Native Cloud Support for Running WSO2 Middleware on Microsoft Azure

Rathnayake O. N [1], Dumendra W.A.S [2], Jayaratne V.V.M [3], Barthelot A.J [4], Shashika Lokuliyana[5]

Department of Computer Systems and Networking, Sri Lanka Institute of Information Technology (SLIIT), Malabe, Sri Lanka.

[1] osuranew@gmail.com, [2] sahan.wickramaarachchi@gmail.com , [3] vish120292@gmail.com ,
[4] allan.barthelot@gmail.com, [5]shashika.l@sliit.lk

***Abstract* - WSO2 Carbon is the core platform on which WSO2 middleware products are built. WSO2 products can be deployed in on premise servers as well as in cloud environments such as Microsoft Azure and Amazon Web Services (AWS). Apache based WSO2 products uses membership schemes to manage the members in their clusters. Addition and removal of members in a cluster and maintaining the consistency of the members list is the main objective of a membership scheme. Well-known address based membership scheme and multicast membership schemes are in-built membership schemes supported by Apache axis2, which are most suitable for in house server clusters. Apart from them, Kubernetes and AWS membership schemes are specially developed for cloud based deployments, for the cloud platforms Google Kubernetes and AWS respectively. The authors focuses on deployment of WSO2 product clusters in Microsoft Azure including a carbon membership scheme that uses Azure native features to automatically discover members. This new membership scheme guarantees that members in a WSO2 product cluster is managed smoothly. In addition, the authors focus on development of load balancing, auto healing and centralized logging using native features provided by Azure.**

Keywords – Membership Scheme, WSO2 Custer, Azure REST API, Auto Discovery, Clustering in Azure.

I. Introduction

WSO2 Carbon is an open source platform developed for Enterprise level WSO2 middleware products. The WSO2 Carbon core platform features a set of middleware components that together produce capabilities such as security, clustering, logging, statistics, management and more. These are basic features offered by all WSO2 products developed on top of the base platform [1]. The main focus of this research is the membership scheme which facilitates auto discovery of members in a WSO2 cluster deployed in Azure. Apart from that, dynamic load balancing, auto healing, centralized logging and are also provided as sub components.

A. Membership Scheme (auto discovery)

WSO2 products are based on Apache Axis2 web services framework. Hence WSO2 products use features inherited and derived from Apache Axis2. In Apache Axis2, membership schemes are used to manage the members in a cluster. Managing members includes notification of existing members when a new member is added and removed. Furthermore the membership scheme holds a list of members in a given cluster. These membership schemes are pluggable thus provides the flexibility to add a membership scheme of choice. By default, there are two membership schemes, well-known address based (WKA) membership scheme and multicast membership scheme, shipped with any WSO2 product. These in-built membership schemes can be deployed in an on premise server cluster. In addition, AWS membership scheme and Kubernetes membership scheme are developed specially in order to work in respective cloud environments. Even though WKA and multicast membership schemes work perfectly fine in those cloud platforms, yet the customized membership schemes provides feature-rich and efficient functionalities. Apart from that, such built-in membership schemes are of limited features and not fully flexible and very importantly do not make use of native cloud support. For an example multicast membership scheme can be used only in an environment where multicasting is enabled.

Carbon membership scheme for Azure is a fully functioning membership scheme which uses native cloud support provided by Azure. The main focus of this membership scheme is auto discovery of members, which is not supported by any of the existing built-in membership schemes. Unlike built-in membership schemes, auto discovery feature majorly reduces the configuration time by allowing the cluster to get its own list of members by itself. Apart from that, this new membership scheme lets users to make dynamic changes to the IP addresses of members.

B. Dynamic Load Balancing

Azure features consist of the option called Internet facing load balancer. It maps the public IP and the port number of incoming traffic to the private IP address and port number of the virtual machine set. Authors have configured the load balancing rules to use round robin method as it is one of the best and the simplest methods for distributing client requests across a groups of virtual machines [2]. Round Robin method does not always result the most accurate or most efficient distribution of traffic, but to implement more complex algorithm, the algorithm itself will consume a considerable amount of resources. Hence authors have decided to go ahead with round robin method. Previously, Load balancing part was handled by the software called Nginx [3]. Nginx has few issues such as, process spawning, concurrent request handling, Decoupling actual work requests etc. Authors have used the native functions and APIs in Azure platform to overcome this issue, since the underlying infrastructure is catered from the Azure Platform itself.

