Achieving Dynamic and Distributed Session Management with Chord for Software as a Service Cloud

Zeeshan Pervez, Asad Masood Khattak, Sungyoung Lee[†], Young-Koo Lee
Ubiquitous Computing Lab, Kyung Hee University, Yongin, Korea
Email: {zeeshan, asad.masood, sylee}@oslab.khu.ac.kr, yklee@khu.ac.kr

Abstract— **Cloud computing - started as a buzz word is rapidly embraced by the enterprises and preached by the technological evangelist. Availability of high bandwidth internet at the end user level, and the adoption of virtualization for efficient resource utilization by the datacenter management has given birth to this new computing paradigm. It promises colossal on-demand processing and storage capacity along with scalable service delivery model. Software solution providers are applying cloud computing to reduce service provisioning cost, by providing their business functionality as a service. However, it requires modification in context of how existing services are provisioned. Existing session management policies require dedicated computing resources to process sessions; this deviate from the concept of "Pay-As-You-Use". To conform to cloud computing architecture there is need to decouple session management from the provisioned services. Derived by the need of on-demand service provisioning in this paper we present a decentralized session management framework inspired by P2P routing protocol. We call the proposed framework Chord based Session Management Framework for Software as a Service Cloud (CSMC). Applying CSMC eliminates the need of separately deploying computing resources for sessions management, in fact CSMC uses existing least utilized resources within Cloud Area Network (CAN). CSMC tested on three different cloud configurations highlights the fact that CSMC can be effectively deployed in cloud to achieve seamless service scalability. Additionally, we have tested CSMC on different web-servers to highlight its efficacy of session management on varied cloud infrastructure.**

Index Terms— **Session Management, Algorithms, Management, Measurement, Performance**

I. Introduction

Software delivery models have evolved over the time: from stand alone applications to client server architecture and from distributed to service oriented architecture (SOA) [1]. All of these transformations were intended to make business process execution effectual and to provide ease of use. New software delivery models emerge due to the fact that either the earlier delivery models were not supporting the business needs or technological advancement have broken some barriers which were considered to be inevitable in previous ones. Exponential increase in processing power of enterprise servers, adoption of virtualization and availability of high bandwidth to the end user have given birth to a new type of computing paradigm known as cloud computing [2]. It encompasses Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS).

Among the cloud computing stack, SaaS is a software delivery model, which provides access to business functionality remotely as a service [3]. Leading companies in Information Technology industry are gradually moving their applications and related data over the internet and delivering them through SaaS [4]. Google has used SaaS platform to offer web applications for communication and collaboration [5], gradually replacing recourse exhaustive desktop applications. Similarly, Microsoft is offering their development and database services though SaaS called Microsoft Azure [6]. SaaS is preached by companies like SalesForce, 3Tera, Microsoft, Zoho and Amazon, as a result of which business specific services can be consumed in ubiquitous environment.

One of the distinguishing features of cloud computing is the adoption of virtualization [2], that helps service providers to provision their services on-demand basis. These services encompass software (business application) and data storage services or even hardware and network resources as a service [7]. The concept of "pay-as-you-use" is principally backed by virtualization. Although on-demand service (service scaling) is just a matter of simple click as advertised by most of the cloud hosting providers [8]. But in fact there are lots of issues related to this simple click event; session management, virtual machine deployment, and load balancing are few of them. Services are scaled (up or down) to comply with service level agreement (SLA) signed between the service provider and service consumer or to reclaim the resources when they are not required (less number of concurrent users).

SaaS can be classified in four levels [9]; at the highest level (Level-IV) services are multi-tenant and configurable. Multi-tenant services are developed keeping in view the heterogeneity of service consumers. Diverse consumers can subscribe to same instance of a service, yet they will experience bespoken response according to their business requirements. Level-IV services are most lucrative for any service provider, since they only need to spend

This paper is based on "CSMC: Chord based Session Management Framework for Software as a Service Cloud" by Z. Pervez, A.M. Khattak, S.Y. Lee, and Y.K Lee, which appeared in the Proceedings of ICUIMC '11, February 21-23, 2011, Seoul, Korea. © 2011 ACM.

This research was fully supported by Microsoft Research Asia.

[†]Professor Sungyoung Lee is the corresponding author of this paper.

once on development process and later on these services can be configured according to customer requirements. However, provisioning these types of services demands some tailored management procedures in terms of session and load balancing.

Session management procedures which are currently deployed by hosting providers works well in web architecture (Client Server), where there is no concept of "pay-as-you-use", and resource utilization is not considered at the utmost priority. Existing session management algorithms used by most of the web-servers are developed by the hypothesis that resources (compute and storage servers) will be available throughout their lifecycle. Although web-servers provide disaster recovery mechanism by replicating session information on multiple servers, nevertheless they are not adaptive to true dynamic nature of cloud, in which resource can be added or removed with a single command on cloud management console.

Deploying service in cloud using these conventional session management procedures increases service provisioning cost as they demand dedicated resources to process and store session information. Apart from that these session management procedures also hinder in the development of Level-IV services, as sessions are bound to particular instance of a web-server, restricting consumers to one instance of a server. In this paper we present a decentralized session management algorithm which is not bound to any particular web-server. Decoupling of session management helps in provisioning on-demand services and reclaiming cloud compute resources when they are not required to reduce service provisioning cost; without affecting existing active sessions. We use P2P routing protocol (i.e., Chord) to distribute session values among the cloud compute resources. With P2P routing protocol, resources are efficiently utilized without the need to have a dedicated session management server.

