International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Tri-Angular Monitoring Approach for Real Time Container Migration

[1]Babu Ram Dawadi, [2]Rajendra Paudyal

[1]Department of Electronics and Computer Engineering, Tribhuvan University, Pulchowk Campus, Nepal
[2]Engineering Division, Nepal Telecom

Abstract: ***Container is the modern age distributed applications packaging toolkit over the cloud environment [1]. It features the management of applications with easy plug and play ability, migration, replication, relocation, upgrading et cetera in the real time. Such containers running different applications over the cloud infrastructure may consume different resources that require real time monitoring. Monitoring of the applications over the distributed cloud environment is important for the better service delivery [2]. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it all out as one package to the multiple cloud. This gives a significant improvement in performance and size of the applications [3]. For efficient management of containerized applications in the distributed cloud environment, we have conceptualized to develop JAVA based docker host monitoring with Container Migration Monitoring (CoMMon), a tri-angular monitoring approach in the IPv6 network. Docker is open source platform independent tool to create, deploy, and run applications by using containers. The monitoring probe collects different monitoring metrics and sends to remote surveillance system in the JSON format. The term tri-angular is proposed in this paper with the concept that the source machine sends the metric information to the monitoring station before and after the migration of containers while destination node sends the status of migrated container to the same monitoring station after the container is received and verify the message from source and destination by unique container identification number (ID).***

Keywords: Container, Cloud, Docker, IPv6 Network, Tri-angular, Time-series

1. Introduction

Resource virtualization and different level of service provisioning is the main achievement of the cloud computing. Cloud and the data center operation for the best service delivery use extensively the virtual machines with its profound benefits on resource control and isolated workload management [4]. Cloud computing leverages the presence of enterprises with powerful data centers that provides resources on demand: the provision mechanism relies on virtualization.

There are two different approaches to virtualization: (i) Hypervisor technique, realization of the whole operating system stack over a slice of the hardware resource (ii) A container technique, uses the lower layers of the running operating system (OS) to implement one or more containers that are isolated environments to run an application. Both types of virtualization are currently available from public cloud providers. The choice between the two approaches is a trade-off between flexibility and efficiency. The hypervisor technique is less efficient, since it implements the whole OS stack, but, for the same reason, it is agnostic with respect to the host OS. Instead the container is layered over the current OS kernel, and therefore it is more efficient. A container-based approach evolves towards the realization of complex but agile distributed architectures, composed of small and specialized services: the micro-service approach is a promising design paradigm that is tightly bound to (or merging with) the container technology. Container-based virtualization presents an interesting alternative to virtual machines in the cloud.

The difference between the two approaches may be immaterial for the public cloud provider but the container-based approach is definitely more attractive for the designer who wants to implement (or develop) a distributed architecture: the limit of adopting a certain OS may be irrelevant, while performance issues are a key factor. Virtualization serves flexibility, security, ease to configuration and management, reduction of Total Cost of Ownership (TCO) for many business systems. However, it also meanwhile incurs some overhead of performance. Performance monitoring of the virtualized machine (VM) is key part for the businessman to maintain reliability of the system and load balancing of the VM. With the innovation of docker based container technology, there seems to require extra levels of abstraction in the virtualization still to improve the efficiency of services in the virtual world.

Several studies have been performed to improve the performance in the Virtual Machine technologies including VM synthesis, repository management, Pareto SLA et cetera. [5-7]. Docker provides efficient management of container, which has several added benefits in compared with VM operations and management. Cloud services (IaaS, PaaS and SaaS) have direct relationship with Serviced Based Applications (SBA) in the business process and management where end users are to be guaranteed with quality and performance [8]. For this, efficient service monitoring on each layer is required. These days service providers provide Monitoring as a Service (MaaS) in the cloud with different dimensions and motivations [9]. For the proper management, the virtualized machines need to be continuously monitored can only be done through the surveillance system. Surveillance system monitors the performance of the host machines during container migration from one machine to another machine within or outside the cloud. Only the surveillance system helps in maintaining the service level agreement (SLA) of the services provided. With the increase of number of applications in the cloud, resource management is critical. Customers want to switch the

infrastructure service to meet the quality they want. On the other hand, cloud service providers, after monitoring the resources, shall migrate the resource under critical condition over available idle infrastructure. We present our surveillance (monitoring) system while migrating the virtual resources to track for the successful migration.