C. Centralized Logging

Microsoft Azure provides an interface for centralized logging. All it requires is the file path of the location of log files in local storage. However, as the requirement specified, there is no actual way of collecting or manipulating log files according to any sort of custom requirement. Log4J provides a lot of options for consolidating system logs into specific locations, according to log level, and similar filtering. However, on its own, it cannot solve this problem. A solution was devised to use a script and Cron mechanism to consolidate all logs to a single, isolated location that could be later accessed.

D. Auto Healing

Auto healing refers to getting a downed server or a virtual machine back on the business. It is happened in a few steps, firstly it checks for the health status of the interested server or the virtual machine and then according to the result returned it takes proceed the next step. If it is up and running then no action is taken otherwise it will be restarted. Cron tab and shell scripts are used in the implementation. Health checking of the servers and virtual machines are done in periods of 10 minutes and that duration can be adjusted as per the requirement of the end user.

II. Literature survey

Research carried out on scalability and multi tenancy of WSO2 products has laid the ground work for this research. Scalability is an important requirement in distributed environments and solutions to provide scalable systems can be developed at different levels [4] [5]. [6] Provide an overview of how cloud computing can support the development of scalable applications. In [7], they have researched on how scalability can be done on WSO2 carbon middleware. As explained in [7], most of the PaaS vendors including WSO2, deploy a single platform supporting clustering and is shared by all tenants. Multiple instances of WSO2 products such as WSO2 application server, WSO2 Enterprise Service Bus can be deployed on different machines across the cluster. Therefore this supports a scalability solution that guarantees the performance levels of the platform by dynamically scaling out/scaling down the resources assigned to the cluster. This functionality is achieved by adding or removing servers and many other resources to the cluster. Furthermore this mechanism manages shared resources preserving security and isolation among tenants. In addition, it implements a set of functionalities including load balancing, PaaS monitoring, and elastic auto-scaling. Besides that this mechanism provides high resource utilization, since the machines in the cluster are shared among different tenants, and some scalability, as system resources can be incrementally extended. Finally, it promises high availability and reliability due to the introduced redundancy.

Multi-tenancy implies that applications should be designed so that they maximize the resource sharing between multiple consumers. Thus, service providers are able to maximize the resource utilization and as a result reduce their servicing costs per customer [8]. Multi-tenancy is a main prerequisite to enable other very important characteristics of applications or services like isolation, configurability and scalability [9], [10], and [11]. In [12] the authors introduce an architecture for a multi-tenant middleware platform, called WSO2 Carbon that enables users to run their services and furthermore provides them an environment to build multi-tenant applications. WSO2 Carbon provides multi-tenancy support by appropriate adaptations to its underlying execution engine Apache Axis2. On top of WSO2 Carbon they support multi-tenancy at the ESB and SCE [13] level.

In the membership scheme perspective, there are few membership schemes already in use. Their capabilities and limitations are illustrated in table 1.

TABLE I. COMPARISON OF EXISTING MEMBERSHIP SCHEMES

Multicast	**WKA**	**AWS**
All nodes should be in the same subnet	Nodes can be in different networks	Amazon EC2 nodes
All nodes should be in the same multicast domain	No multicasting requirement	No multicasting requirement
Multicasting should not be blocked	No multicasting requirement	No multicasting requirement
No fixed IP addresses or hosts required	At least one well-known IP address or host required	No fixed IP addresses or hosts required
Failure of any member does not affect	New members can join with some WKA nodes down,	Failure of any member does not affect

membership discovery	but not if all WKA nodes are down	membership discovery
Does not work on IaaSs such as Amazon EC2	IaaS-friendly	Works on Amazon EC2
No WKA requirement	Requires keep-alive, elastic IPs, or some other mechanism for re-mapping IP addresses of WK members in cases of failure	No WKA requirement

By conducting a basic analysis of the prevailing membership schemes which can be used in Azure could platform, some features were identified, which can be improved in order to increase the efficiency and performance. The main identified problem is that, well-known address based and multicast membership schemes are fully independent from the underlying Azure cloud platform, as they are generic membership schemes. Making use of native cloud support provided by Azure, will improve the efficiency and performance of the membership scheme.

