An Agent-based Services Framework with Adaptive Monitoring in Cloud Environments

Yi Wei and M. Brian Blake
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556
{ywei1, mblake3}@nd.edu

***Abstract*—Cloud computing environments provide flexible infrastructures for third-party management of organizations' information technology (IT) assets. With web services being a standard for realizing web-based business capabilities, the emergence of cloud computing will bring new challenges to different web service activities. In this paper, the authors propose an agent-based framework that provides effective integration of services within clouds. To tackle the dynamism in service operations, an adaptive monitoring algorithm is proposed. The algorithm is inspired by the congestion control approach from the TCP protocol and provides efficient, up-to-date information about service status without exhaustive monitoring. Experimental results show that the monitoring algorithm provides significant benefits when compared to the more exhaustive methods. This approach also facilitates other service activities, such as cross cloud service discovery.**

Keywords-web services, cloud computing, service lifecycle, service management, service discovery

I. INTRODUCTION

Service-Oriented Computing (SOC) principles and web services provide flexibility in Internet operations. Web services are loosely coupled, self-contained, and network enabled software components that realize a specific task [1]. With standards such as WSDL, SOAP, and REST, web services are widely adopted in domains such as e-commerce, scientific workflow, and distributed and embedded systems.

Cloud computing is a recent paradigm that builds on previous computing paradigms such as grid computing and utility computing [2]. Traditional computing capabilities such as CPU hour, storage capacity, network bandwidth and even software functionalities are flexibly configured and commoditized by cloud computing frameworks. Typically, for each cloud, there is a single cloud provider responsible for managing and maintaining the cloud resources. Multiple cloud users can request resources from the provider. These users can be cooperative, as in a company's private cloud, or mutually untrusted, as in a public cloud. The cloud provider provisions resources based on users' requests and availability of the requested resources.

As more and more companies are advertising their services on the web, effective service discovery and management has become a critical aspect of service-oriented computing. At the same time, as cloud computing grows in maturity and gains more acceptances, it will likely become a primary distributed computing paradigm in the future. Consequently, a natural strategic move for service providers is to offer services deployed in cloud computing frameworks [3]. Hence for service developers and users, providers' cloud settings will be the primary location for finding desired services. Examples of services that are already offered by cloud-based service providers include Customer Relationship Management (CRM) services, Enterprise Resource Planning (ERP) services and email exchange services.

The growing number of service activities in cloud environments requires a highly efficient framework that promotes service advertisement, management and discovery operations. A framework that is capable of scanning and incorporating service specifications across multiple cloud providers facilitates higher-level, cloud provider-situated knowledge bases that will, in effect, federate cloud environments with respect to their services. However, such an environment requires intelligent or agent-based capabilities to adapt to the highly-dynamic environment. In this paper, we introduce a supporting software framework for agent-based services in cloud environments, as well as an adaptive algorithm to monitor deployed services. The algorithm has the ability to adaptively adjust service status checking time intervals based on different check responses. This framework reduces the number of messages required to check the status of services (as shown in our simulation results).

The remainder of this paper is organized into the following sections. In Section II we discuss the challenges in service-cloud integration. We also presented a brief survey on related work in this section. In Section III, we introduce a framework for agent based services and an adaptive monitoring algorithm. A prototype implementation of the framework is described in Section IV. Experimental results and analysis are also presented in the same section. Conclusions and future work are described in Section VI.

II. CHALLENGES AND RELATED WORK

A. Challenges of Service-Cloud Integration

Web-based service repositories (such as Seekda [6] and ProgrammableWeb [7]) and service search engines (such as Woogle [4]) tend to leverage user efforts and/or web

1524-4547/12 $26.00 © 2012 IEEE
DOI 10.1109/WETICE.2012.20

crawlers to find potential web services available on the open Internet. However, the Web is a heterogeneous environment, so for different services, how it is deployed and what technology is used can vary greatly. Moreover, the availabilities of the discovered services are usually not identical and there is no monitoring mechanism for those services so that users can acquire the latest information about them. Thus, the unstructured, perhaps disorganized, nature of services published through these means poses a challenge for service advertisement, management and integration. The emergence of cloud environments provides a suitable location for more organized and more effective service management and utilization. However, in order to achieve this goal, several challenges must be addressed, such as (1) *what are best practices for modeling services in cloud settings*, (2) *how are deployed services managed/monitored efficiently* and (3) *what kind of software is needed to support the service-cloud integration.*

