A Cloud Computing Caching Service
Rampur Srinath¹, Prince Kumar Singh², Bhawesh Jha³, Aditya Srivastava⁴ and Aakash Deep Sarkar⁵

¹Associate Professor, Dept. of ISE, National Institute of Engineering Mysore
²,³,⁴,⁵Information Science and Engineering, National Institute Of Engineering Mysore

Abstract—The basic function of Web services is standardizing how services expose their data and interfaces, and how applications can discover and interact with them the response time of Web services is relatively long because of the encoding and parsing processes. This problem is worth investigating, with the aim of improving performance especially when certain users in the CLOUD-ACCESS share common interests due to physical and temporal locality. Developing a caching strategy for the whole CLOUD-ACCESS allows improving the performance of the network as a whole. This project proposes a distributed model for caching Web service method signatures and call responses in CLOUD-ACCESSs. This project helps to enable nodes joining the network to take advantage of Web services already consumed by the network and to contribute by consuming additional services so that they would be used by other nodes.

Keywords—Cache, Cloud Computing, Clustering, Artificial Bee Colony

I. INTRODUCTION

1.1. Overview
When a PC receives a method call, it searches sequentially for a match in the RDs (Each DBMS identified on the basis of its location in the network is called an RD-node.) to determine if the response is cached in the network. This system, eliminates the sequential search process. Aside from the architectural changes, this work provides comprehensive mathematical analysis that studies the theoretical behavior of the system, and presents experimental results that examine its performance.

1.2. Purpose
In CLOUD-ACCESS environments, mobile computing devices can invoke Web services methods to gain access to needed data, like stock quotes, currency exchange rates, etc. However, in addition to the communication costs, device mobility can cause temporary loss of connectivity to the server, thus rendering needed services are inaccessible during those periods. For such reasons, caching of Web service responses within the CLOUD-ACCESS can become beneficial for increasing data availability and reducing delays. This project describes a distributed system designed for CLOUD-ACCESS environments to cache proxies of consumed Web services and responses of their invoked methods.

II. PROBLEM DEFINATION

2.1 Existing System:
In the previous system Request Directories (RD) are used for caching the response. RD is totally different from the Proxy Cache (PC) which is solely responsible for running the proxy. When a PC receives a method call, it searches sequentially in the RD.

2.2 Proposed system:
In the proposed system work of RD and PC are combined to eliminate the sequential process.

Key terms are listed as follows
Here service proxy is created in the network to make the request to the server on behalf of the client. PC is referred as the service proxy that will search the Universal Description Discovery and Integration (UDDI) that contains the information about requested services. Node in the network that invokes a web service and gets response will cache to make it available for whole network. Such a node is called Caching Nodes (CN).

Cloud computing offers a prominent service for data storage known as cloud storage. The flow and storage of data on the cloud environment in plain text format may be main security threat. So, it is the responsibility of cloud service providers to ensure privacy and security of data on storage as well as network level. The following three parameters confidentiality, integrity and availability decide whether security and privacy of data stored on cloud environment is maintained or not. Cloud computing is a distributed computing style which offer integration of web services and data centers. There are several major cloud computing providers including Amazon, Google, Yahoo, Microsoft and others that are providing cloud computing services. Amazon web services was first to provide an architecture for cloud based services in 2002 and after that advancements and new models for cloud architecture had been proposed and implemented. There have been many techniques of storing data on server storage. Such data storages provided by cloud service providers have to ensure client about Confidentiality, Integrity and Availability of data.

Confidentiality: Confidentiality refers to keeping data private. Privacy is of importance as data leaves the borders of the owner. Confidentiality is supported by technical tools such as encryption and access control, as well as legal protection.

Integrity: Integrity is a degree of confidence that what data is supposed to be in cloud, what is actually there, and is protected against accidental or intentional alteration without authorization.

Availability: Availability means being able to use the system as anticipated by cloud user. Cloud technologies can increase availability through widespread internet-enabled access, but the client is dependent on the timely and robust provision of resources. Availability is supported by capacity building and good architecture by the provider, as well as well-defined contracts and terms of agreement. Cloud data storage security addresses the need of enforcing selective data access by providing an approach that supports the user in specification of access restrictions and security measures.