2. Virtual Machine and Container

The container technology in the open source environment has been conceptualized since past several years as a Linux Container (LXC) that support isolated name spaces. Docker provides additional advantages for efficient container management and operation. Efficient virtual machine management and monitoring in the federated cloud environment is still in the phase of research but in the meantime the container based technology is replacing the heavy weight virtual machine technology with efficient and light weight container technology for cloud applications management, monitoring and operations. Docker containers are featured as below [10]:

- Portability of applications over multiple clouds: Docker provides the platform for building applications and its dependencies into a single object as a container which shall be migrated to any other docker enabled machine and executes independently.
- Application centric: Docker engine is highly application centric. It provides a best packaging of applications over container.
- Tools specific: Docker tools help developer to automatically assemble a container from source code with full control over application dependencies, build tools and packaging.
- Container re-use: Docker based containers are the base images which shall be used and reused for packaging any kind of applications and migrate it to any other platform.

Virtual machines run with full OS, where all the functionalities of OS required for its operation, while Containers share the host's OS and are therefore lighter in weight, start up is comparatively faster, and have better performance. The right way to think about Docker is thus to view each container as an encapsulation of one program with all its dependencies. The container is pluggable to any host and it might have everything it needs to operate. In a virtual machine, valuable resources are emulated for the guest OS and hypervisor, which makes it possible to run many instances of one or more operating systems in parallel on a single machine. Every guest OS runs as an individual entity from the host system consumes higher resources. On the other hand, Docker containers are executed with the Docker engine rather than the hypervisor. Containers have therefore less isolation and greater compatibility possible due to sharing of the host's kernel. A virtual machine could take up several minutes to create and launch whereas a container can be created and launched just in a few seconds. Also, a single server can pack more than one containers. Docker Containers can run inside Virtual Machines though they are positioned as two separate technologies. Working in heterogeneous environment, VM provides high flexibility whereas Docker containers' prime focus is on applications and their dependencies [11]. Hence Containers are wrapped-up applications and pieces of software that include all dependencies but use a shared kernel with other containers (applications). Essentially, containers are isolated processes in the user-space on the host operating system. While the Hypervisors abstract the entire device, containers just abstract the OS kernel [12].

3. Docker and Container Monitoring

However one docker engine may run more than one isolated containers, the increasing number of containers consumes resources shall make the host system more critical. VMs are normally allocated a fixed amount of resources limiting the applications to run. Resources utilized by containers are in sharable mode. By default, when a container is launched, there is no memory limits. This can lead to a single container consumes whole memory leading to the system unstable. However the tools like cgroups help manage the resource utilization by a container. Hence host's capacity planning and infrastructure load balancing is sensitive in the infrastructure running containers over the docker. For this, quality of service metrics like CPU utilization, Memory, Disk, I/O and Network utilization are to be continuously monitored as docker metrics. Measuring of the docker metrics help to automatic scheduling of resources like i) stop or move the containers where resource utilization crossed the limit, ii) start new containers if the resources are idle. Docker metrics collected at real time shall be stored into the time series databases (like influxDB) and visualization by graphing tools (like Graphana). Monitoring of Docker and Container has the main objective to optimize the resources and properly balance the load of existing infrastructure. Figure 1 depicts the basic concepts of collecting Dockers' metrics like CPU, Memory, Disk I/O and Network utilization which shall be stored into the time series database (TSDB) as well as processed with the defined rule that provides the threshold for every metric measured and help for automatic adaption of resources in the cloud. The visualization of metrics in the graphs provides the health status of docker engine and its associated containers to take proper decision for better SLA.