B. Related Work

Cloud computing has attracted much attention from both academia and industry. Armbrust et al [2] identify the opportunities and challenges brought by this new computing paradigm. Agents for inter-organizational interoperability have been investigated broadly [8][9]. Furthermore, several researchers have applied agent-based technologies to the cloud. In [10], the authors integrate Multi-Agent Systems (MASs) with a cloud environment for intelligent behaviors, while in [11] an agent-based testbed is proposed to bolster cloud resource discovery and service level agreement negotiation. A multi-agent architecture for QoS-assured cloud service delivery is described in [12]. J.O. Gutierrez-Garcia et al in [13] proposed a three-layered self-organizing multi-agent system that can deal with the dynamicity of service composition process in the cloud. In [14], the concept of agent-based cloud computing is proposed by the author. The work introduced this paradigm to aid the development of software tools for service operations in the cloud using agent-based cooperative techniques. And in [15] an agent-based dynamic resource allocation mechanism for federated clouds is introduced.

Different flavors of service monitoring mechanisms have been described in the literature. Barbon et al [16] proposed a solution that monitors both instances and classes of BPEL processes. Similarly, a monitoring framework targeting BPEL is also explained in [17] which utilize a specification language, while [18] focused on the monitoring of service QoS attributes. Moser [19] took a step further by not only monitoring the service process but also replacing them based on QoS requirements, and [20] leverages the monitoring results as the basis for service selection. Our proposed work differs from previous research in that we not only enable effective service activities in a single cloud, but also use the agent-based services as a means to connect multiple clouds.

III. A FRAMEWORK FOR AGENT-BASED SERVICES

A. Service Model Description

Currently, three major types of cloud computing delivery models exist. The *Infrastructure-as-a-Service (IaaS)* cloud provides users with raw computing resources like block drives where they can upload their virtual machines. The *Platform-as-a-Service (PaaS)* cloud offers various kinds of computing platforms as rentable resources which replace local servers and desktops. And the *Software-as-a-Service (SaaS)* cloud offers on-demand and longer-standing *functional* software service to its users. Our service model and framework target the IaaS type of clouds, because it is common that a service-oriented system is developed as a stand-alone application and the service providers often require a configurable virtualized platform so that they can install the required supporting software packages. In the following sections, we will simply use "**cloud**" to refer to "**IaaS cloud**".

We present here a service model that captures the information of services deployed on the same host as well as the information of the hosting environment. Figure 1. presents this model in a UML class diagram. Due to limitation of space, all methods are omitted. A service model stores the following information:

- Service provider's information, including its name, address and telephone number;
- Hosting environment information, including the IP address of the host and related information about this host, such as service container information or hardware configuration information;
- One or more service(s) deployed on this host.

A service contains the following attributes:

- Service name;
- Service description;
- Service endpoint reference;
- One or more operation(s).

An operation of a service defines the functionalities exposed by this service. An operation entity has the following information:

- Operation name;
- Operation description;
- Operation input message (zero or more);
- Operation output message (zero or more).

This model represents the fundamental information for communication in the proposed agent framework. It is worth mentioning that the service model here is an internal data model used by the systems agents, so *it is transparent to the client or end users*.

B. Agent-based Service Management Framework

Figure 2. presents the overarching architecture, in a multi-cloud scenario, of the framework for Service Activity Management in Clouds. The framework utilizes the aforementioned model and has two major components, namely extractor agent and aggregator agent. The extractor agent is responsible for processing services on each virtual machine and building models. To do that, it runs as a web

service and scans the service repository on a virtual machine (VM) to build service models. This step is automatic and does not require interventions from service developers or providers.

After extractor agents finish building models, they communicate with the aggregator agent running in the same cloud to register the services. The aggregator agent collects different service models across multiple VMs to form a large web service knowledgebase to facilitate service discovery, management and integration. By using the enhanced service model that contains detailed information about a web service, the aggregator agent can support more search schemes than traditional processes do.

