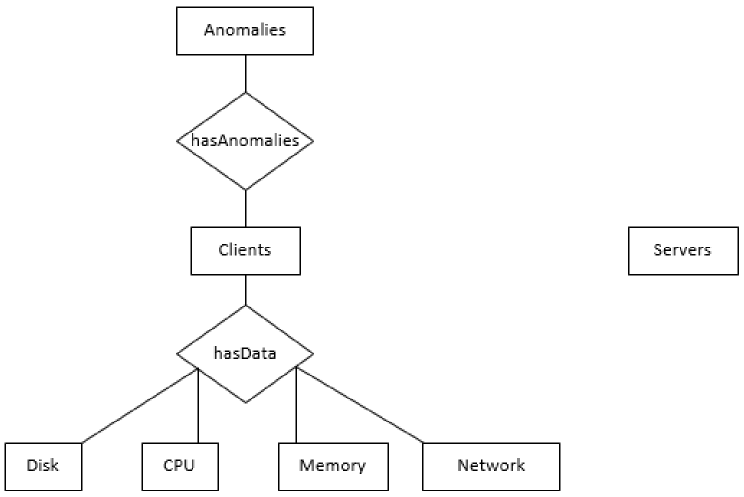


Figure 4: Database Schema for ADS system

4. IMPLEMENTATION

The ADS system has been developed in Python completely. The implementation of the system is described below for each component.

4.1. ADS Agent

ADS client package has been developed that runs on each of the client machines to be monitored. “ads_client.py” is the main controller module and “ads_client_config.py” is the configuration file for the file tuning purpose. ADS agent performs following activities on the client side:

a) Initially, make an RPC call to the primary server to get an UID which will be used in all the future communication with the detection server.
b) Make another one time RPC call to the primary server requesting for the creation of a repository on the server.
c) Perform following operations in an infinite loop:

.

i. Check if the primary server is alive. If not, request the secondary server to create a repository on its disk using one time RPC call. The liveliness has been monitoring by checking if the acknowledgement file in the server repository gets updated in the timeout period. The SFTP (provided by Paramiko SSH) is used to get the stat of the file and “st_mtime” is check to know time of last modification.
ii. Fetch the data for the various performance parameters using the APIs provides by the “system_monitor.py” module and store in a dictionary.
iii. Generate and XML file with the collected data and some metadata (client IP, ID and Name) naming it using the current epoch time. XML Element Tree library is used for to realizing this task.
iv. Transfer the XML to the primary server if alive otherwise send it to the secondary server. SSH has been implemented using “Paramiko” to transfer the file to the repository created on the detection server. SSH provides the functionality of SFTP to transfer the file securely.

4.2. ADS Server Implementation

The “ads_config.py” is a configurable file available to the administrator for fine tuning the parameters such as monitoring period, collection period and parameter types. The “ads.py” is the main controller module which forks two processes “store_data_in_db” and “run_rpc”.

4.2.1. RPC Service

“run_rpc” starts the RPC service providing following functionalities.

a) Public function that can be called by the clients to get a newly generated client ID. The UID is globally unique and is generated by computing SHA-256 of (Client IP Address + Client Hostname + Current Time + Random Number). This information is also inserted in the “clients” table. In case the client is already registered (checked with IP and hostname), existing UID of the client is returned.
b) Another function that can be invoked for creation of a new repository on the server which can be used by clients to SSH XML data files. The absolute path of the repository is returned to the client. An acknowledgement file in this repository is also created. At this time, number of processes equal to the number of parameter types are spawned. These processes are responsible for carrying out the learning and prediction tasks.

4.2.2. Data Storage

“Store_data_in_db” performs following operations in an infinite loop.

a) Check if there are new XMLs from some client. If yes, read the XML into a dictionary and update the current IP address and hostname of the client into the database. If the UID is not present in the database, this file is ignored as a genuine client should have a valid UID existing in the database.
b) Update the database with this new data and commit the changes once after updating all the tables.
c) Update the acknowledgement file with the XML file name and delete the XML file.