In addition to the efficiency and performance parameters, it can also benefit the improvements mentioned in below table. Apart from that, it also can be customized to make use of native special features provided by Azure. Table 2 shows issues that were identified in existing membership schemes.

TABLE II. ISSUES IN EXISTING MEMBERSHIP SCHEMES

Issue	**Description**
Dynamic IP Address changes	Well-known address based or multicast membership schemes do not provide features for IP addresses to be changed or assigned dynamically. If an IP address of a member was changed, it has to be changed accordingly in the other members as well.
Hassle in configuration	IP addresses of the members should be configured manually
Multicast issues	AWS and Azure restrict multitasking.

Apart from above issues, the authors have found some other issues existing in WKA when it is deployed in Azure, which definitely need some alteration. Those issues are illustrated in following scenarios.

Scenario 1: Well-known node instance being shut down.

If the well-known node goes down, other nodes will be treated as individual nodes running in separate clusters. If this happens, the whole clustering concept would be broken and re-implementation will be needed.

Solution: This issue can be overcome by introducing more well-known members in the cluster. Even in this solution, many efficiency issues can be seen. Being one of them, having many well-known nodes in places makes the production costly, that is, it needs more resources and configuration.

Scenario 2: Well-known node VM rebooting

If the VM in which the membership instance is deployed gets rebooted, then the rest of the nodes will lose the members list and will act as in different clusters.

Solution: As the previous scenario, this issue also can be overcome by introducing more well-known members in the cluster. Even in this solution, many more efficiency issues can be seen. Being one of them, having many well-known nodes in places makes the production costly, that is, it needs more resources and configuration.

III. METHODOLOGY

A. *Membership Scheme*

Membership scheme is a pluggable software component which runs with any product that is based upon WSO2 Carbon middleware. As a pre-configuration, user has to create either a network security group or a tag and assign respective virtual machines to that. This network security groups or tag is used to identify a group of virtual machines. Thereafter an API call is made to Azure REST API, with the network security group name or tag, to retrieve the list of virtual machines (refer figure 1). In the response, a list of network interface names can be seen. Afterwards another few API calls are made to get the IP addresses associated with each network interface name (refer figure 2). In this step. Number of API calls made is the number of network interface names retrieved in the previous API call and ultimately that is the number of virtual machines (members or nodes) in the cluster. Once the list of IP addresses is obtained, it is updated with the Hazelcast configuration instance. Hazelcast configuration instance will then be distributed among members and that is done by Carbon middleware.

Once the above scenario is completed, the WSO2 product in interest can be started. While it is starting up, log messages can be seen as and when members added, joined and left the cluster. Table 3 shows the detailed description of all the log messages issued by the membership scheme.

TABLE III. LOG MESSAGES

Log message	Implication
Member added	This log message can be seen when the virtual machine where the WSO2 product installed is added to the cluster.
Member joined	This log message can be seen when the instance of a WSO2 product is started up
Member left	This log message can be seen when an instance was shut down

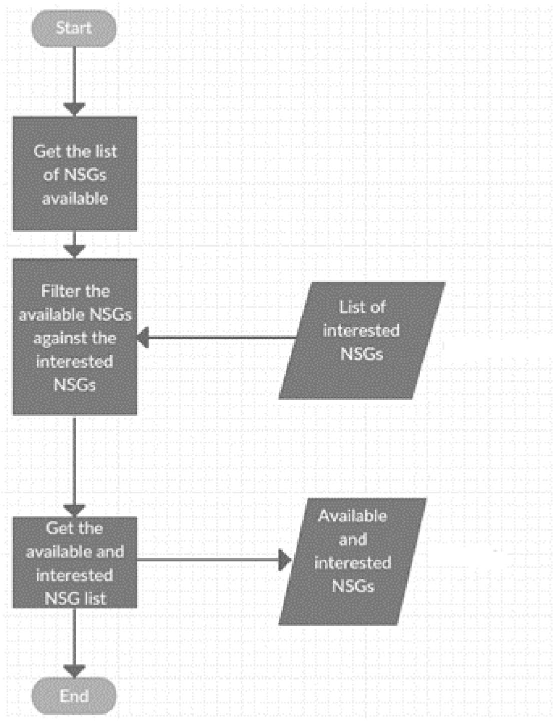

Start
Get the list of NSGs available
Filter the available NSGs against the interested NSGs
List of interested NSGs
Get the available and interested NSG list
Available and interested NSGs
End