With the aggregator agent managing web services on multiple VMs, we can further extend this structure to a cross cloud environment. Aggregator agents in different clouds can communicate with each other and exchange their service information. An aggregator agent can also forward user queries to its peers to see whether there are other services from other clouds that can better fulfill the user's requirements.

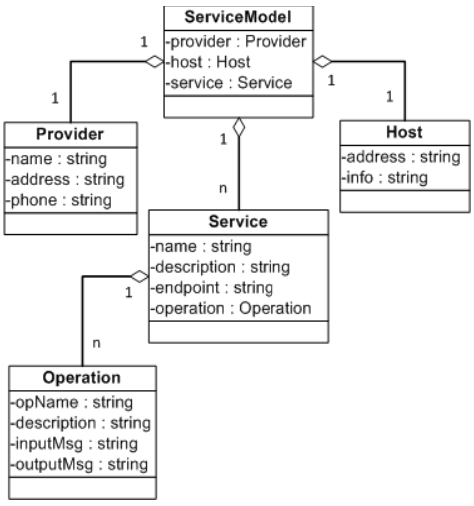

Figure 1. Service Model in UML Diagram

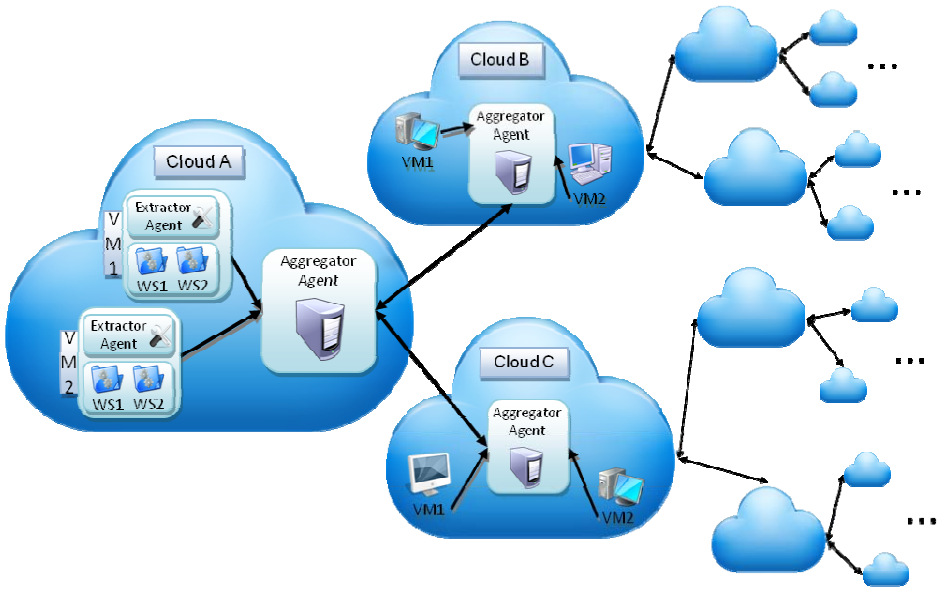

Figure 2. Overarching Architecture of the Framework in a Cross-Cloud Scenario

C. Adaptive Service Monitoring Algorithm

In order to facilitate the service management in single- or multiple-cloud environment, we need an efficient monitoring method. We proposed a service monitoring algorithm which leverages the idea of congestion control in TCP protocol. This algorithm adaptively governs the communication pattern of the agents. Instead of poking each extractor agent on a regular time interval, the aggregator agent checks each of them using an algorithm inspired by several characteristics in the TCP protocol suite, such as "*slow start*" and "*additive increase and multiplicative decrease (AIMD)*". The basic idea of the proposed algorithm is that the aggregator agent will prolong the checking periods of those consistently-available extractors so that it can issue a reduced amount of monitoring commands. We name this algorithm the *Check Period Relaxation* (CPR) algorithm.