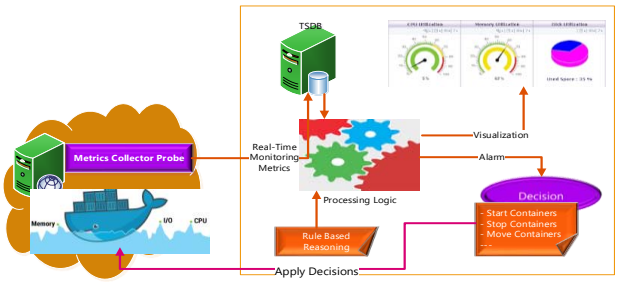

Figure 1: Docker container monitoring and decision making approach

4. Related Work

Figure 1 provides the concept of monitoring the docker.

There are many monitoring systems already developed for docker based platform and container monitoring [13-15]. Similarly, a self adaptable platform has been proposed in [16] to make the infrastructure self configurable and adaptable based on the decision performed by the system after getting alarm of resource overloading in the cloud.

Docker itself provides the command line tools to its client to extract the status of container's resource consumption. "*Docker stats*" command provides the CPU utilization for each container, the memory used, total memory available, network input/output by the container. Additionally Docker provides the remote API via netcat [16].

CAdvisor [17] is a free and open source web based tool that simply visualizes the command line output generated by Docker stats and Docker Remote API. It provides the graphs for CPU usage, Memory usage, Network throughput and Disk space utilization. It does not have alerting system when resource consumption exceeds the limit.

Scout [18] is another web based hosted monitoring tool for docker containers. It can collect metrics from any hosts and containers then visualize it. It has logical reasoning engine that generate alerts based on those metrics and its defined threshold.

DataDog [13] provides the more professional cloud monitoring service including docker and containers. It collects metrics about CPU usage, memory and I/O for all containers running in the system including counts of running and stopped containers as well as counts of docker images. It has flexible alerting system where we can set the metrics threshold as per the SLA.

Sensu [19] is an another open source monitoring framework that supports the docker/container monitoring after installing docker-Sensu-Server[20] container and provides self hosted centralized metrics monitoring service. It also supports many plugins for monitoring to support different business needs.

Prometheus [21] is also a self-hosted open-source system monitoring and alerting tool which collectively provide metrics storage, aggregation, visualization and alerting service that collect the metrics on the PULL basis. It features the multi-dimensional time-series data model, flexible query language and autonomous self-hosted and shall be maintained by any organization.

Logentries [22] is a log analyzer provides monitoring service based on the log file analysis. It recently added docker container monitoring based on the log. It enables logging for docker of all sizes, aggregate the log for containers, hosts and applications.

Ruxit [23] is the dynamic and scalable Dockerized applications monitoring tool able to monitor micro-services running on docker containers. It automatically detects the creation of new containers and monitors the applications and services contained within them. This helps to applications capacity planning and load balancing of containers optimizing container orchestration and clustering.

Sematext [24] has added docker monitoring services into its SPM solution. It is capable to monitor container lifecycle events (create, exec_create, destroy, export) and runtime events (like die, exec_start, kill, pause, restart, start, stop, unpause) with containers. It correlates with metrics, alerts and anomalies. SPM-AGENT-DOCKER [25] installed on every docker host captures the appropriate events and ship this to the SPM which visualize into web based system.

Looking into the State-of-the-Art in the Docker Container monitoring, requirement of application migration to the best performed platform/network help increase the efficiency of services to customers. For infrastructure load balancing, the possible movement of containers and its real time monitoring during migration within or outside the cloud network is the main purpose of this paper. Monitoring of migration system is required to guarantee that the new decision to move the applications form one cloud to another cloud increase the performance and meet associated service level agreement (SLA). Every cloud server provider maintains the minimum quality of their services as well as they have to meet the minimum requirements to be fulfilled while providing the services agreed upon with customers as SLA. There might have different indicators that affect the quality of service. Our proposed monitoring platform "*CoMMon*" is open source docker based monitoring system shall be expandable to have alerting system and easily integrated with other monitoring platforms.

