Contemporary Engineering Sciences, Vol. 8, 2015, no. 13, 557 - 566
HIKARI Ltd, www.m-hikari.com
http://dx.doi.org/10.12988/ces.2015.5388

An Enhanced Automated, Distributed, SLA for Dynamic Infrastructure Management in Real Cloud Environment Using SEQ-BP(R)M Algorithm

V. Pavethra

Department of IT, Sathyabama University
Chennai, Tamilnadu, India

Senduru Srinivasulu

Department of IT, Sathyabama University
Chennai, Tamilnadu, India

A. Veeramuthu

Department of IT, Sathyabama University
Chennai, Tamilnadu, India

Abstract

Cloud Computing is an on-demand model on which Businesses, individuals access the need from anywhere in the on-demand way, which means software need not be installed for running the applications. Infrastructure need not be built physically, instead infrastructure has been built in the form of virtual for running the applications, Platform is one which consists of an operating system, Programming language for developing the applications. The Client Server-Agreement (CS-Agreement) is based on Service Level Agreements (SLAs) that regulates the mutual dealing between parties by defining the terms of involvement

for the active entities. This paper used JSON for better performance. The client is one who sends SLA Request in form of parameters, so that the SLA manager find the requirement for the client with the help of Sequential Best Particle (Resource) Matching (SEQ-BP(R) M) Algorithm for Dynamic Infrastructure management to improve efficiency by selecting the best Infrastructure as a Service (IaaS) from real cloud provider Amazon Elastic Compute Cloud (Amazon EC2 Instances) and Virtual private servers of GoDaddy product for running the job. Further, the system will be distributed to avoid single point of failure that is, if any failure occurs while sending the request for running the job, the request is passed to the other node without any corruption within zero deployment time using OpenGridGain, so that we get the response back regarding the job. A main objective is to attain dynamic infrastructure management by selecting the best infrastructure for job and also to overcome a single point of failure by using OpenGridGain. All these mentioned details are achieved.

Keywords: Cloud Computing, CS-Agreements (Client Server–Agreements), Infrastructure as a Service (IaaS), OpenGridGain, Sequential Best Particle (Resource) Matching (SEQ-B(R)M) Algorithm**,** Service Level Agreements (SLAs)

1. Introduction

This Paper proposes CS-Agreements, in which the client sends the request like os_type, cpu_num, free memory in a form of SLA Request [2]. The SLA manager is one who just finds the requirements is present or not. The catalog is one which gets the feed from the cloud regarding the client requirements and which is stored in the database. Data that are stored in database are dynamically updated (Current update) [5]. The orchestrator is one which has the complete knowledge of the element status, so that it can perform the scheduling of the resource allocation based on Best Particle (Resource) matching algorithm to select the best Infrastructure for the client to run the job. This is how the best Infrastructure is chosen for the job based on the client requirements.

In this system, we use OpenGridGain which helps to distribute the system to avoid the single point of failure, if any failure occurs while sending the request for the job at that time without any corruption the request is passed to the other node within zero deployment time, and we get the response back for the job. Due to this we need not do the same process again and again, which is too tough. By this we can save our time. The proposed work is carried out to avoid the failure and select the best infrastructure for running the job.

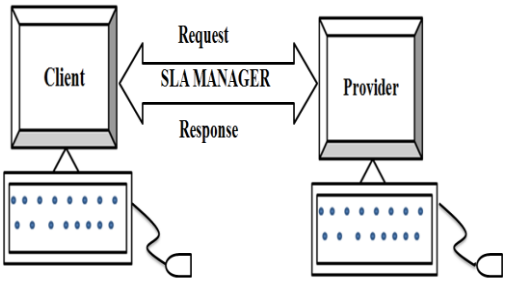

Fig. 1 SLA Transaction

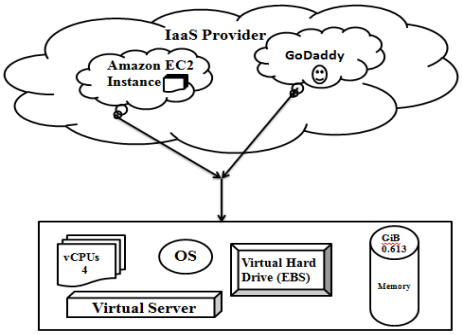

Fig. 2 IaaS Service

2. Proposed Model

This Proposed model main aim is to get the best infrastructure for running the job and avoid single point of failure [1].