III. DESIGN MODELS

4.1 Architectural Diagram for the Proposed System

A service proxy is created if not present in the network to make the requests to the server on behalf of the client. There are nodes in our architecture that run the proxies of the services, and we refer to them as Proxy Caches (PCs). To create the proxy of an unknown Cloud service, given some keywords, the PC node will search the Universal Description, Discovery, and Integration (UDDI) registry, which contains information that helps locating the requested service. A mobile device (node) that invokes a Cloud service method and gets the response from the server will cache this response to make it available for the whole network. Such nodes will be referred to as Caching Nodes (CNs). This allows a node that requests the same data later to get it from the MANET. Fig. 5.1 shows a general diagram illustrating the components of the system.
4.2 System Operations

When a node joins the network, it floods a HELLO packet containing its address to inform the nodes within the network about its presence. Each node in the network calculates and stores a special score that summarizes its resource capabilities, including expected time during Caching service proxy and getting data from remote server which the device is in the MANET, battery life, and available bandwidth and memory. To be considered a candidate PC, the device must meet a minimum criterion in each category. That is, the value of each resource (i.e., time, battery life, bandwidth, and memory) must be above an empirically set threshold. Nodes that do not qualify like small devices and sensor nodes can still play an active role in the architecture by being CNs. The scores of the nodes are collected every time a new PC node is needed to host a service proxy. The details of the PC election algorithm are illustrated in subsection.

4.3 Flowchart and Control Flow Diagram

In this section we are going to explain the execution of an application with Flowchart and Control Flow Diagram towards cloud service request and caching method condition.
The fig. used to explain entire project flow. The diagram created with numbering sequenced from 1, 2, 3….10 in the order of their execution, which is explained in the following section.

1) User request for a service
2) Decision is made dynamically to retrieve CSDL and DISCO file from the local or from neighboring machines or from the web server.
3) Store the copy of CSDL and DISCO file in the local machine
4) Get URI to call the service from the DISCO file
5) Retrieve Arguments for the service stored in CSDL
6) Find the service cached from the neighboring nodes for faster response to the user
7) If found, in the neighboring nodes then display result and cache in the local machine
8) If not found, Dynamic Invocation for the service.
9) Response from the server will be HTTP-XML based response, which requires parsing and extracting result.
10) Extracted information shown to the user and also cached in the local machine for referring in future.
11) The above steps are simple in words for the entire project execution for faster response on cloud service invocation, which is implemented taking care of reusability, less memory consumption, dynamic decision making and code full of self documented comments for easy analyzing.

IV. IMPLEMENTATION AND EVALUATION

We evaluated CC against Adaptive Replacement Cache (ARC), Clock, First-in-first-out (FIFO), Greedy Dual (GD), Greedy Dual Size Frequency (GDSF), Hybrid, Least Frequently Used (LFU), Low Inter-reference Recency Set (LIRS), Least Recently Used (LRU), Most Recently Used (MRU), Two Queue (2Q) and Random Replacement (RR) cache management policies using cache-hit rate, cache acceptance count and cache-rejection count as metrics. To carry out the evaluation, we implemented CC within a simulation environment.

4.1 Simulation Environment

We installed the software on 8 computers running windows 10 with 4 GB RAM and 500 GB Secondary storage. We connected the systems through MANNET and borrowed two web services namely Currency Converter and Weather Forecast from webservicex.net to run the simulations.

4.2 Simulation Results

The result is divided into 3 columns. First column gives the average time to fetch result directly from cloud. The second column gives the time to fetch for the same service with same parameters from a neighboring node. The third column gives the fetch time for the same service but with different parameters from a neighboring node. The result is shown in table 4.2.

<table>
<thead>
<tr>
<th>Web service</th>
<th>Average Time from cloud (in ms)</th>
<th>Average Time Neighbouring node(same parameters) ms</th>
<th>Average Time Neighbouring node(Different parameters) ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Currency exchange</td>
<td>3844</td>
<td>4</td>
<td>108</td>
</tr>
<tr>
<td>Weather forecast</td>
<td>984</td>
<td>9</td>
<td>108</td>
</tr>
</tbody>
</table>

As can be seen from the table the efficiency is much higher when the data is being fetched from the neighboring node than from cloud.

V. CONCLUSION

We propose taking a community-based caching (CC) approach. CC manages caching as a cloud services intelligently. It eliminates cache pollution and minimizes monopoly problems inherent in other cache management policies, improving overall cache-hit rate. In our experiment, the cache-hit rate achieved by CC was between 0.7% and 55% better than that of other tested cache policies. Our t-Tests demonstrate that these improvements are statistically significant. The simulation results and tests of statistical significance show that CC manages caches intelligently, achieving extra value for a server providing caching as a cloud service. Future investigations should focus on how the size of historic data used in CC and the pre-set community count affect cache-hit rate.

REFERENCES