Table 1: Properties of container monitoring platforms

Platform	*Docker Support*	*Container Status Monitoring*	*Container Migration Monitoring*	*Container Log Monitoring*	*Open Source*	*Alert System*
CAdvisor [17]	Yes	Yes	No	No	Yes	No
Scout [18]	Yes	Yes	No	-	Yes	Yes
DataDog [13]	Yes	Yes	No	Yes	Yes	Yes
Sensu [19]	Yes	Yes	No	-	Yes	-
Prometheus [21]	Yes	Yes	No	-	Yes	Yes
Logentries [22]	Yes	Yes	No	Yes	Yes	No
Ruxit [23]	Yes	No	No	No	-	No
Sematext [24]	Yes	Yes	No	Yes	-	No
*CoMMon**	*Yes*	*Yes*	*Yes*	*No*	*Yes*	*Yes**

5. Our Contributions

The major objective of moving/migrating containers to another cloud or another system within the same cloud is to guarantee better performance and meet the service level agreement. Basically system throughput, network throughput, memory and CPU utilization are the major indicators that may lead to take the decision required to migrate containers form one host to another. The migration decision shall be from service provider side or from customer side depending upon the type of service and agreement. Looking into the existing monitoring tools reviewed in section 4, our concept is not only to monitor system and application performance of containers but also monitor the events related to container migration. Existing open source tools shall be used for host status monitoring. Our tool is useful for the purpose of container migration whether the container is successfully migrated to destination or not. Its associated metrics shall be measured from the destination network after migration. We have developed our own java based client server communication system, in which the agent simply detect the container events at the source and integrate the destination events after successful migration to destination. It can also performs the host system status monitoring with associated individual container status monitoring and Hence there is tri-angular communication occurs where source moves the container to destination and send the events parameter to monitoring station. After the successful transfer, the destination sends the destination events (import, start, re-start) to the same monitoring station where the information shall be mapped with the source events with the help of unique container ID.

5.1 Conceptual Framework

The CoMMon client monitoring probe, installed on the docker host, collects the container status (CPU, Memory utilization, Network Bandwidth) using docker stats command and converts the parameter values into JSON format and periodically send this information to the monitoring station. Similarly while migration, the probe sends another category of message (DateTime, ContainerID, Container Description, Source IP, Destination IP, Status, ContainerSize) to the station. The destination docker host also has the same monitoring probe which sends the same message with receive status to the station once it successfully receive and run the container. Figure 2 outlines the approach. The collected parameters passed into the monitoring station are stored into database and visualized in real time graphics display.

Source, destination and monitoring stations are three separate entities shall be located independently far away with each other, however TCP connections is to be established over IPv6 based network while sending and receiving the data forms a triangular communication. Monitoring server continuously receive the data from the migration source and destination, store the collected monitoring data to the database and visualize those data as graphs in the web which shall be accessible from anywhere. For the migration monitoring of number of containers from multiple cloud sources to multiple destinations, then it creates a complex communication network like in the Figure 3.