4.2.3. Learning and Prediction

Learning and Prediction Processes are responsible for carrying out following tasks.

a) Check every 5 minutes, if the requisite amount of data is stored in the database for the client and parameter for which the process is configured.
b) Train on the configured amount of data and learn the weights for each output node in the self-organizing map.
c) After every 5 minutes, pull the latest data from the database for the client and the parameter for which the process is configured and run the prediction algorithm on the pulled data. If the prediction algorithm finds a certain number of anomalies in the data and reports it.

The intervals specified above and the number of anomalies detected before reporting are configurable parameters.

4.2.4. Implementation Strategies

The implementation strategies for various modules are described below.

a) **DB Communication** has been implemented using python's SQLAlchemy which is an Object Relational Mapper. All the tables can be represented and operated on as objects. It facilitates the creation of database tables, connection the database, inserting/update/fetch from the database, commit and rollback. This technique helps in preventing the SQL Injection attacks.
b) **RPC** has been realized using SimpleXMLRPCServer and xmlrpclib.
c) **SSH and SFTP functionalities** have been provided using paramiko.
d) **XML Read/Write Operations** have been implemented using XML ElementTree.
e) **Machine Learning** implementation utilizes the scipy library

5. EVALUATION AND DISCUSSION

Our system was evaluated while running in a prototype environment. The prototype environment consisted of the following:

- 4 client machines, running the Hadoop Mad Reduce sample sorting application.
- 2 detection servers, running the ADS server application.
- 2 database servers, running the MySQL database cluster.

All 8 nodes, mentioned above were part of the Emulab virtualized environment and were connected to each other using a switch. Each node had the following configuration:

- Dell PowerEdge 2850s with a single 3GHz processor
- 2 GB of RAM
- 2 10,000 RPM 146GB SCSI disks

Each node had Ubuntu 12.04 operating system running on it.

5.1. Client Performance

In this section, we evaluate the change in the performance of the client due to the execution of the client package. The clients were running a sample sorting application in the Hadoop Map Reduce Framework. To test the performance of the client package performance, we ran the client package while the client was otherwise idle as well as when it was running the Hadoop application. We saw that the average time taken to run one loop of the client package took about 15.2 seconds while the clients were otherwise idle and 17.39 seconds when the clients were running Hadoop. As can be seen in the Figure 5, while the execution time for the client package is fairly constant while the client was idle, the execution time varies during the Hadoop execution. We believe that this may be because of the difference in resource utilization while running Hadoop which slows down the execution of the client package.