The rest of the paper is organized as in Section II we will discuss the different session management methodologies provided by the existing web-servers. Section III will summarize some of systems in which Chord is successfully utilized to develop distributed applications. Section IV will present our proposed distributed session management framework. Section V will talk about the session management procedure using Chord. Section VI will outline our test bed used for experiments, along with implementation strategy. In Section VII we will present our results in three different configurations. Section VIII will present the future work and finally in Section IX we will conclude our work.

II. Related Work

Web applications and services are hosted on web-servers to deliver their contents and functionality to the end user. There is an exhaustive list of web-servers used by the industry to provision web applications and services. Three well known and commonly in use web-servers are Internet Information Services [7], Apache Tomcat [10], and GlassFish [11]. These web-servers are designed to

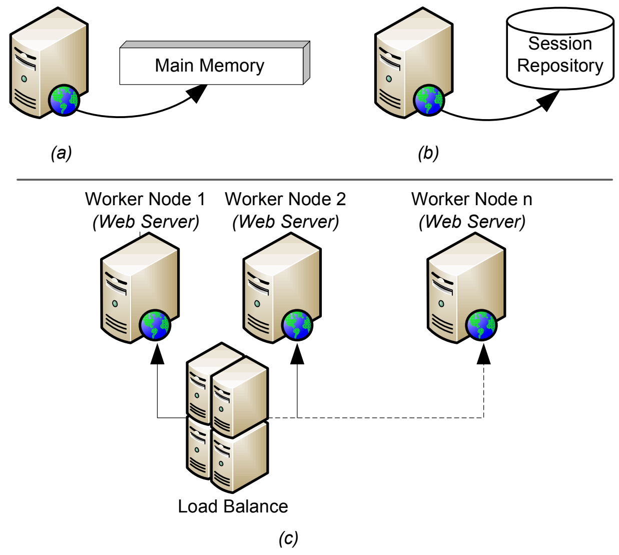

Main Memory
(a)
Session Repository
(b)
Worker Node 1 (Web Server)
Worker Node 2 (Web Server)
Worker Node n (Web Server)
Load Balance
(c)

Figure 1: Web-Server Session Management Methodologies

achieve high throughput, and to cater flash crowd problem. Besides this with the emergence of Web 2.0 interactive application concept, these web-servers are configured to execute numerous HTTP Post requests generated by an individual client application. Various strategies have been adopted by these web-servers to provide desktop application like experience in web applications, which led them to session management in various ways.

Mainly sessions are handled by applying three distinctive methodologies, subject to application requirements. These requirements include number of concurrent users, session validity period, and inter arrival time of a request. Apart from that, usage of session in application also influences the decision of an application architecture in selecting the suitable session management methodology. Below we have discussed three different session management procedures used in most of the web-servers.

A. *Main Memory Based Session Management (MMB)*

Main Memory Based Session Management (MMB) is frequently used and is enabled by default in most of the web-servers [12], [10], [11]; persists session information in worker process of the web-server shown in Figure 1(a). This strategy is best suited for an application which has a limited number of concurrent users. Whenever, session information is required web-server can extract it from the worker process. In this methodology session state depends on the life time of the application; if application is restarted all of the active sessions are lost. This methodology works well for applications where session is not intensively used in business logic to persist data.

B. *Repository Based Session Management (RB)*

Repository Based Session Management (RB) is applied to medium size applications [12], [10], [11]. It persists sessions in dedicated database or file, called session repository, shown in Figure 1(b). This approach is used in applications where sessions are rigorously used to store business objects when navigating between web pages. With this approach session validity period can be increased to a much longer duration as compared to MMB. In addition to that, it also facilitates application developer to persist entire business object in session without compromising application response time. It also trims down the usage of main memory and takes advantage of database algorithms for searching and indexing huge repository of active sessions.

C. Dedicated Machine Based Session Management (DMB)

Third methodology is applied for massively large applications. It is best suited for the applications [12], [10], [11], where sessions are persisted on multiple locations in order to avoid any failure and to achieve load balancing in case if there are too many hits on a web-server. Figure 1(c) shows DMB topology, consisting of one Load Balancer (master node) and multiple worker nodes (web-server). This approach is mainly adopted by enterprise applications where sessions are created for a much longer duration of time and must be persisted to increase user experience, and to reduce the dependency on other components (in case if business object are not subject to frequent changes). This approach requires dedicated resources for session management. Usually scheduling algorithms is deployed on a master node that routes the incoming requests to an appropriate web-server. Apart from that, this technique of session management requires replica of session repository on each web-server.

IIS, Tomcat, and Glassfish are shipped with these three session management approaches described earlier, with a few variations. IIS use Microsoft SQL Server for Repository Based session management whereas; Glassfish use local file system to persist session information instead of dedicated database. Tomcat use FileStore as an alternative of main memory, for every new session a separate file is created in FileStore that persist session information. However, for the DMB approach all of these web-servers employ same strategy, at master node load balancer is deployed and actual sessions are persisted on multiple worker nodes.