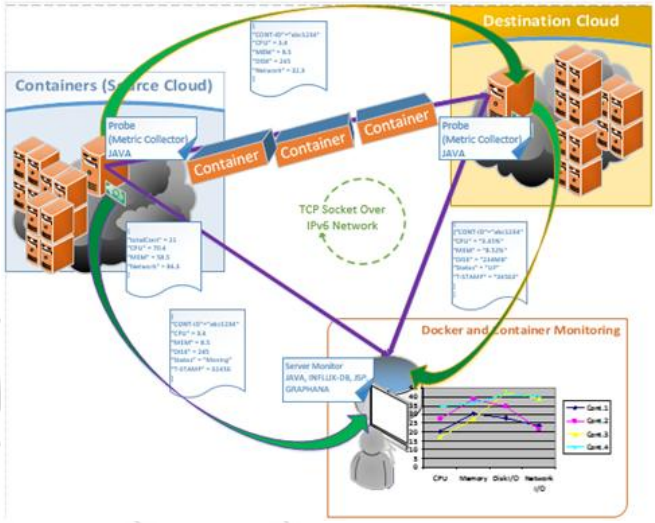

Figure 2: Tri-Angular container migration surveillance (Monitoring) system

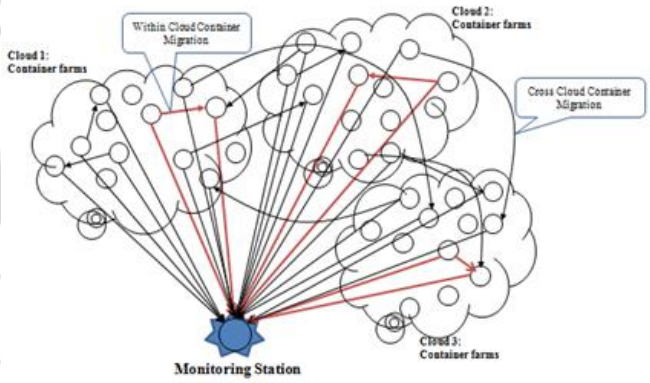

Figure 3: Container migration in the multi-cloud environment

Considering the migration of containers from multiple sources to multiple destinations, we need to measure the capacity of monitoring server on behalf of load balancing. We also need to calculate the status of source and destination hosts based on the performance metrics to take decision for migration. Considering the utilizations based on the containers only and also considering the equal virtual channel capacity for throughput to each container, the following nomenclature:

Source Host CPU Utilization, C_u^s	Container CPU Utilization, c_u^s
Source Host Memory Utilization, M_u^s	Container Memory Utilization, m_u^s
Source Host Disk Utilization, D_u^s	Container Disk Utilization, d_u^s
Source Host Network Utilization, T_u^s	Container Network Utilization, t_u^s
Destination Host CPU Utilization, C_u^d	Container CPU Utilization, c_u^d
Destination Host Memory Utilization, M_u^d	Container Memory Utilization, m_u^d
Destination Host Disk Utilization, D_u^d	Container Disk Utilization, d_u^d
Destination Host Network Utilization, T_u^d	Container Network Utilization, t_u^d

The corresponding metric values at the source and destination docker host can be calculated as the sum of resources consumed by the individual Container as:

$$C_u^s = \sum_{i=1}^{m} c_u^s[i], \quad C_u^d = \sum_{i=1}^{m} c_u^d[i] \quad \ldots\ldots (1)$$

$$M_u^s = \sum_{i=1}^{m} m_u^s[i], \quad M_u^d = \sum_{i=1}^{m} m_u^d[i] \quad \ldots\ldots (2)$$

$$D_u^s = \sum_{i=1}^{m} d_u^s[i], \quad D_u^d = \sum_{i=1}^{m} d_u^d[i] \quad \ldots\ldots (3)$$

$$T_u^s = \sum_{i=1}^{m} t_u^s[i], \quad T_u^d = \sum_{i=1}^{m} t_u^d[i] \quad \ldots\ldots (4)$$

S_i is the source hosts from where containers are to be migrated, where $1 \leq i \leq M$

D_j is the destination hosts where the containers are to be received, where $1 \leq j \leq N$

Every migration requires sending two status messages to the server from source and destination both followed by host status messages regularly monitored. Hence for n migrations, it requires 2n messages to be received and processed by the monitoring server. Similarly for M hosts each having n containers shall have maximum of 2(M x n) messages on the network during migration. The number of messages flowing in the channel helps to estimate the traffic congestion within the network.

The implementation of the system shall be categorized into two (i) service provider oriented surveillance system and (ii) service user oriented surveillance system. To meet the required service level agreement with the customers, if the running containers make the docker host overloaded like defined monitoring parameters consumed high resources, and then service provider shall take a decision to move those specific containers to other idle hosts within the cloud. Customers, who require only the infrastructure services, shall request any service provider to host their container from the provider's network. If the service provided is not satisfactory, then customer shall move their container to another cloud service provider to achieve better performance. In such case, customer shall independently use this monitoring service.