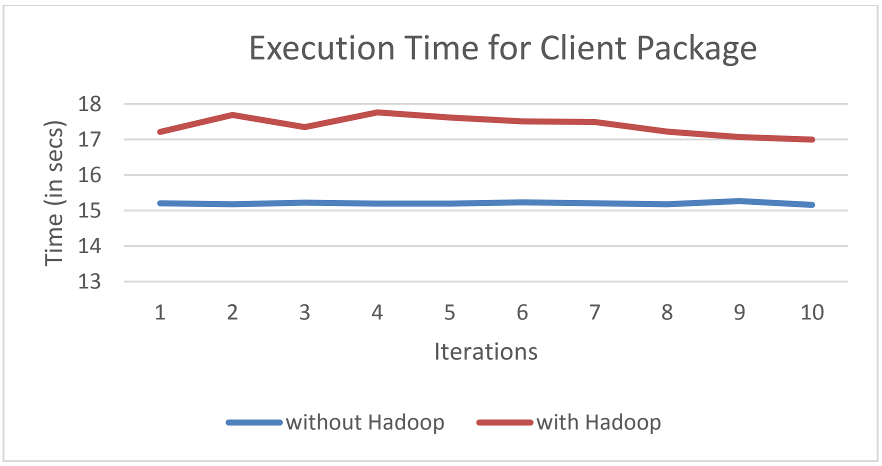


Figure 5: Execution time for the client package over a number of iterations

5.2. Server Performance

In this section, we evaluate the performance of our system on the detection servers. The servers were dedicated to only perform logging, communication with clients and database and running the learning and detection module. Figure 6 shows the change in CPU and memory utilization over time. The first half of the graph (1-10 ticks along the horizontal axis) shows the performance on the server before the ADS server process is started. As can be seen in the graph, that the memory utilization is constant around 39% while there are momentary spikes in the CPU utilization. Once the ADS Server process begins, we see that there is an increase of 4 to 6% increase in memory utilization once the client starts connecting to it. We see that there is little or no change in the CPU utilization. There is are momentary spikes when one of the server processes wakes up and performs some action, however, it is usually keeping low at about 40%.

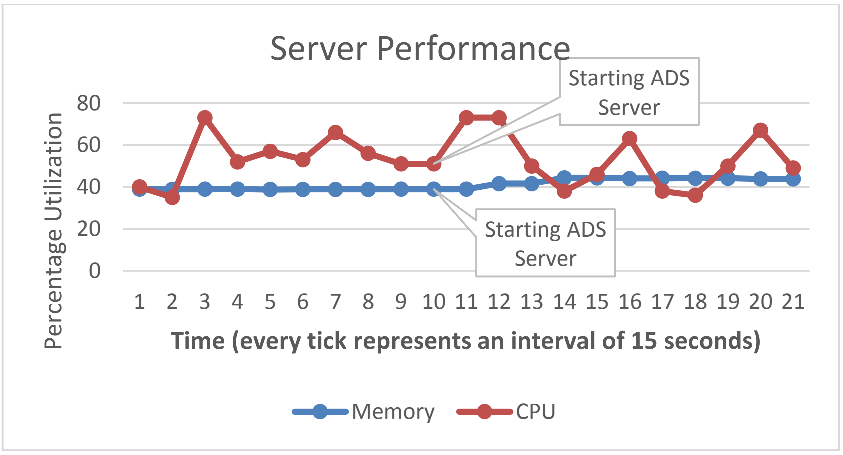


Figure 6: Server Performance over time before and after starting the ADS server

5.3. Server Failover

Server failover mainly involves the client realizing that the primary server is no longer acknowledging to its data files and hence it must start reporting to the secondary server. For this purpose, we have a configurable timeout period that the client waits for before it decides that the primary server is down and starts communicating with the secondary server. When the client starts to communicate with the secondary server, the secondary server needs to create a repository for the client and spawns the processes for the client. We have seen that the time taken for this is roughly 115 ms on the Emulab nodes.

5.4. Scalability

In this section, we evaluate the change in performance of the system while more and more clients connect to the clients. We see that there is an increase of about 2.5 to 3% in memory utilization per client. Once again the CPU utilization does not show much variation before or after the additional clients connected to the server. The servers on Emulab have only 2 GB of memory, which is very less for the servers that would be used in an actual implementation. This shows that each server is scalable across multiple clients and could easily support more than 50 clients on the Emulab server.

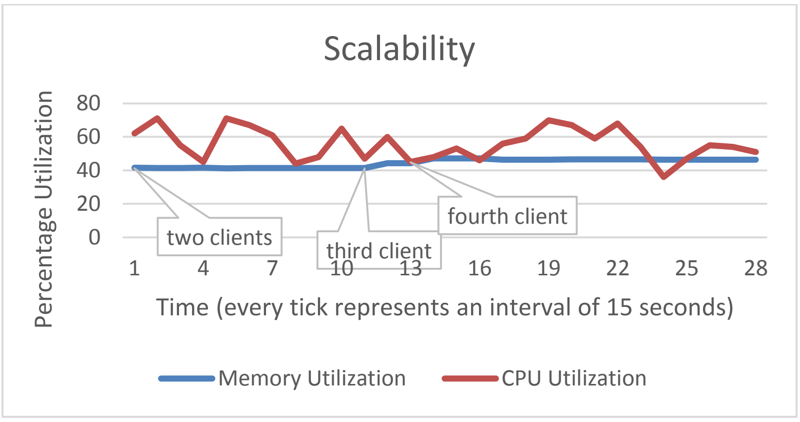


Figure 7: Server Performance as multiple clients increase

5.5. Anomaly Detection

In this section, we discuss the performance of the anomaly detection in the servers. During experiments, we saw that the slight modifications in the parameter values does not lead to the system detecting anomalies, however, major changes in the parameter values would lead to the anomalies being detected as shown in Figure 8. We did see that there were major changes in the number of false positives that were detected with the amount of data that was being provided during the training phases. This was expected as small amount of data did provide the system with a representative state of the system.