The algorithm uses an initial value (set to 1s in our current implementation) as the first check period for a given extractor agent, say *A*. If *A* successfully responds to the first check, the aggregator agent knows that *A* is running correctly and doubles the checking period for the next check for *A*. The aggregator agent also marks *A* as in the *Fast Relax (FR)* stage. This check period doubling continues as long as *A* keeps responding to the checks and the checking period for *A* does not exceed a threshold, called the *Fast Relax Limit* (*FRL, default 32s*). After the check period for *A* exceeds FRL, *A* is put into another stage called the *Cautious Relax (CR)* stage. In this stage, the check period of *A* increases by 1

second after each successful check. The purpose of setting a cautious stage is that even a consistently-performing extractor agent can eventually discontinue its service. If the check period for *A* reaches another threshold, called the *Cautious Relax Limit* (*CRL, default 64s*), it will not increase and the aggregator agent will use this value as the following check period for *A* as long as *A* remains responsive. Detailed steps are listed below for the algorithm:

1) If a status check for *A* is successful and *A* is in the *FR* stage, the aggregator agent will double the current check period for *A*. If the new check period is larger than the *FRL*, the new check period will be set to *FRL* and *A* is changed to the *CR* state. Otherwise *A* remains on the *FR* state;

2) If a status check for *A* is successful and *A* is in the *CR* state but hasn't reached the *CRL*, the aggregator agent will increment the current check period for *A* by 1 second and use this value as the new check period. If *A* has reached the *CRL*, the aggregator agent will leave the check period for *A* unchanged and just use *CRL* as the next check period for *A*;

3) If a status check for *A* is unsuccessful, the aggregator agent will halve the current check period for *A* and, if *A* is not in *CR* list, changes *A*'s state to *CR*;

4) If *A* is at *CR* state but the check period for *A* is smaller than the *FRL* value, then after 3 consecutive successful status checks for *A*, the aggregator agent will change *A* back to the *FR* state and follow step (1);

5) If 3 consecutive status checks for *A* are all unsuccessful, the aggregator agent will mark *A* as inactive. All web services associated with *A* will also be labeled as inactive. After 3 minutes, the aggregator agent will make its final attempt to check *A*'s status. If it's still unsuccessful, *A* will be removed as well as all web services associated with *A*. If it's successful, *A* will be put back to the *CR* state, its check period reset to 1s and follow step 4).

A Finite State Automaton (FSA) showcasing the states and transitions of the CPR algorithm is shown in Figure 3.

The goal of the CPR algorithm is to reduce the number of monitoring commands issued by the aggregator agent. To evaluate whether it achieves this goal, we implemented a simulation environment to compare the performance between CPR and the exhaustive method. The simulation mimics an environment with 1 aggregator agent and 100 extractor agents. The simulated time is 1 day (24 hours) and the smallest time step in the simulation is 1 second. To evaluate the algorithm under different service availability levels, several sets of simulations, with services that have different availability probabilities, were run. The availability level is used as the check success probability in the simulation. For example, if a service has 90% availability, then in the simulation, for each check from the aggregator service to this service, the check has a 90% probability to succeed.

IV. Implementation and Evaluations

A. Prototype Framework Implementation

We have implemented the extractor agent and the aggregator agent of the proposed framework as Axis2 [5] services. The extractor agent queries the Axis engine to get a list of objects which present all the deployed web services. Then the extractor filters out inactive services and builds a service model for active ones using Axis APIs. After the model construction is done, the extractor agent initiates a communication to the aggregator agent to register itself with the latter. The aggregator agent keeps track of all the active extractor agents in the cloud and employs a monitoring algorithm developed by us to obtain the latest status information about each extractor agent.