6. Implementation and Evaluation

We implemented our approach with JAVA by installing Docker on Ubuntu 14.04 virtual machines and container is configured as web server over IPv6 network environment. The software module basically is designed for CoMMon Probe and CoMMon Server. The probe performed the tasks of collecting monitoring metrics data. It uses the "docker stats" [26], the command line tool to display current status of containers installed on the docker hosts. The status information extracted every 15 seconds period converted into JSON format and sent to remote monitor. The probe also collects data while moving the Containers and sends this information to server. The pseudo codes for CoMMon Probe and server are depicted in Figure 4 & 5. Server module consists of monitoring manager which receive the data into JSON format and stored into mysql database. The analytical processing logic within the manager incorporates Java Script Charts provided by AMCharts [27]. However the preliminary test is implemented with mysql as a storage engine, we planned to implement with time series database like InfluxDB to properly visualize the real time data. InfluxDB [28] is the distributed scalable time series database having no any dependencies. It has REST HTTP API which directly receives the JSON data with SQL like query language and store at different points.

```
//CoMMon Approach at source and destination cloud
client_probe()
Define the list of variables for host status monitoring
/*  Host_IP, Cont_ID[], CPU_Utilization,
    Memory_Utilization, Network_IO */
Define the list of variables for migration monitoring
/*  Cont_ID, Cont_Name, Src_IP=Host_IP, Dst_IP,
    Cont_Size, Cont_Status /UP-Running,Migrating,Down */
while(TRUE){
        if(migrationMonitoring){ //check for category of data collection
            getMigrationMonitoring(); //collects migration monitoring variables
            encodeJSON(ListofParameters);
            //converts the collected parameters values into JSON format
            sendRemote(JSON_data); //send the data to remote monitoring station
            }
        else{
            getStatusMonitoring(); //collects host status monitoring variables
            encodeJSON(ListofParameters);
            //converts the collected parameters values into JSON format
            sendRemote(JSON_data); ////send the data to remote monitoring station
        }
    }
wait(Interval_Time);
} //end while
```

Figure 4: Pseudo code for Client Probe

```
//CoMMon Approach
    Server_Monitoring_Manager()
    registerContainers();
    while(TRUE){
        listenSocket(IPv6:port);
        //JAVA multi-threading for remote data processing
        verifyContainers()
        //Monitoring of registered containers only
        storeTDB();
        //insert received data to database
    }

//real time graphs visualization
    loadJSChart();
    //JS chart supported by AMChart
    connectDB();
    while(TRUE){
        plotGraphs();
        //plot for different metrics & status
        wait(Interval_Time);
    }
```