Figure 8. Flow chart of network interfaces retrieval

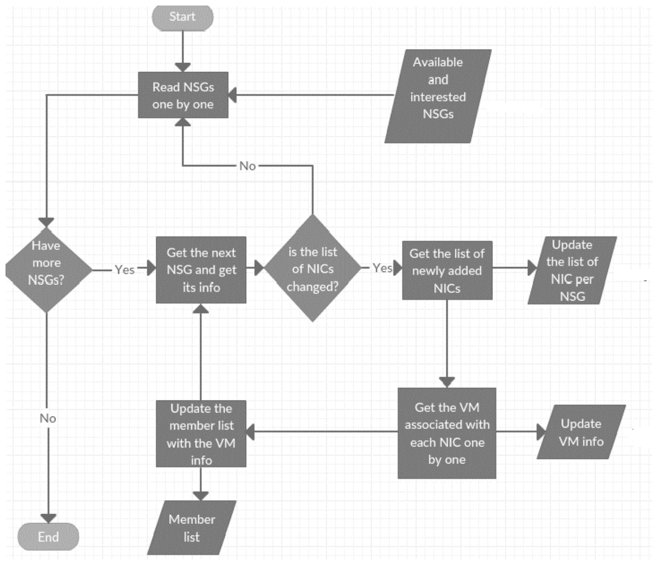

Start
Read NSGs one by one
Available and interested NSGs
No
Have more NSGs?
Yes
Get the next NSG and get its info
is the list of NICs changed?
Yes
Get the list of newly added NICs
Update the list of NIC per NSG
No
Update the member list with the VM info
Get the VM associated with each NIC one by one
Update VM info
End
Member list

Figure 9. Flow chart of IP address retrieval

B. Dynamic Load Balancing

Azure platform supports heterogeneous workloads consisting of different requests and transaction types. Authors concentration is on the performance, As the dynamic load balancing mechanism is handled in software platform level. There is an entity called backend Azure platform, and all the VMs are attached and managed by the backend pool. Once it is configured to handle the load balancing in Azure platform, it uses the Round Robin, as authors have already defined.

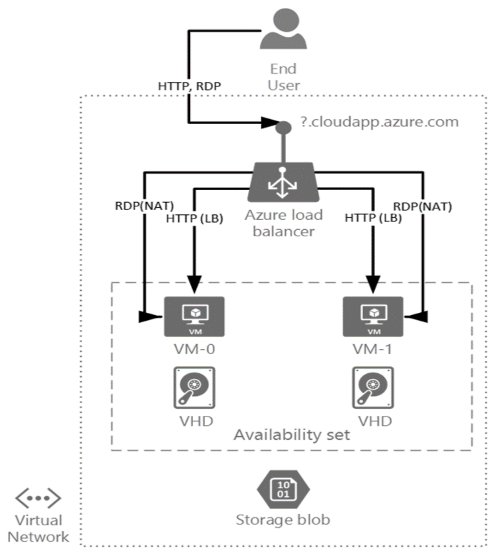


Figure 10. Overview of load balancing

Ultimately what happens is HTTP redirection to the particular set of VMs (refer figure3). Backend is the centralized dispatcher and it receives all incoming HTTP requests and distributes among the VM nodes. Unlike most dispatcher based solution, the HTTP redirection does not require additional modifications of IP addresses, it accepts all the requests from the public IP and redirect it to the private IP addresses (refere figure 4).

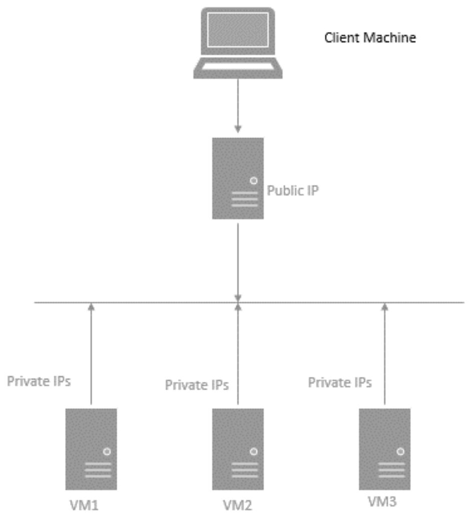


Figure 11. High Level Network Diagram

C. *Centralized Logging*

Logging is implemented using Log4J. Logs at each level are put into different files, and can be configured independently. The solution was devised to use a script and cron mechanism to consolidate all logs to a single, isolated location that could be later accessed (refer figure 5).