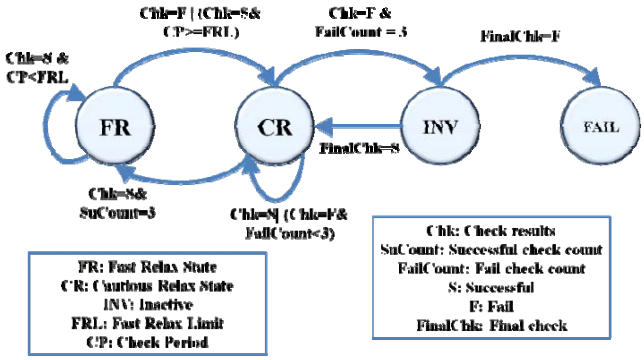

Figure 3. Finite State Automaton for CPR Algorithm

B. Evaluation of the CPR Algorithm

Two sets of experiments were conducted to evaluate the performance of CPR algorithm. In the first set shown in Figure 5. , all services have the same availability level in a single simulation execution. And services with 75% to 99% availabilities are simulated in different simulation runs. The "Regular" line in the diagram denotes the method that checks each extractor agent on a fixed time interval, while the "CPR_1e" line represents the CPR algorithm data. Based on the results, we have made two observations. The first is that the total number of messages generated by the CPR algorithm during the simulation is smaller than the exhaustive approach, especially when all services have a high level of availability. And the second observation is that the CPR algorithm may generate too many messages for services with relatively low availability. Those are the services that may occasionally fail for several checks, but they can still respond to most of the monitoring checks. The consequence of this behavior is that the check periods for these services are always very short, thus a large number of messages are generated. As an example, the numbers of messages for CPR algorithm at 80% and 85% availability are roughly the same as the exhaustive approach.

To resolve this message number bursting problem in the algorithm, we have made two separate modifications to the original algorithm. One modification is to delay the cutting off of subsequent check period from after every failure to

every other failure. We called this modified algorithm CPR_2error, or CPR_2e. The rationale of this change is that occasional single check failure may occur due to many external factors, such as random network packet loss or momentarily busy service environment. But two consecutive check failures is a good indicator of a possible unresponsive extractor agent, especially when the time interval between two consecutive checks is relatively long (half a minute or one minute). So by postponing the action to after two consecutive check failures, the algorithm can avoid false alarms, thus reducing unnecessary message generation.

The other modification we made to the algorithm is to add another state to the FSA, called the *Unstable* state. We call this algorithm the Modified-CPR, or M-CPR. The new FSA is shown in Figure 4. A service's state can change to Unstable from Cautious Relax if two consecutive checks both fail. The state of the service will remain as unstable until two consecutive checks both succeed, and then the service will change back to the CR state. In the unstable state, the aggregator agent will check the extractor agent every 60 seconds. If another check fails in the unstable state, the service will be marked as inactive. While at the inactive state, if the final check succeeds, the service will be put to unstable state again for further checks.

The experimental results of these two modifications are also shown in Figure 5. We can observe that the CPR_2e modification reduces message number in the 90% to 95% availability level, while the M-CPR algorithm handles services at 80% to 85% availability range better. Both CPR_2e and M-CPR achieve the same performance with the original CPR algorithm (denoted by CPR_1e in the graph) at 99% availability.

To further test the performance of CPR algorithm and its variations in a more realistic scenario, we used a set of randomly generated availability values instead of a single value as the simulation output. The set of availability values ranges from 75.0% to 99.9%, differ by 0.1%. The same set of availability values is used by the exhaustive approach, the CPR_1e and the M-CPR algorithms. Results are shown in Figure 6. The total number of messages generated by the exhaustive method in a 24 hour duration is 373555, while the results for CPR_1e and M-CPR are 282086 and 250622, respectively. The percentages of reductions on message number are 24.5% and 32.9%, respectively.

V. CONCLUSION

In this paper, we introduce a framework, containing extractor and the aggregator agents to facilitate the management of web services considering the dynamic nature of cloud environments. Two adaptive algorithms, the CPR algorithm and its variation the M-CPR algorithms have been created to efficiently govern the communication between the agents that monitor the web services. Experimental results show that the proposed methods are more efficient than traditional exhaustive method with respect to a reduction in message amount for similar accuracy. The information obtained by the algorithm can also be used to facilitate service discovery and selection. In the future, we plan to evaluate the framework in an operational cloud environment and use the results to enhance the overarching inter-cloud approach.

REFERENCES

[1] M.P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," Comm. of ACM, Oct. 2003, Vol. 46, No. 10, pp. 25-28.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "A View of Cloud Computing," Comm. of ACM, Apr. 2010, Vol. 53, Issue 4, pp. 50-58, doi:10.1145/1721654.1721672.

[3] Y. Wei and M.B. Blake, "Service-Oriented Computing and Cloud Computing: Challenges and Opportunities," IEEE Internet Computing, Nov./Dec. 2010, Vol. 14, Issue 6, pp. 72-75, doi:10.1109/MIC.2010.147.