Figure 5: Pseudo code for Manager

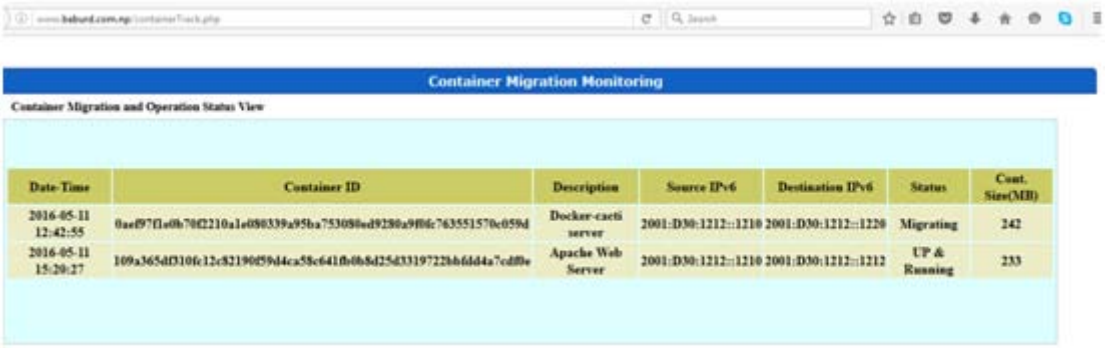

Figure 6: Status view of Container migration

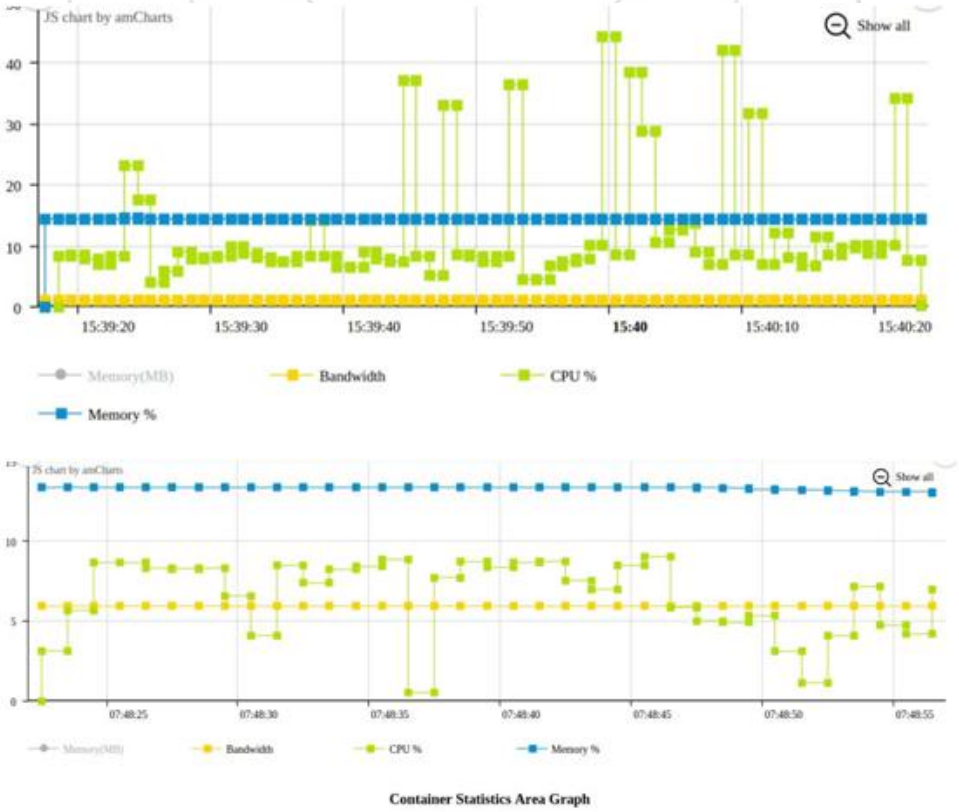

Figure 7: Container statistics in the source host at different time frame

Figure 8: Container statistics after migration in the destination host

The total CPU, Memory and Disk utilization of a host can't be measured from the container parameters only, however if a host only runs the containers, then the cornel libraries, system processes shall have fix amount of utilization. However this amount shall be of different values with different host. The graph shows the different pattern of utilization in the source and destination. This research is focused on total utilization based on containers that ultimately take decision for the migration.

7. Conclusion and Future Work

Cloud service providers have to meet the minimum quality of their services to customers as per service level agreement. The proposed monitoring system "*CoMMon*" provides the complete solution for service providers and customers to monitor the applications. Distance matters between service user clients and the server towards the variances of latency and jitter including the application's resources utilization factors. Either clients shall take a decision to move their containers or service providers shall move the containers transparently within its network. For both of the cases, our monitoring approach monitors the container migration events with its resource utilization status.

Several challenges like (i) Approaches for secure and efficient container migration (ii) Scalable distributed monitoring for load balancing with the increase of number of containers (iii) Clustering of the containers based on their resource utilizations for proper migration planning are the area of further research to be considered.