Existing web-servers provide session management procedures explicitly engineered keeping in view the web architecture. Cloud computing preaches on-demand virtualized services which can scale accordingly to their utilization requirements. There is need of session management procedure that can scale with services. Making use of dedicated compute nodes for session management will restrict service provider to provision services on-demand. P2P algorithms are well known for their scalability and distributed nature. A lot of literature has been published on P2P routing protocols. Chord is a P2P routing protocol which has been successfully used in various application to achieve scalability.

III. Chord in Distributed System

Chord [13] is a lookup protocol dedicated to internet applications that need to discover any type of resources maintained by the users that form an underlining network [14]. It provides an elementary service: for a given key, Chord returns a node identifier that is responsible for hosting or locating the resource. Chord has been deployed in several applications: CFS (Collaborative File System) [15] which is an internet scale distributed file system, and ConChord [16] which uses CFS to provide a distributed framework for the delivery of SDSI (Simple Distributed Security Infrastructure) security certificates.

Some applications employ Chord in a much different way as compared to file sharing applications. Snapshot [17] is a distributed network management algorithm developed on the basis of Chord. This management scheme helps telecommunication carries to gather information about the current performance capabilities of their network. Besides this it also assists them in monitoring entire or subset of the network. Each subset of the network creates the snapshot of the underlying network which is then used to identify the point where counter measures are required.

Network heterogeneity is another problem which can affect the response time of a lookup query in Chord network. Not all participating nodes possess the same processing power and network bandwidth. [18] is another file sharing variant of Chord which addresses network heterogeneity. To overcome this problem they proposed an improved Chord model, based on Topic-Cluster and Hierarchic Layer (HTC-Chord). The proposed algorithm divides the network according to the interest (Topic) and processing capabilities of the available nodes. Through this scheme, lookup request is restricted to a subset of nodes, in which the nodes have same interests. As a result of which, response time of a request is reduced since it is only routed to the nodes having similar interest and possess appropriate processing power.

[19] proposed a key look strategy based in Power Law. They have introduced the concept of Super Node, which possess huge processing and high bandwidth availability. Super Node works as an anchor node, instead of diving deep in Chord network, request are entertained by the Super Nodes, avoiding nodes which has less processing capabilities.

IV. System Architecture

CSMC is a Chord based session management framework for Software as a Service cloud (SaaS), which provides distributed session management enabling session decoupling. CSMC enables services providers to achieve seamless service scalability, without interfering the processing of existing active sessions. The component stack of CSMC consists of six managerial components shown in Figure 2.

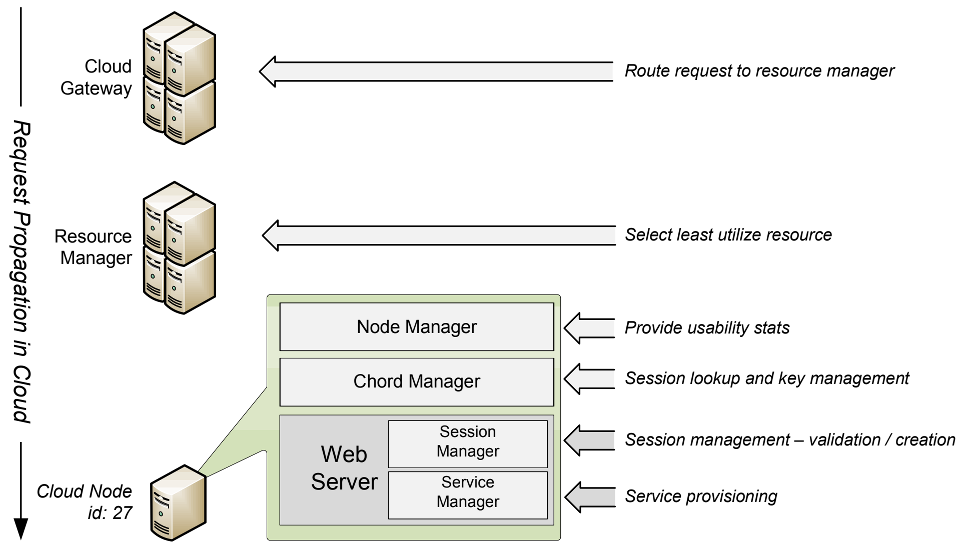

Request Propagation in Cloud
Cloud Gateway
Route request to resource manager
Resource Manager
Select least utilize resource
Node Manager
Provide usability stats
Chord Manager
Session lookup and key management
Web Server
Session Manager
Session management – validation / creation
Service Manager
Service provisioning
Cloud Node id: 27

Figure 2: CSMC Component Stack

A. Cloud Gateway (CGW)

The top most layer of CSMC is CGW, it works as an entry point in SaaS. Every service provisioned by the cloud is accessible through CGW. Applications consuming cloud hosted services will have no idea about the underlying component stack, for them CGW is the service provider. This high level of abstract is very useful during service scaling. As client applications are only interacting with CGW, there is no need to change service binding when multiple instances of a service are deployed. Internal components of CSMC will automatically route the clients' request to the most appropriate instance. Underlying components of CSMC stack will ensure that services are provisioned accordingly to the signed SLA.

B. Resource Manager

Effective resource utilization is one of the key selling point of cloud computing. Resource Manager is the component which has the global view of resource utilization in cloud. In order to avoid bottlenecks, Resource Manager constantly routes the requests to least utilized resources. It works like a bridge between CGW and actual computing resources in cloud. Consequently, CGW does not need to handle request forwarding task instead it has been be delegated to the Resource Manager. This enables CGW to manage incoming requests while Resource Manager governs selection of the most appropriate service instance.