A. Problem Description

Due to the absence of the distributed SLA Manager, the process cannot be redeployed if any failure occurred; we need to do all the process again which is too tough, and time consuming. Based on client requirements it cannot able to select the best infrastructure for the job, Dynamic infrastructure management is missing. Here it uses XML format which involves more time for creating and parsing.

B. System Architecture

Here with the help of OpenGridGain the system is distributed. We just start the node with the help of OpenGridGain which is shown in User Interface [9]. Here it deploys SLA manager dynamically to all nodes. If node fails main nodes set, detect and update. If the new node starts the main Node will detect it or deploy it. All remote nodes can be redeployed. All such things are done within zero deployment time.

The catalog gets the Current System Information like IP Address, Total Memory, CPU Count and RAM, all these details are stored in Database [10] so that it can allocate [9] the Job. That is, the request (parameter) is sent by the client to the SLA manager, which is shown in Figure 1. The SLA manager checks the database which has current information. All these details are stored in the database from the catalog. [7] The catalog gets feed from the cloud. Orchestrator checks all the requirements are present that is it just knows all the current status,

which is shown CS Agreement architecture in Figure 3. Then it goes to the connector and to the real cloud setup, which is shown in Figure 2, for running the job by choosing the best infrastructure using (SEQ-BP(R) M) Algorithm**.**

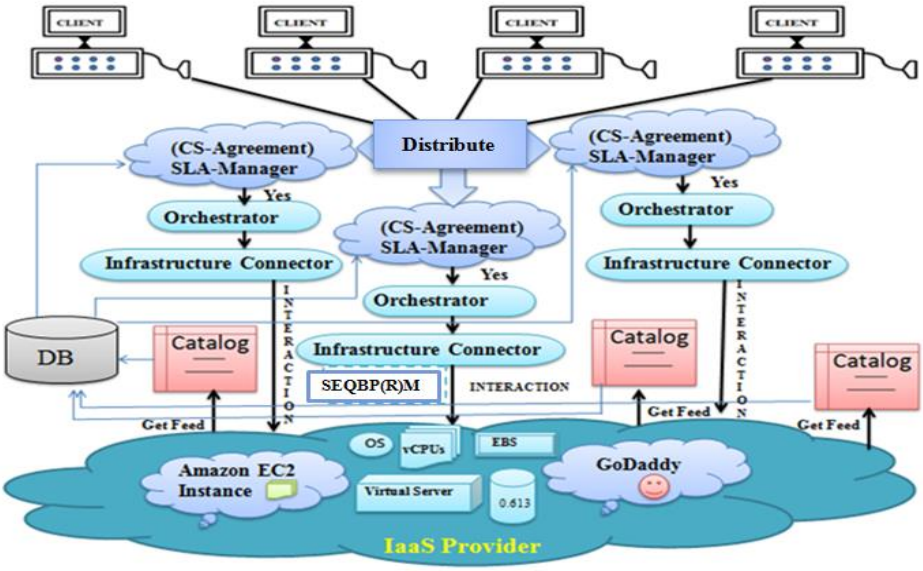

Fig. 3 CS Agreement Architecture

As the proposed model consists of a distributed SLA manager [4] to avoid single point of failure, It just Eliminate Disaster Recovery by Selecting the Best Infrastructure [3] for running the job using (SEQ-BP(R)M) Algorithm in an on-demand [8] way, and Use JSON to avoid Time Consumption for creating and parsing[6].

C. Algorithm

Sequential Best Particle (Resource) Matching Algorithm

Step 1: Initialize the particle List;
Initialized ();
Step 2: Get the particle Manager Values;
PSODBManager ();
Particle List = manager. getAllParticles ();
Step 3: GetFitness for OSType
OSSelectedList ();
Step 4: Calculate Fitness For OS
params. getOsType ();
Print ("OperatingSystemselection list +osSelectedList");
if (OSSelectedList.size() > 0)
particleList = OSSelectedList;
Step5: Do fitness for the no of cpus
cpuSelectedList ();
Step6: Calculate Fitness For CPUS
params.getCpuNums();

```
        print("cpu based selection "+cpuSelectedList);
    if(cpuSelectedList.size()>0)
Collections.Sort(cpuSelectedList,newComparator<Particle>()
Step7: compare(Particle h1, Particle h2)
    if(h1.getNoOfCPUs() > h2.getNoOfCPUs())     return 1;
    if(h1.getNoOfCPUs() < h2.getNoOfCPUs())     return -1;
Step 8:Return
    System.out.println(cpuSelectedList);
    Particle particle = cpuSelectedList.get(0);
    return particle.getHostIP();
Step 9:Same as that fitness is calculated for free memory.
    If (particle.getNoofCPUs () >= CpuNumber)
```