References

[1] Docker Website, https://www.docker.com/
[2] Yunshui L., Yule D, Junxiao G.(2013), "Real-Time Dynamic Cloud Monitoring System Based On SLA", International Journal of Automation and Power Engineering (IJAPE) Volume 2 Issue 4.
[3] https://opensource.com/resources/what-docker/26-02-2016
[4] Felter W., Ferreira A., Rajamony R., Rubio J., "An Updated Performance Comparison of Virtual Machines and Linux Containers", IBM Research Report, RC25482 (AUS1407-001) July 21, 2014, USA
[5] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, and Anthony Liguori (2007), "KVM: the Linux virtual machine monitor". In the Proceedings of the Linux Symposium, volume 1, pages 225–230, Ottawa, Ontario, Canada.
[6] McDougall R. and Anderson J.: "Virtualization performance: Perspectives and challenges ahead". SIGOPS Oper. Syst. Rev. 44(4):40–56, December 2010.
[7] ENTICE, Horizon2020 Project, http://www.entice-project.eu/
[8] K.-K. Lau, W. Lamersdorf, and E. Pimentel (ESOCC 2013), *LNCS 8135*, pp. 188–195, Springer-Verlag Berlin Heidelberg.
[9] G. Aceto et al. (2013), "Cloud monitoring: A survey", Computer Networks 57, 2093–2115
[10] https://www.openstack.org/summit/tokyo-2015/videos/presentation/monitoring-docker-container-and-dockerized-applications.
[11] http://devops.com/2014/11/24/docker-vs-vms/
[12] http://axibase.com/docker-monitoring/
[13] Data dog: dynamic infrastructure monitoring tool, https://www.datadoghq.com/
[14] Zabix: The Enterprise-Class Open Source Monitoring for Network and Applications, http://www.zabbix.com
[15] Sysdig Cloud: Infrastructure and Application Monitoring for Container, https://sysdig.com/
[16] Zhao, Z. et all.(2015), "Developing and operating time critical applications in clouds: the state of the art and the SWITCH approach", In the proceedings of HOLACONF – Cloud Forward: From Distributed to Complete Computing, Pisa, Italy. Elsevier, Procedia Computer Science, Vol (68) page (17-28)
[17] Armstrong T.(2001): "Netcat - The TCP/IP Swiss Army Knife", SANS Institute.
[18] CAdvisor, https://github.com/google/cadvisor
[19] Scout Monitoring, https://scoutapp.com/
[20] Sensu Monitoring, https://sensuapp.org/
[21] Docker-sensu-server, https://hub.docker.com/r/hiroakis/docker-sensu-server/
[22] https://prometheus.io/docs/introduction/overview/
[23] Logentries: Log Management & Monitoring System, https://logentries.com/
[24] Ruxit: All-in-one monitoring for cloud natives, https://ruxit.com

[25] Sematext, https://sematext.com/spm/integrations/docker-monitoring/
[26] Sensu-Docker, https://sematext.com/blog/2015/06/24/docker-events-and-metrics-monitoring/
[27] Docker Stats API, https://blog.logentries.com/2015/02/what-is-the-docker-stats-api/
[28] AMCHARTS, https://www.amcharts.com/
[29] Leighton, B., Cox, S. J., Car, N. J., Stenson, M. P., Vleeshouwer, J., & Hodge, J. (2015). "A Best of Both Worlds Approach to Complex, Efficient, Time Series Data Delivery in Environmental Software Systems Infrastructures, Services and Applications" (pp. 371-379). Springer International Publishing

Author Profiles

Babu Ram Dawadi Received his B.Sc. in Computer Engineering, M.Sc. in Information and Communication Engineering and Masters in Public Administration degree from Tribhuvan University. He worked as a system & network engineer and lecturer at Pulchowk Campus, TU for 5 years and Assistant Director for 3 years at Nepal Telecommunications Authority. Currently, he is working as a lecturer at department of electronics and computer engineering, IOE Pulchowk Campus. His area of interest is Networking, Distributed Computing and Data Mining.

Rajendra Paudyal received his BE in Electronics and Communication Engineering in 2010 from Pokhara University, Nepal Engineering College, Nepal. He worked as an Assistant engineer for 2 years at Ministry of Information and Communication. Currently, he is working as a Network Engineer at Nepal Telecom. His area of interest is networking, network measurement & traffic analysis and virtualization.