C. Node Manager

Node Manger assists Resource Manager in selecting the appropriate service instance depending on clients' SLA. It periodically updates information about resource utilization to Resource Manger. It monitors the worker thread of web-server to analysis the response time of the node. Each node in CSMC enabled cloud is deployed to provide two primarily functions; first is service provisioning and second is Chord based session management. Service provisioning is the core function of every node. In order to avoid the situation where too many request are routed to the same node; each individual Node Manager constantly examines the processing capacity of the node, and update the Resource Manager.

D. Chord Manager

Chord Manager is the core component in CSMC, entire Chord related functions are handled by Chord Manager. Functions like key value lookup (*session identifier*), finger table scan, and request forwarding are handled by it. In short, Chord Manager is responsible for distributed session management. In CSMC every node is responsible for storing fraction of the active sessions. A unique chord identifier is assigned to each compute node by the Resource Manger. Sessions are allocated to each compute node according to its chord identifier. In interactive applications, session management is one of the prime concerns of application developers, as well as for the service providers. As applications are becoming more and more interactive, sessions are intensively used by the application developers to persists business objects during HTTP Post request [20] or in case of partial call back operation (*AJAX*) [21]. Chord Manager together with Session Manger help hosted services to validate session authenticity and provide desired session information.

E. Session Manager

For every legitimate user new session is created if it does not exist or if its validity period has been expired, by session manager. Since HTTP is a stateless protocol, session management is the most efficient mechanism to persist business objects while navigating between the web pages. Apart from that, session is also used to validate user. Every session is valid for a particular period of time after that it is discarded. Importance of session management is escalated if the client application is an interactive application, which needs quicker response as compare to conventional web applications. Besides this, as web applications are providing functionalities with desktop like experience, more and more business objects are persisted in session variable that demands more memory space and reduced lookup time.

F. Service Manger

In SaaS architecture single compute node provides multiple services though virtualization. In order to identify their usage pattern Service Manger is added. Service Manger keeps track of all of the hosted services on a single compute node. Service Manger assists SLA manger in deciding which service should be scaled for effective resource utilization. It also assists Node Manager in analyzing worker process of the web-server that helps in reducing session lookup time.

Collectively, all six managerial components of CSMC facilitates in achieving distributed session management in cloud, driven by the need of cost effective resource utilization. With CSMC, there is no need of dedicated session management components in cloud which increases service provisioning cost and can become a bottleneck in case of flash crowd.

V. Session Management in CSMC

In cloud computing services are scaled according to the number of concurrent users/requests. [9] describes four level of service provisioning models, at the highest level (*Level - IV*); services are scalable, configurable and possess the multi-tenant property. To achieve true multi-tenancy, there is a need to decouple session management from a particular web-server. In cloud, services are provisioned on virtualized resources (*Virtual Machines*) and these resources can be reclaimed back if not required or more virtualized resources can be added if necessary. In this context availability of web-server is depended on number of concurrent users. To achieve seamless service scaling (*up or down*) there is need of session decoupling so that execution of concurrent session is not disrupted and newly session can be created seamlessly.

With CSMC we have accomplished true session decoupling with the hosted services. CSMC enables service providers to scale their services without making any changes in the underlying configuration. Figure 3 shows CSMC topology in Cloud Area Network (*CAN*).

CSMC works in a collaborative manner. Every component of CSMC provides assistance to other component. As a result single point of failure is avoided and also this type of disseminated strategy is best suited for flash crowd problem that demands additional compute nodes. CGW is the point of interaction for every service consumer. The idea is similar as that of Service Oriented Architecture (*SOA*), services are exposed without providing the internal service composition logic. Every service consumer binds client application with CGW to consume the hosted services. Received request is then delegated to Resource Manager which routes the request to the least utilized

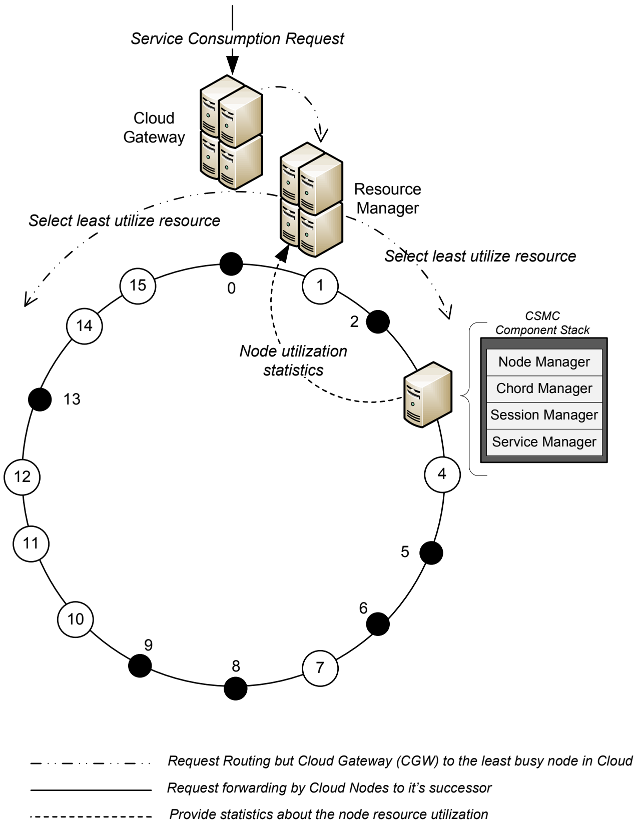

Service Consumption Request
Cloud Gateway
Resource Manager
Select least utilize resource
Select least utilize resource
Node utilization statistics
CSMC Component Stack
Node Manager
Chord Manager
Session Manager
Service Manager
0
1
2
4
5
6
7
8
9
10
11
12
13
14
15
Request Routing but Cloud Gateway (CGW) to the least busy node in Cloud
Request forwarding by Cloud Nodes to it's successor
Provide statistics about the node resource utilization