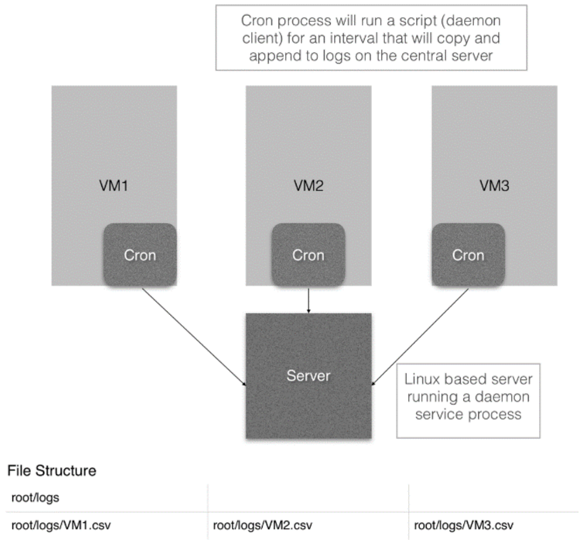
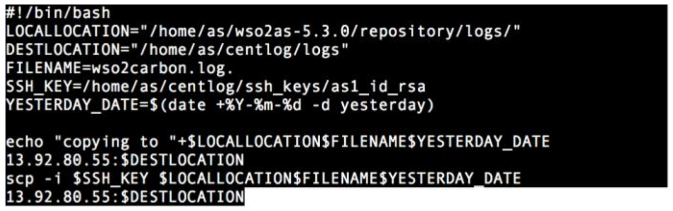

Cron process will run a script (daemon client) for an interval that will copy and append to logs on the central server
VM1
VM2
VM3
Cron
Cron
Cron
Server
Linux based server running a daemon service process
File Structure
root/logs
root/logs/VM1.csv
root/logs/VM2.csv
root/logs/VM3.csv

Figure 12. Block diagram of centralized logging functionality

A script is written to,

- Identify the most recent log file (log file with the date of the previous day)
- Copy this file to a pre defined central location on another server

The copy operation is done using a ssh key mechanism so that the remote location can be accessed securely.



```
#!/bin/bash
LOCALLOCATION="/home/as/wso2as-5.3.0/repository/logs/"
DESTLOCATION="/home/as/centlog/logs"
FILENAME=wso2carbon.log.
SSH_KEY=/home/as/centlog/ssh_keys/as1_id_rsa
YESTERDAY_DATE=$(date +%Y-%m-%d -d yesterday)

echo "copying to "+$LOCALLOCATION$FILENAME$YESTERDAY_DATE
13.92.80.55:$DESTLOCATION
scp -i $SSH_KEY $LOCALLOCATION$FILENAME$YESTERDAY_DATE
13.92.80.55:$DESTLOCATION
```

Figure 13. Bash script for log consolidating

This script (figure 6) is then put in a cron job that will execute once a day, at a time after a new log file is created, and the old one is archived, with the date appended to the file name. This file will have the date of the previous day. This will make sure that the script will copy the right file. With all these functions implemented and deployed, centralized logging functions successfully.

D. *Auto Healing*

Auto healing is achieved in 2 approaches.

- Service level
- Virtual machine level

In the service level auto healing, an applications running inside the virtual machine is healed in case of a sudden service shut down and is handled by a shell script with a cron tab. Periodic checks of the state of the application will ensure that its downtime is minimal. Parameters required to run this shell script is the Service Name (name of the WSO2 product with location). Once the Service name is given, shell script will periodically check for any running instances of the application and restart if it is down.

In virtual machine level auto healing, status of the virtual machine is examoined by using Azure REST API calls [14]. The response would contain the status of the virtual machine is interest. Then if the virtual machine is powered down, using another Azure REST API call [15], it can be started again.