[4] X. Dong, A. Halevy, J. Madhavan, E. Nemes and J. Zhang, "Similarity Search for Web Services," Proc. 30th Intl. Conf on Very Large Data Bases (VLDB 04), Aug. 2004, Vol. 30, pp. 372 - 383.

[5] Apache Axis2/Java: http://axis.apache.org/axis2/java/core/, retrieved: Mar. 7th, 2011.

[6] Seekda Web Service Search Engine: http://webservices.seekda.com/, retrieved: Mar. 7th, 2011.

[7] ProgrammableWeb: http://www.programmableweb.com/, retrieved: Mar 7th, 2011.

[8] M.B. Blake and M. Gini (Guest Editors), "Introduction to the Special Section: Agent-based Approaches to B2B Electronic Commerce," International Journal of Electronic Commerce, Vol. 7, No.1, 2002

[9] M.B. Blake, "Agent-oriented Approaches to B2B Interoperability," The Knowledge Engineering Review 16 (4) (2001) pp. 383 – 388.

[10] D. Talia, "Clouds Meet Agents: Toward Intelligent Cloud Services", IEEE Internet Computing, Mar./Apr. 2012, pp. 78 - 81.

[11] K. Sim, "Agent-based Cloud commerce," Proc. IEEE 2009 International Conference on Industrial Engineering and Engineering Management (IEEM'09), pp. 717 - 721.

[12] B. Cao, B. Li and Q. Xia, "A Service-Oriented Qos-Assured and Multi-Agent Cloud Computing Architecture," Proc IEEE 1st International Conference on Cloud Computing (CloudCom'09), 2009, pp. 644 -649.

[13] J.O. Gutierrez-Garcia and K-M. Sim, "Self-Organizing Agents for Service Composition in Cloud Computing," Proc. 2nd Intl Conf. on Cloud Computing Technology & Science (CloudCom'10), pp. 59 - 66.

[14] K-M. Sim, "Agent-based Cloud Computing," IEEE Transaction on Service Computing, 2011, Issue 99, pp. 1 - 14.

[15] M.V. Haresh, S. Kalady and V.K. Govindan, "Agent based Dynamic Resource Allocation on Federated Clouds," Proc. 2011 IEEE Recent Advances in Intelligent Computational Systems (RAICS'11), pp.111 - 114.

[16] F. Barbon, P. Traverso, M. Pistore and M. Trainotti, "Run-Time Monitoring of Instances and Classes of Web Service Compositions," Proc. 2006 IEEE International Conference of Web Services (ICWS'06), pp. 63 - 71.

[17] L. Baresi, S. Guinea and P. Plebani, "WS-Policy for Service Monitoring, " Technologies for E-Services, 2006, pp. 72 - 83.

[18] L. Zeng, H. Lei and H. Chang, "Monitoring the QoS for Web Services," Proc 2007 IEEE International Conference of Service-Oriented Computing (ICSOC'07), pp. 132 - 144.

[19] O. Moser, F. Rosenberg and S. Dustdar, "Non-intrusive Monitoring and Service Adaptation for WS-BPEL," Proc. 17th International Conference of World Wide Web (WWW'08), 2008, pp. 815 - 824.

[20] M. Tian, A. Gramm, H. Ritter and J. Schiller, "Efficient Selection and Monitoring of QoS-Aware Web Services with the WS-QoS Framework," Proc. 2004 IEEE/WIC/ACM International Conference on Web Intelligence (WI'04).