Figure 3: CSMC in Cloud Area Network

service instance according to the SLA. At very abstract level Resource Manger performs the request routing but internally it segregate the request according to the SLA and select the resource (*compute node / service instance*) which is most suitable for conforming the SLA. This type of resource selection requires resource utilization information, which can provide information about current processing capabilities of a compute node. In CSMC this utilization information is provided by the Node Manager, which periodically intimate Resource Manger about its processing capabilities.

On receiving service usage request, individual service needs session information in case of HTTP Post request. In CSMC enabled cloud sessions are not bound to any particular instance of a service; in fact sessions are bound to compute nodes according to the session identifier. To locate session within CAN, session identifier is utilized which indicates the node responsible for maintaining the session information. Session information is retrieved from CAN though Chord in logarithmic time $\frac{1}{2}LogN$, where N is number of compute nodes in CAN [13].

Figure 4 shows Chord topology for CAN of 8 nodes having maximum capacity for 16 compute nodes. It is clear that current cloud capacity can be doubled without requiring any configuration alteration in existing topology. Black dots in Chord ring show the absence of compute nodes, whilst white circles show the actual compute node on which request can be routed. On each compute node CSMC component stack is deployed which helps in

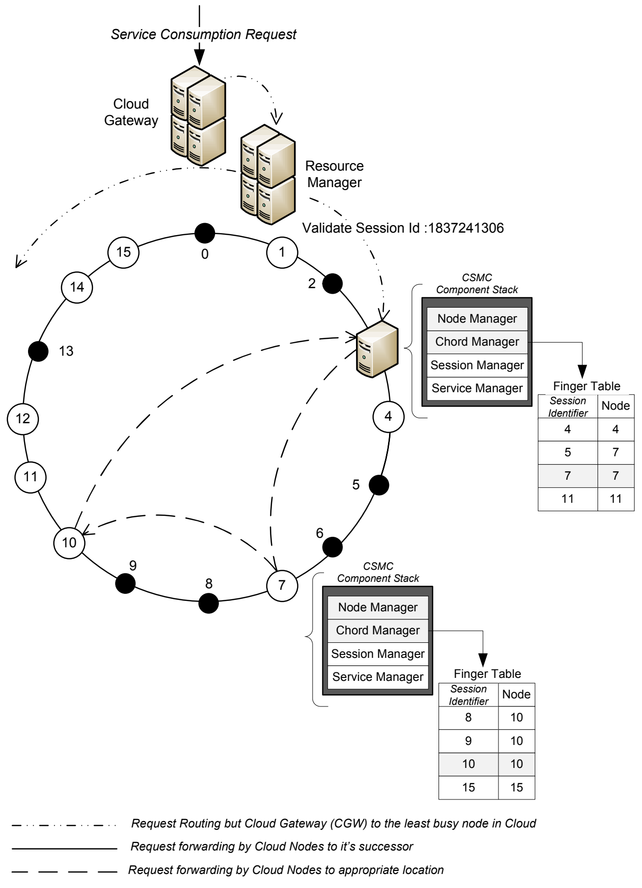


Figure 4: CSMC Session Lookup Request Propagation

maintaining Chord ring and intimating Resource Manager about its processing capability. This information helps in automated request routing and service scaling. Each compute node is connected to its successor in the Chord and additionally contains the finger table to route session lookup query to an appropriate node.

VI. Test Bed Implementation

We tested CSMC for different number of compute nodes 5, 9 and 20 represented by the modulo of 2^3, 2^4 and 2^5 chord space respectively. The test bed consists of a Cloud Gateway and a Resource Manager, and multiple compute nodes. Cloud Gateway and Resource Manager are deployed on Windows 7 Enterprise running on Intel Quad Core with 4 GB RAM and 360 GB hard drive. Compute node are virtualized images of Windows XP Service Pack 3.0 having 2.0 GHz of processing power and 1.5 GB of main memory. On all compute nodes IIS 5.10, Apache Tomcat 6.0, GlassFish Server 3.1.1 are deployed as web-servers along with .Net Framework 4.0 and jre 1.6 as runtime environments. On each compute node same instance of a web service is deployed to mimic the business logic provisioned by multiple compute nodes. We used OpenSTA [22] as a load generator for the hosted services. To test CSMC for IIS, component stack is developed in .Net Framework 4.0 whereas for Apache Tomcat and GlassFish Server it is developed by using jdk 1.6.

CSMC components stack developed in .Net Framework 4.0 is deployed on each compute node as a WCF web service. One of the benefits we get from WCF is dynamic service binding. Instead of having a predefined binding between the compute nodes, successor and finger references, only generic binding is defined whose end points can be dynamically configured at the run time accordingly. Whereas, for Java based component stack dynamic binding is not possible and component stack has to load predefined binding between the successor and finger references. However both implementations of component stack support direct response to requester node once session information is identified in CAN.