Algorithm 1

Then print the CPU Type matched same as that process is done for all other parameters. Here client sends the request in form of parameters such as os_type of client is fixed as p_best and it is checked with current updated os_type in the database (Fitness value). Each and every time it checks for matching with client requirement. Now p_best is made to g_best, because it gets matched with client requirement, which we got from database same as that we do the sequential form of matching for other attributes such as cpu_num and free memory to get the best infrastructure for running the job based on client requirement with the help of SEQ-BP(R) M algorithm, which is given in Algorithm 1.

3. Experimental Results

The number of nodes started, here the place where client sends the required parameters as a request for running the job, which is shown in Figure 4.

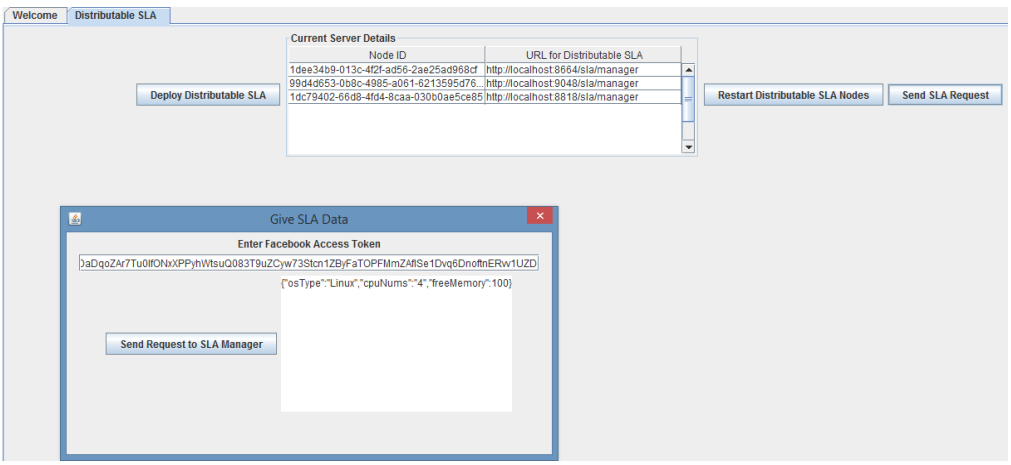

Fig. 4 Distributed service level agreement

The current updates, of all nodes such as os_type, cpu_num, total memory, free memory are updated in database, which is shown in Figure 5.

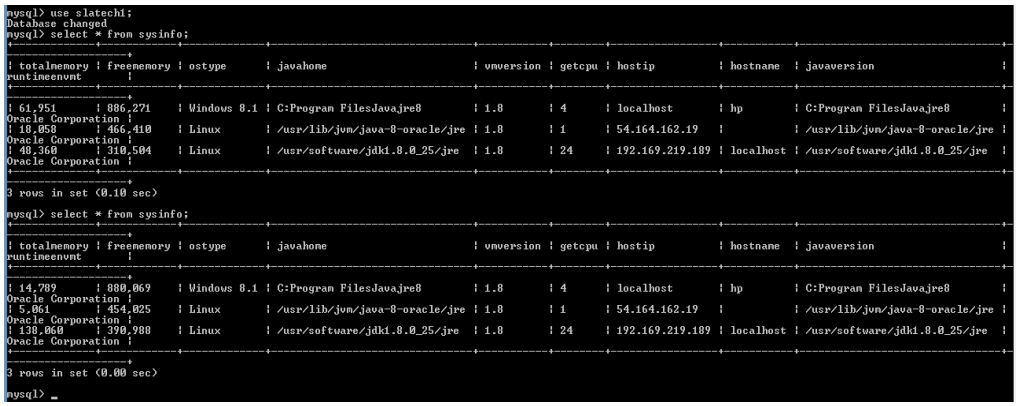

Fig. 5 Current database updates

The SLA Service running in real cloud setup and it also represents the request and response process of the job, which is shown in Figure 6.