IV. RESULTS AND DISCUSSION

Number of results were taken to test the performance of new membership scheme. First one being, time taken to start the server were obtained from a server having WKA membership scheme and the new membership scheme for Azure. In the setup 6 virtual machines were deployed as 2 clusters each having 3 virtual machines in total. First cluster had WSO2 Application Servers installed in each of its virtual machines along with the WKA membership scheme. Second cluster had the same setup with new membership scheme plugged in. Authors started up each server 5 times in order to take an average out of them. There were few other constraints and dependencies that the time taken for a server to start up relied upon [16].

- Database connectivity
- Used memory
- Available memory
- CPU usage
- Internet speed

Countermeasures were taken to make sure that above constraints were unchanged from one result to another. Each time the virtual machines having the server installed were rebooted to gain a trustworthy constant value for used memory, available memory and CPU usage. Internet speed was beyond authors control as it is provided by the Azure platform. Therefore an assumption was made that all the virtual machines have the same internet speed as they are in the same region of Azure and sharing the same Azure subscription. Database connectivity made from each virtual machine to the virtual machine which has the MySQL instance were reset each time. Following are the results

1. Servers with WKA membership scheme

TABLE IV. TIME TAKEN FOR A SERVER STARTUP FOR WKA MEMBERSHIP SCHEME

Occurrence no	Time taken to start up (seconds)		
	VM1	VM2	VM3
01	130.8	123.6	126
02	131.4	130.2	132.6
03	120.6	125.4	129.0
04	123.6	127.2	127.2
05	129.0	129.6	120.0

Mean values for each virtual machine across the 5 occurrences were then calculated as follows,

Virtual machine 1 = 127.08 seconds
Virtual machine 2 = 127.2 seconds
Virtual machine 3 = 126.96 seconds
Mean value out of all the virtual machines = 127.08 seconds

2. Server with the new membership scheme for Azure

TABLE V. TIME TAKEN FOR A SERVER STARTUP FOR AZURE MEMBERSHIP SCHEME

Occurrence no	Time taken to start up (seconds)		
	VM1	VM2	VM3
01	56	56	55
02	58	56	61
03	61	59	58
04	55	69	64
05	60	62	57

Mean values for each virtual machine across the 5 occurrences were then calculated as follows,

Virtual machine 1 = 58.0 seconds
Virtual machine 2 = 60.4 seconds
Virtual machine 3 = 59.0 seconds
Mean value out of all the virtual machines = 59.1 seconds

When compared the duration of time in second that took for WKA membership scheme to start up with that of the new membership scheme, a short duration of time has been taken by the new membership scheme.

Authors have tested the load balancing function by firing 500 HTTP requests as in Nginx Testing [17] , Nginx has been fired 500 HTTP requests. Below are the results of turnaround time for both the tests. Since the HTTP requests is handled from the hardware layer, this solution has been able to perform better than the existing Nginx load balancer.

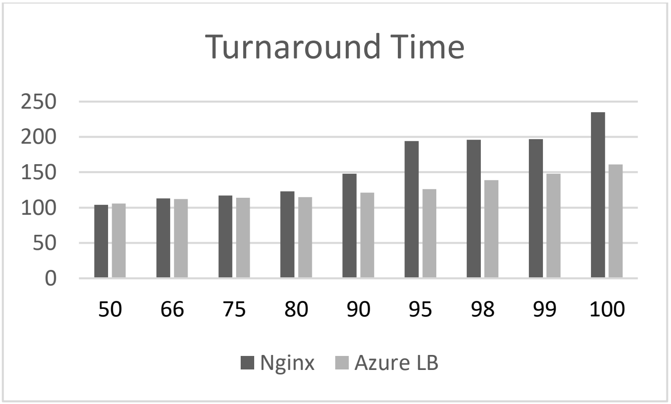


Figure 14. Dynamic load balancing test results

V. CONCLUSION AND FUTURE WORK

A few membership schemes are available for the management of members in a cluster. Even though it is the case, there was a lack of a membership scheme that uses Azure native cloud support. This new carbon membership scheme for Azure has overcome a few issues existed and introduce new features. Especially in case where well-known member goes down, WKA membership scheme has a very limited capability to get the cluster working back to normal condition. Azure membership scheme has eliminated the need of well-known members or special members and so is the problem.

Auto discovery of members is a new feature that enables the user to add and remove members without making changes to the other members. Network security groups and tags are used for the grouping purpose, in the implementation of this feature. Performance results have shown that a server with Azure membership scheme take a shorter startup time than that of a server which uses WKA membership scheme.