To notify Resource Manger about resource utilization .Net based component stack uses PerformanceCounter class provided by System.Diagnostics namespace, whereas Java component stack uses java.lang.management package. Information about resource utilization is send back to Resource Manager periodically after every 15 seconds (but is configurable according to the requirement).

Since we are dealing with dynamic environment, where compute nodes can be added or removed from the cloud depending on resource requirement. Whenever new compute node is added to CAN, chord identifier is assigned to it by Resource Manager, and session values are allocated to it which falls within the range of assigned chord identifier and its successor's chord identifier. Once node is added to CAN and sessions values are assigned to it, Resource Manager sends an update finger table request to compute nodes in CAN. Every compute node maintained its finger table in a XML file. Each finger entry consists of chord identifier of a compute node and its IP address. Chord identifier is used in selecting the most suitable node during session lookup, whereas IP address is used to define the dynamic end points between the nodes for .net framework based component stack.

Session values are stored on individual nodes using SQL Server 2008 but is not limited to database. CSMC is configurable, depending on the requirement, file based session management can be used simply by configuring CSMC component stack to a file based session management. Different type of session management policies are handled by Session Manager, providing the abstraction for querying the underlying session management policy. Depending on the management policy Session Manager will create new sessions and retrieve desire session values from the configured medium. CSMC does not support main memory based session management policy, because CSMC is implemented as a web service; storing the entire session repository in main memory is a not a feasible solution.

VII. Experiments and Results

We examined CSMC behavior on three different configurations. At the very basic level we evaluated CSMC for modulo 2^m (where $m = 3$) chord space, Table I shows the number of compute nodes considered in this basic configuration. Each node's chord identifier is mentioned along with its finger table entries. Additionally we have added Hosted ID, indicating node responsible for storing the session in modulo 2^3 chord space, because Node-0, Node-3, Node-6, are not available in chord space.

For the medium sized cloud we considered modulo 2^m (where $m = 4$) chord space (see Table II). In total 9 compute nodes are used on which services are deployed along with the CSMC component stack. Sessions are distributed among the nodes in the modulo 2^4 chord space, if the desired compute node is missing then it successor is held responsible for storing and processing the sessions repository.

We have tested CSMC in modulo 2^m (where $m = 5$) chord space with the maximum capacity of 32 compute nodes. Table III shows the compute nodes, along with their finger table and hosted ID.

We have tested CSMC on three different test bed configurations (modulo 2^m where $m = 3, 4,$ *and* 5 respectively) explained earlier in this section. Session decoupling has been successfully tested in all of these configurations. The purpose of these experiments is to emphasize on the fact that CSMC outperform the conventional session management architecture irrespective of the size of cloud.

In total one million sessions values are distributed among the compute nodes modulo 2^m (where $m = 3, 4,$ *and* 5 respectively). Session object consists of a session identifier (*8 bytes*) and an user business object. User business object constitute of date of birth, gender, security credentials and time stamp of his last interaction with the system (*3, 1, 8 and 3 bytes respectively*). In total session object for a particular user consist of 23 bytes. On each compute node load is generated by periodically generating request through OpenSTA.

In Figure 5a best and worst case response time for session lookup is shown. In case of Chord the best case for key lookup is when lookup request is routed directly to the compute node that is responsible for persisting the values, without involving any intermediate request forwarding compute node. Whereas, the worst case in Chord is when lookup request is routed to the compute node that is multiple hops away from the actual compute node. These intermediates nodes will fractionally increase the session lookup time as they will try to forward the request to the compute node closest to the required compute node.

Figure 5b shows the average response time for three configurations. In case of first configuration, the average response time is greater than that of the other as the processing load on each service in much higher than that of the second and third configurations. Same is the case with second and third configurations. However, in all of these configurations we have achieved seamless service scaling without the need of replicating the session pool for every new instance of deployed service.

In order to demonstrate practicality of CSMC we further tested it on Apache Tomcat 6.0 and GlassFish Server 3.1.1. Figure 6a and 6b show the average session retrieval time for Tomcat and GlassFish respectively. Standard Java web services were deployed on these web-servers. In contrast to WCF .Net service standard Java web service does not support dynamic binding. Due to lack of dynamic binding both of these web-servers take more time to generate service response as compared to IIS 5.10. The core purpose of CSMC is to decouple active sessions from service instances; through these results we have shown that regardless of the underlying web-server session decoupling can be achieved.

With CSMC active sessions can be migrated within the cloud when services are scaled according to computational load and number of active users. By testing CSMC on three different web-servers we have shown that our proposed system is not confined to any specific web-server. Our evaluation results highlighted the fact that CSMC can be adopted by small as well as large scale cloud infrastructure. Through the experiments we have shown that for every web-server (i.e., IIS, Apache Tomcat and GlassFish Server) session retrieval time is directly proportional to number of computer nodes. Although CSMC can be used to achieved session decoupling for fewer number of compute nodes however its utility is certainly increased with higher number of compute nodes.

VIII. Future Work

CSMC can be used within private and public cloud to achieve seamless service migration within cloud. So far we have evaluated CSMC within private cloud, for service scaling (up or down) and session migration. However, as cloud services are becoming prevalent the concept of service migration between cloud is emerging. This concept is very important to cater vendor lock in problem. Migrating services between cloud would be a trivial task if the source and target cloud are using same web-servers. For our future work are planning to migrate services from our private to Amazon EC2. Service instance from our private cloud will be deployed on a EC2 compute node and active session will be distributed among EC2 instances using CSMC.