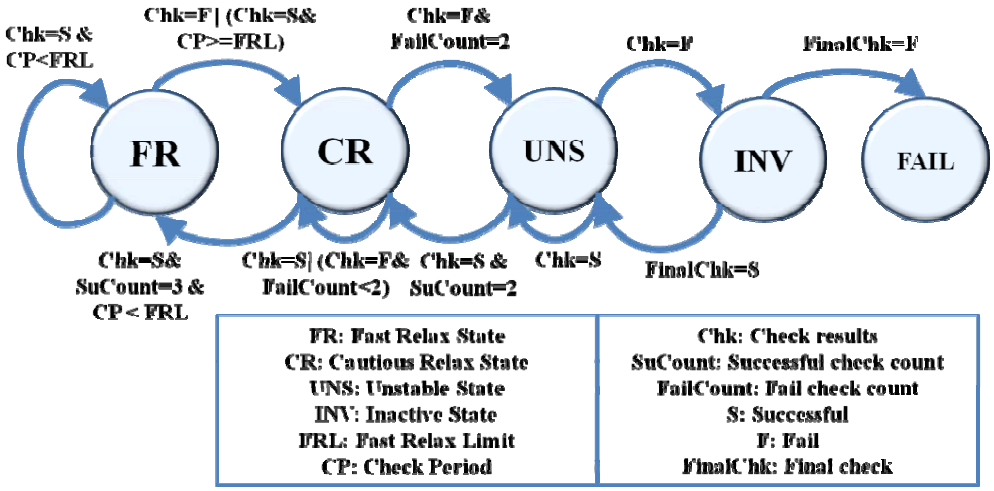

Figure 4. Finite State Automaton for Modified CPR Algorithm

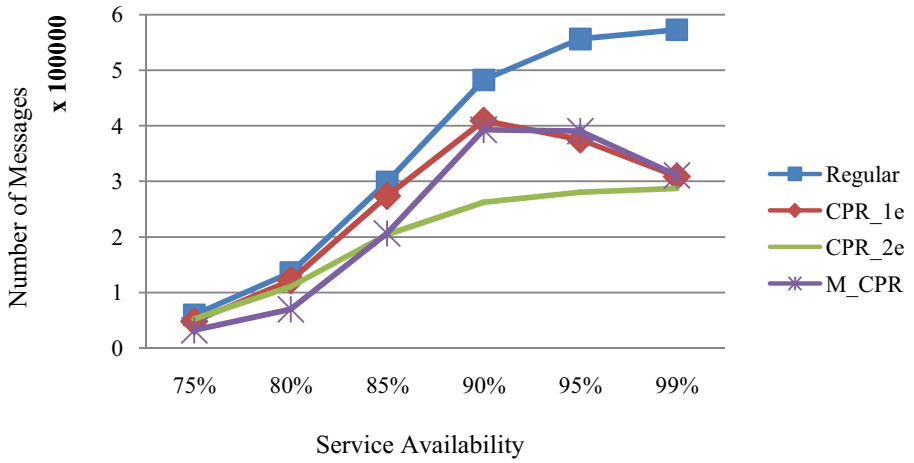

Figure 5. Number of messages generated by different approaches on different, but uniform availabilities

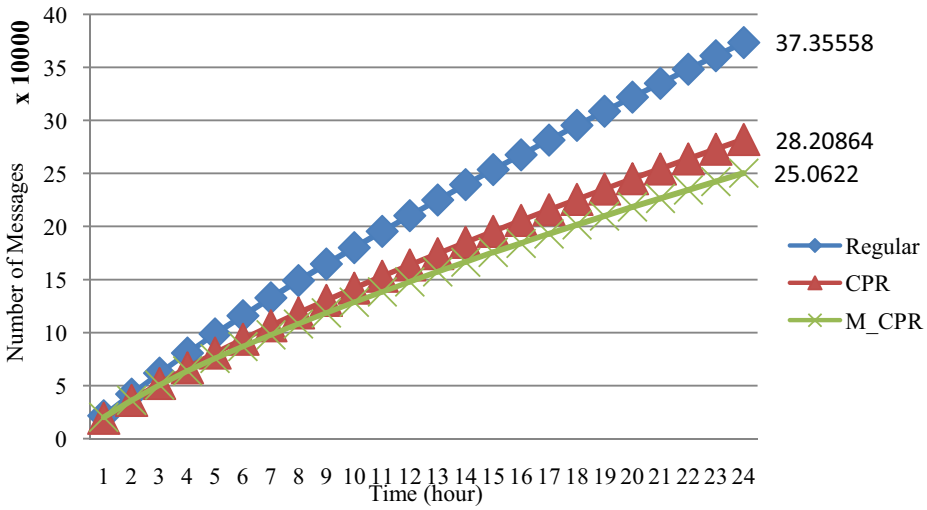

Figure 6. Number of messages generated by different approaches with a set of generated availability values. Each set of experiment is repeated 10 times and the average value is calculated and used.