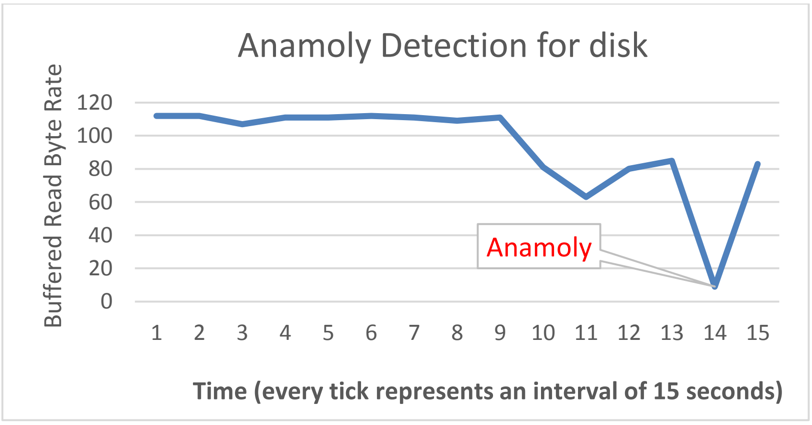


Figure 8: A representation of when our system would detect an anomaly

6. CONCLUSION AND FUTURE WORK

We have been successful in building a fully functional anomaly detection system with all the components proposed. Our system is scalable, efficient and fault tolerant as well as the machine learning algorithms provide the desired correctness in anomaly detection. During our experiments we saw that the change in amount of data being provided to the anomaly detection module for training directly impacts the number of false positives that are produced by the system.

In the future, we would like to implement, other machine learning algorithms and evaluate their performance with respect to that of self-organizing maps. We would like to build an interface (as mentioned in our proposal) for the administrators that could not be implemented in our current implementation due to time constraints. We would also like to implement a functionality that would allow the administrator to retrain the system for certain nodes to prevent the triggering of repeated anomalies.

Finally, in the original proposal we had mentioned that we would be implementing a 9 node client infrastructure however, due to lack of resources, we were able to only test our system on 4 client nodes. We are, therefore, interested in implementing our system on a real cloud infrastructure and study the effects of our system on such an environment.

7. TEAM PARTICPATION

The participation of the team member in project development is as follows:

- **Manish Choudhary:** Development of the client package and server package with communication modules involving RPC, SSH and SQL Alchemy based database communication.
- **Shailesh Lohia:** Development of the machine learning module for the system.
- **Sahil Chadha:** Development of the system monitoring module.
- **Aditya Upreti:** Development of the network monitoring module.

In addition, the integration and the testing of the system have been done by Manish and Shailesh. The setup of infrastructure and running the system on emulab to generate data for evaluation have been achieved by Sahil and Aditya. Finally, the data mining from the generating data and report have completed by Manish and Shailesh.

8. REFERENCES

[1] Chandola, Varun, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." *ACM Computing Surveys (CSUR)* 41.3 (2009): 15.

[2] Guan, Qiang, and Song Fu. "Adaptive Anomaly Identification by Exploring Metric Subspace in Cloud Computing Infrastructures." *Reliable Distributed Systems (SRDS), 2013 IEEE 32nd International Symposium on*. IEEE, 2013.

[3] Dean, Daniel Joseph, Hiep Nguyen, and Xiaohui Gu. "Ubl: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems." *Proceedings of the 9th international conference on Autonomic computing*. ACM, 2012.