IX. Conclusion

Through CSMC, we have achieved session decoupling, enabling service provider to scale (up or down) services if required without the need to replicate existing active sessions. Besides this, whenever new compute node is added to cloud, CSMC automatically distributes the sessions among the compute nodes. CSMC eliminates the need of having a dedicated session state server, which would increase the service provisioning cost. The added advantage we get by applying Chord is the self-maintainability. Whenever new compute node is added or removed sessions are automatically distributed among the available resources.

CSMC evaluated on number of different configurations shows how compute nodes can be virtually deployed or

TABLE I.: Chord Space of modulo of 2^3 compute nodes

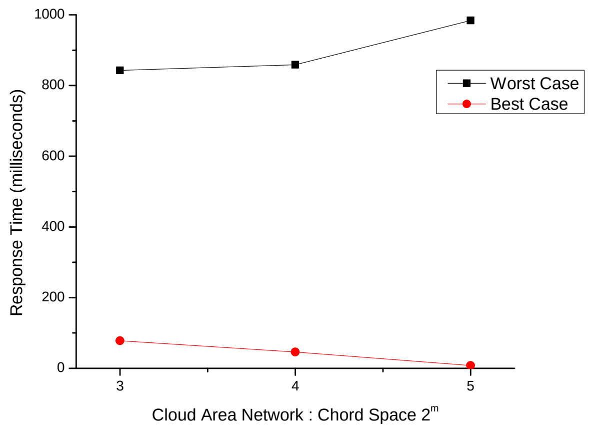
	Finger $(n+2(k+1))modulo\ 2^3$			
Chord Identifier (n)	$K=1$	$K=2$	$K=3$	Host Id
1	2	3	5	0,1
2	3	4	6	2
4	5	6	0	3,4
5	6	7	1	5
7	0	1	3	6,7

TABLE II.: Chord Space of modulo of 2^5 compute nodes

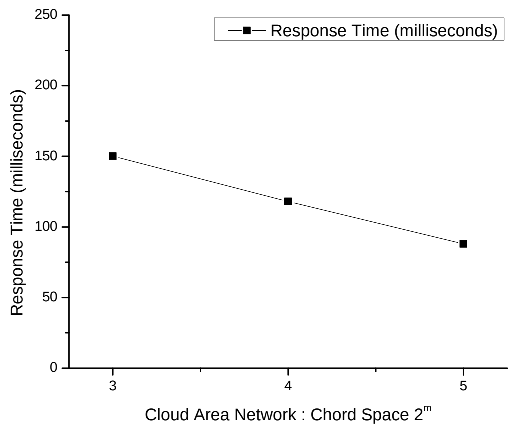
	Finger $(n+2(k+1))modulo\ 2^5$					
Chord Identifier (n)	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	Host Id
0	1	2	4	8	16	0
1	2	3	5	9	17	1
4	5	6	8	12	20	2,3,4
6	7	8	10	14	22	5,6
9	10	11	13	17	25	7,8,9
12	13	14	16	20	28	10,11,12
13	14	15	17	21	29	13
14	15	16	18	22	30	14
15	16	17	19	23	31	15
17	18	19	21	25	1	16,17
19	20	21	23	27	3	18,19
21	22	23	25	29	5	20,21
22	23	24	26	30	6	22
23	24	25	27	31	7	23
24	25	26	28	0	8	24
25	26	27	29	1	9	25
27	28	29	31	3	11	26,27
28	29	30	0	4	12	28
30	31	0	2	6	14	29,30
31	0	1	3	7	15	31

TABLE III.: Chord Space of modulo of 2^5 compute nodes

	Finger $(n+2(k+1))modulo\ 2^5$					
Chord Identifier (n)	$K=1$	$K=2$	$K=3$	$K=4$	$K=5$	Host Id
0	1	2	4	8	16	0
1	2	3	5	9	17	1
4	5	6	8	12	20	2,3,4
6	7	8	10	14	22	5,6
9	10	11	13	17	25	7,8,9
12	13	14	16	20	28	10,11,12
13	14	15	17	21	29	13
14	15	16	18	22	30	14
15	16	17	19	23	31	15
17	18	19	21	25	1	16,17
19	20	21	23	27	3	18,19
21	22	23	25	29	5	20,21
22	23	24	26	30	6	22
23	24	25	27	31	7	23
24	25	26	28	0	8	24
25	26	27	29	1	9	25
27	28	29	31	3	11	26,27
28	29	30	0	4	12	28
30	31	0	2	6	14	29,30
31	0	1	3	7	15	31