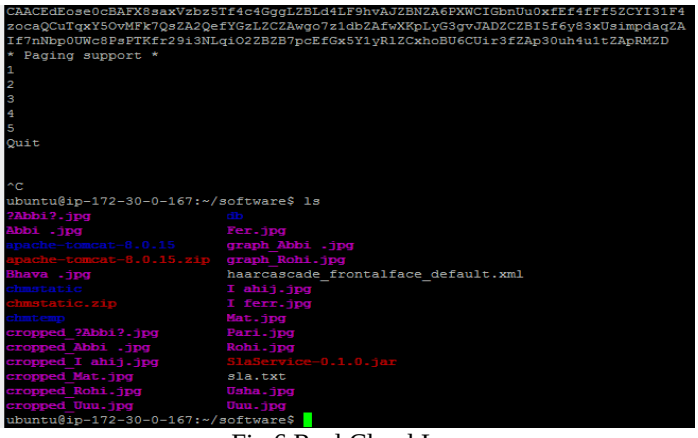

Fig.6 Real Cloud Log

The response for the job and it also represents, which cloud the job is running like Amazon EC2 Instance or GoDaddy product, which is shown in Figure 7.

Fig. 7 Response

4. Performances Analysis

A. Nodes Started by OpenGridGain Shown in User Interface

The OpenGridGain used to start the nodes, which is same as our own system. Here it consists of Node ID, Number of nodes started, version, CPU and heap. Here the Table 1 shows the number of nodes started with OpenGridGain that is, it shows Node ID and URL for Distributed Service Level Agreements (DSLA) for the number of nodes started.

As of now Node ID 1dee34b9-013c-4r2f-ad56-2ae25ad968cf, is Started by code and its URL Is http://localhost:8664/sla/manager other node started by OpenGridGain has Node ID 99d4d653-0b8c-4985-a061-6213595d76ff and its URL is http://localhost:9048/sla/manager. Same as those other two nodes are started, which is shown in the Table 1. All these things are done to deploy the request of job into another node, in case of any failure occurred while passing the parameter for running the job in IaaS server of the real Cloud.

Table 1: Current Server Details Started by OpenGridGain

Node ID	URL for Distributed SLA
1dee34b9-013c-4r2f-ad56-2ae25ad968cf	http://localhost:8664/sla/manager
99d4d653-0b8c-4985-a061-6213595d76ff	http://localhost:9048/sla/manager
1dc79402-66d8-4fd4-8caa-030b0ae5ce85	http://localhost:8818/sla/manager
F9d1bad8-198c-438e-ae0d-ad4a7542ae09	http://localhost:8202/sla/manager

B. Response for given Request by SLA Manager

As the Table 2, shows the Parameters passed by the Client for running the job in real cloud setup (IaaS) [1]. Here we use SEQ-BP(R)M Algorithm for selecting the best IaaS server for running the job as of now if client passes the parameter 1 such as os_type- Linux, cpu_num-1, free memory-95 with the help of SEQ-BP(R)M Algorithm, job run in Amazon EC2 Instance which is represented in Figure 8**.** The Table 2 shows the parameter 2 such as os_type-Linux, cpu_num-4, free memory-80, which is passed by the client for running the job. Here SEQ-BP(R)M Algorithm is one which select the best real cloud setup (IaaS) such as GoDaddy for running the job based on parameter 2,which is shown in Figure 9, same as that all other parameters are passed and tested which is shown in Table 2.The graph which has time(sec) represent the time taken for getting response for the given job. In Figure 10 shown the result of two different parameters passed at a time such as parameter 3, os_type-Linux, cpu_num-1, free memory-100 and parameter 4 such as, os_type-Linux, cpu_num-12, free memory-96, so that the job run in both Amazon and GoDaddy (IaaS) Server at a time, and we get the response from both the server. Here it represents the minimum time taken by the job to run in Amazon and GoDaddy as compared to Figure 8 and Figure 9. By this we can tell that performance is good.