The coding of this membership scheme needs a touch of code optimization techniques in order to achieve even better performance results. Apart from that for the communication purpose with the Azure API, basic http requests has been used. Implementation of proper secured communication method or an API query system can be stated as future work. In addition, this membership scheme is suitable only for Azure Resource Manager (ARM) deployments. Hence another compatible version of this membership scheme for Azure Classic deployments also can be mentioned as future work.

ACKNOWLEDGEMENT

Authors would like to express their sincere gratitude to Mrs. Shashika Lokuliyana and Mr. Anuradha Jayakody for being the supervisor and co-supervisor, for their kind advices and guidance in different perspectives and directing them throughout the research. Authors would like to convey a special thanks to Mr. Lakmal Rupasinghe for highlighting the availability of the Native Cloud Support for Running WSO2 Middleware on Microsoft Azure. Lecturer in charge, Mr. Jayantha Amararachchi and other lecturers who provided their immense support also acknowledged. Last but not least, many thanks to the family and friends of the authors for their support over the months.

REFERENCES

[1] "WSO2 platform," WSO2, 2016. [Online]. Available: http://wso2.com/platform. [Accessed 03 October 2016].

[2] S. Akl, "The Design and Analysis of Parallel Algorithms," Prentice-Hall, Englewood, 1989.

[3] "Nginx," Nginx, 2016. [Online]. Available: https://www.nginx.com/. [Accessed 30 September 2016].

[4] M. Pati˜no-Martinez, B. Kemme, F. Perez-Sorrosal, and D. Serrano, "A system of architectural patterns scalable, consistent and highly available multi-tier serviceoriented infrastructures," *Architecting Dependable Systems,* vol. LNCS 5835, 2009.

[5] J. S¨oderlund, "Scalability patterns and an interesting story," August 2010. [Online]. Available: http://thebigsoftwareblog.blogspot.com/2010/scalability-fundamentals-and.html,. [Accessed 05 2016].

[6] J. Caceres, L. Vaquero, A. P. L. Rodero-Merino, and J. Hierro,, "Service scalability over the cloud," *Handbook of Cloud Computing,* 2010.

[7] Claudio A. Ardagna, Ernesto Damiani, Fulvio Frati, and Davide Rebeccani, "Scalability Patterns for Platform-as-a-Service," in *IEEE Fifth International Conference on Cloud Computing*, 2012.

[8] S. Strauch, V. Andrikopoulos, F. Leymann and D. Muhler, "ESBMT: Enabling Multi-Tenancy in Enterprise Service Buses," *Proceedings of CloudCom*, pp. 456-463, 2012.

[9] F. Chong and G. Carraro, "Architecture Strategies for Catching the Long Tail," 2006. [Online]. Available: http://msdn.microsoft.com/en-us/library/aa479069.aspx. [Accessed 04 2016].

[10] C. M. a. S. K. R. Krebs, "Architectural Concerns in Multitenant SaaS Applications," *CLOSER,* pp. 426-431, 2012.

[11] E. T. a. W. J. S. Walraven, "A Middleware Layer for Flexible and Cost-Efficient Multi-tenant Applications," *Springer,* pp. 370-389, 2011.

[12] A. Azeez, S. Perera, D. Gamage, R. Linton, P. Siriwardana, D. Leelaratne and S. Weerawarana, P. Fremantle, "“Multi-tenant SOA Middleware for Cloud Computing,” in Cloud computing (cloud)," in *ieee 3rd international conference on cloud,* 2010.

[13] S. P. I. K. a. S. W. M. Pathirage, "A Multi-tenant Architecture for Business Process Executions," in *IEEE International Conference on, 2011*, 2011.

[14] Microsoft Corperation, "Microsoft Azure," Microsoft Corperation, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/azure/mt163682.aspx. [Accessed 10 August 2016].

[15] Microsoft Corperation, "Microsoft Azure," Microsoft Corperation, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/azure/mt163628.aspx. [Accessed 10 August 2016].

[16] A. A. Deepal Jayasinghe, Apache Axis2 Web Services 2nd edition, 2011.

[17] [Online]. Available: https://docs.jelastic.com/testing-load-balancing. [Accessed 20 September 2016].