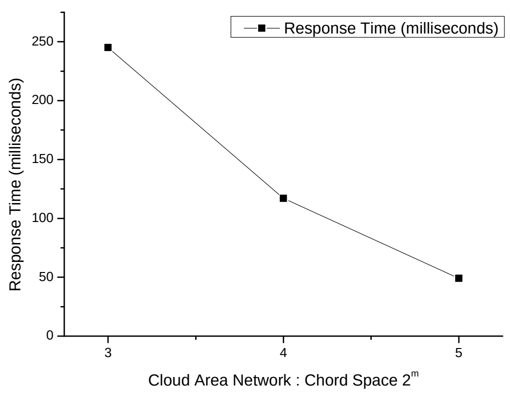


(a) Best and Worst Case

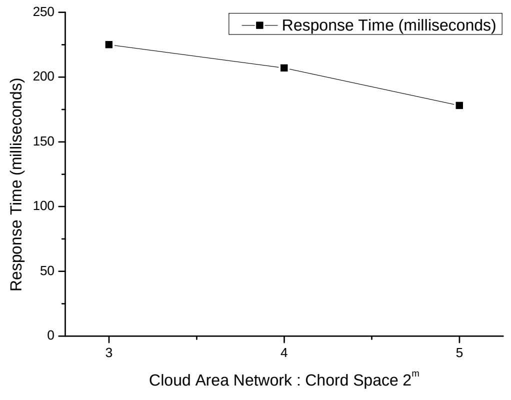


(b) Average Case

Figure 5: Session Lookup Time For 10,000 Session



(a) Apache Tomcat



(b) GlassFish Server

Figure 6: Average Session Lookup Time For 10,000 Sessions

reclaimed. Results shows that CSMC can be effectively utilized in varied size of cloud and number of concurrent users. CSMC enables service provider to develop multi-tenant services.

ACKNOWLEDGMENT

This research was fully supported by Microsoft Research Asia.

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, A. Katz, Randy Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zahari, “Above the clouds: A berkeley view of cloud computing,” UC Berkeley Reliable Adaptive Distributed Systems Laboratory, Tech. Rep., 2009.
[2] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, “Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility,” vol. 25. Amsterdam, The Netherlands, The Netherlands: Elsevier Science Publishers B. V., June 2009, pp. 599–616.
[3] W. Sun, K. Zhang, S.-K. Chen, X. Zhang, and H. Liang, “Software as a Service: An Integration Perspective,” 2009, pp. 558–569.
[4] L.-J. Zhang and Q. Zhou, “Ccoa: Cloud computing open architecture,” in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, july 2009, pp. 607 –616.
[5] “Google web applications for communication and collaborations.” [Online]. Available: http://www.google.com/apps
[6] “Windows azure platform.” [Online]. Available: http://www.microsoft.com/windowsazure
[7] M. Sato, “Creating next generation cloud computing based network services and the contributions of social cloud operation support system (oss) to society,” in *Enabling Technologies: Infrastructures for Collaborative Enterprises, 2009. WETICE ’09. 18th IEEE International Workshops on*, 29 2009-july 1 2009, pp. 52 –56.
[8] “Aws management console, a web-based interface to manage your services.” [Online]. Available: http://aws.amazon.com/console
[9] “Asp.net state management overview.” [Online]. Available: http://msdn.microsoft.com/en-us/library/75x4ha6s.aspx
[10] “The apache tomcat 5.5 servlet/jsp container, clustering/session replication how-to.” [Online].

Available: http://tomcat.apache.org/tomcat-5.5-doc/cluster-howto.html

[11] "Sun glassfish enterprise server v3 prelude developer's guide." [Online]. Available: http://docs.sun.com/app/docs/doc/820-4496/beaha?l=jaa=view.

[12] "State management overview," vol. Article ID: 307598. [Online]. Available: http://support.microsoft.com/kb/307598

[13] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160. [Online]. Available: http://doi.acm.org/10.1145/383059.383071

[14] G. Doyen, E. Nataf, and O. Festor, "A performance-oriented management information model for the chord peer-to-peer framework," in *Management of Multimedia Networks and Services*, ser. Lecture Notes in Computer Science, J. Vicente and D. Hutchison, Eds. Springer Berlin / Heidelberg, 2004, vol. 3271, pp. 29–49.

[15] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica, "Wide-area cooperative storage with cfs," *SIGOPS Oper. Syst. Rev.*, vol. 35, pp. 202–215, October 2001.

[16] S. Ajmani, D. E. Clarke, C. hue Moh, and S. Richman, "Conchord: Cooperative sdsi certificate storage and name resolution," in *In First International Workshop on Peer-to-Peer Systems*. Springer-Verlag, 2002, pp. 141–154.

[17] A. Binzenhöfer, G. Kunzmann, and R. Henjes, "Design and analysis of a scalable algorithm to monitor chord-based p2p systems at runtime," *Concurr. Comput. : Pract. Exper.*, vol. 20, pp. 625–641, April 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1358302.1358309

[18] Z. Jingling, X. Yonggang, and L. Qing, "Htc-chord: An improved chord model based on topic-cluster and hierarchic layer," in *Broadband Network Multimedia Technology, 2009. IC-BNMT '09. 2nd IEEE International Conference on*, oct. 2009, pp. 655 –658.

[19] S. Ktari, A. Hecker, and H. Labiod, "Power-law chord architecture in p2p overlays," in *Proceedings of the 2008 ACM CoNEXT Conference*, ser. CoNEXT '08. New York, NY, USA: ACM, 2008, pp. 39:1–39:2. [Online]. Available: http://doi.acm.org/10.1145/1544012.1544051

[20] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol – http/1.1," oct. 1999, pp. 655 –658. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[21] J. J. Garrett, "Ajax: A new approach to web applications." [Online]. Available: http://www.adaptivepath.com/ideas/essays/archives/000385.php

[22] "Open system testing architecture," 2003. [Online]. Available: http://www.opensta.org