Table 2: Job Run in Real Cloud Server (IaaS)

os_type	cpu_num	Free memory (mb)	IaaS Server
Linux	1	95	Amazon
Linux	4	80	GoDaddy
Linux	1	100	Amazon
Linux	12	96	GoDaddy

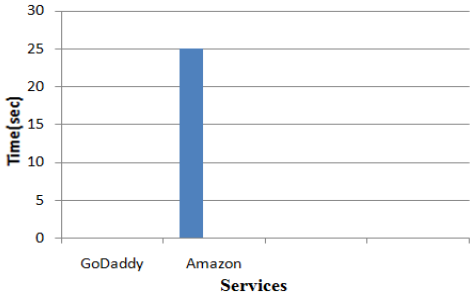

Fig. 8 Job runs in Amazon EC2

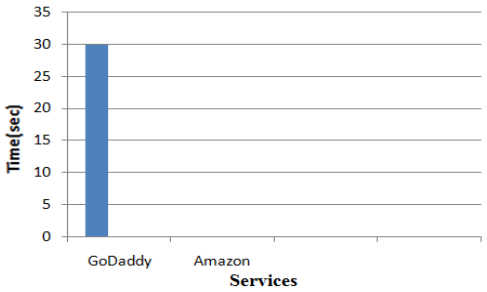

Fig. 9 Job runs in GoDaddy

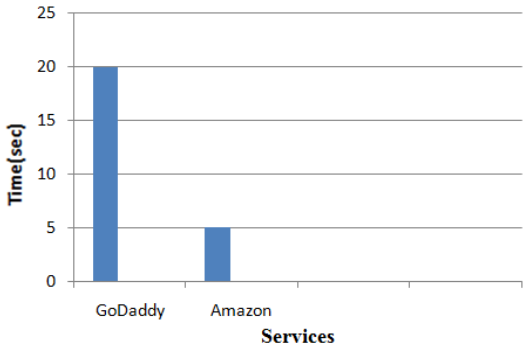

Fig. 10 Job runs in both Amazon EC2 and GoDaddy

5. Conclusion

We introduced the real cloud network service stipulation, CS-Agreement which can be deployed in an on-demand stage. For this implementation, we have entered on these modules and function, proposed by the CS-Agreement which is called as real client-server scheme. In this scheme the cloud client integrate with the service provider. In this paper CS-Agreement has been accosted and implemented using SEQ-BP(R) M algorithm and quite often using OpenGridGain to avoid single point of failure, while requesting the job to run in infrastructure as a service. In conclusion, we proposed to quickly implement an efficient cloud service, the behavior of the CS-Agreement between client and the provider for running the job in best infrastructure base on client needs, this provided the opportunity to make use of service level agreements in an on-demand stage.

References

[1] An Oracle White Paper, Making Infrastructure-as-a-Service in the Enterprise a Reality, April 2012.

[2] C. Coutcoubetis, V. Siris, Managing and pricing service level agreements for differentiated services in Quality of Services, 1999, IWQoS ’99, 1999 Seventh International Workshop on 1999. http://dx.doi.org/10.1109/iwqos.1999.766492

[3] Demosthenes Kyriazis, Cloud Computing Service Level Agreements.

[4] M. Alhamad, T. Dillon, E.Chang. Conceptual Service Level Agreement framework for cloud computing. In Digital Organization and Technologies (DEST), 2010 4th IEEE International Conference on, page 606 -610, 2010. http://dx.doi.org/10.1109/dest.2010.5610586

[5] P. Goncalves, S. Roy, T. Begin and P. Loiseau, Dynamic Resource Management in Clouds: A Probabilistic Approach, August 2012.

[6] Roland Kubert, Gregory Katsaros, Tinghe Wang, A RESTful implementation of the WS-Agreement Specification, 2011.

[7] R. Moreno-Vozmediano, R. S. Montero and I. M. Llorente, IaaS Cloud Architecture: From Virtualized Datacenter to Federated Cloud Infrastructures, IEEE Computer, vol. 45, no. 12, pp. 65 -72, December 2012.
http://dx.doi.org/10.1109/mc.2012.76

[8] R. Murch, Autonomic Computing, On Demand Series: IBM Press, 2004.

[9] Sameera Abar, Georgios K. Pierre, Automated Dynamic Resource Provisioning and Monitoring in Virtualized Large Scale Datacenter, May 16, 2014.

[10] Senduru Srinivasulu, P. Sakthivel, Contemporary semantic web: The primary social and technical challenges, Journal of Theoretical and Applied Information Technology, vol.72, no. 3, 2015, 337-346.

Received: March 18, 2015; Published: June 4, 2015