International Journal of Engineering Research and Development
e-ISSN: 2278-067X, p-ISSN: 2278-800X, www.ijerd.com
Volume 10, Issue 8 (August 2014), PP.58-63

Ascendable Architecture for Wandering Services in Social Networks

[1]Potturi Reshma, [2]Dr.B.Srinivasarao
[1]PG Student, [2]Professor & HOD, Department of CSE Dhanekula Institute of Engineering & Technology Ganguru, Vijayawada-39

Abstract:- A mobile ubiquity services is an important element of cloud computing environments, for the reason it keeps an up-to-date list of presence information of mobile user. If presence updates occur often the number of messages distributed by presence server may lead to scalability problem and buddy list search problem in large-scale mobile presence services. To overcome the scalability problem proposed an efficient and ascendable server architecture called presence cloud. It organizes the presence server in to quorum based server-server architecture for efficient searching. When a mobile user joins a network or internet, presence cloud searches the presence information. It also achieves small constant search latency by the directed search algorithm and one-hop caching strategy. Anatomize the performance of presence cloud in terms of search cost and search satisfaction level, without compromising each other.

Keywords:- Mobile ubiquity services, presence cloud, one-hop cache, latency, buddy list.

I. INTRODUCTION

Instant messaging (IM) and internet chat communication have seen enormous growth over the last several years. Mobile devices and cloud computing environments can provide presence-enabled applications, i.e., social network applications/services, worldwide. Facebook, Twitter Foursquare, Google Latitude, buddy cloud and Mobile Instant Messaging (MIM) are examples of presence-enabled applications that have grown rapidly in the last decade. Social network services are changing the ways in which participants engage with their friends on the Internet. The sharing of basic presence information can result in a large volume of traffic as users log on or off throughout the life of a presence session, especially for users with large numbers of contacts (e.g., the author of this document has over 1,700 contacts in his presence-enabled contact list). The volume is increased by communication of information beyond basic on-off network availability, such as availability (e.g., "away" and "do not disturb"). The volume is further increased if the presence "transport" is used to communicate information such as device capabilities, geolocation, mood, activity, even the music to which a user is listening. A mobile presence service is an essential component of social network services in cloud computing environments. The key function of a mobile presence service is to maintain up-to-date list of presence information of all mobile users. The presence information includes details about a mobile user's location, availability, activity, device capability, and preferences. The service must also bind the user's ID to his/her current presence information, as well as retrieve and subscribe to changes in the presence information of the user's friends. In social network services, each mobile user has a friend list, typically called a buddy list, which contains the contact information of other users that he/she wants to communicate with. The mobile user's status is broadcast automatically to each person on the buddy list whenever he/she transits from one status to the other. For example, when a mobile user logs into a social network application, such as an IM system, through his/her mobile device, the mobile presence service searches for and notifies everyone on the user's buddy list. To maximize a mobile presence service's search speed and minimize the notification time, most presence services use server cluster technology. To improve the efficiency of the search operation, PresenceCloud requires a caching strategy to replicate presence information of users. In order to adapt to changes in the presence of users, the caching strategy should be asynchronous and not require expensive mechanisms for distributed agreement. In PresenceCloud, each PS node maintains a *user list* of presence information of the attached users, and it is responsible for caching the *user list* of each node in its PS list, in other words, PS nodes only replicate the *user list* at most one hop away from itself. The cache is updated when neighbours establish connections to it, and periodically updated with its neighbours. Therefore, when a PS node receives a query, it can respond not only with matches from its own *user list*, but also provide matches from its caches that are the user lists offered by all of its neighbours. In this section, we describe previous researches on presence services, and survey the presence service of existing systems. Well known commercial IM systems leverage some form of centralized clusters to provide presence services. Jennings III *et al.* presented taxonomy of different features and functions supported by the three most popular IM systems, AIM, Microsoft MSN and Yahoo! Messenger. The authors also provided an overview of the system architectures and observed that the systems use client-server-

based architectures. Skype, a popular voice over IP application, utilizes the Global Index (GI) technology to provide a presence service for users. GI is a multi-tiered network architecture where each node maintains full knowledge of all available users. Since Skype is not an open protocol, it is difficult to determine how GI technology is used exactly. Moreover, Xiao *et al.* analyzed the traffic of MSN and AIM system. They found that the presence information is one of most messaging traffic in instant messaging systems. In, authors shown that the largest message traffic in existing presence services is buddy NOTIFY messages. Recently, there is an increase amount of interest in how to design a peer-to-peer SIP. P2PSIP has been proposed to remove the centralized server, reduce maintenance costs, and prevent failures in server-based SIP deployment. To maintain presence information, P2PSIP clients are organized in a DHT system, rather than in a centralized server. However, the presence service architectures of Jabber and P2PSIP are distributed, the *buddy-list search problem* we defined later also could affect such distributed systems. It is noted that few articles in discuss the scalability issues of the distributed presence server architecture. Saint Andre analyzes the traffic generated as a result of presence information between users of inter-domains that support the XMPP. Houri *et al.* Show that the amount of presence traffic in SIMPLE can be extremely heavy, and they analyze the effect of a large presence system on the memory and CPU loading. Those works in study related problems and developing an initial set of guidelines for optimizing inter-domain presence traffic and present DHT-based presence server architecture.

II. DESIGN OF PRESENCECLOUD

The past few years has seen a veritable frenzy of research activity in Internet-scale object searching field, with many designed protocols and proposed algorithms. Most of the previous algorithms are used to address the fixed object searching problem in distributed systems for different intentions. However, people are nomadic, the mobile presence information is more mutable and dynamic; anew design of mobile presence services is needed to address the buddy-list search problem, especially for the demand of mobile social network applications. PresenceCloud is used to construct and maintain distributed server architecture and can be used to efficiently query the system for buddy list searches. Presence Cloud consists of three main components that are run across a set of presence servers. In the design of Presence Cloud, the ideas of P2P systems and present a particular design for mobile presence services
has been refined. The three key components of Presence Cloud are summarized below:

- **Presence Cloud server overlay:** It organizes presence servers based on the concept of grid quorum system. So, the server overlay of PresenceCloud has a balanced load property and a two-hop diameter node degrees, where n is the number of presence servers.
- **One-hop caching strategy:** It is used to reduce the number of transmitted messages and accelerate query speed. All presence servers maintain caches for the buddies offered by their immediate neighbours.
- **Directed buddy search:** It is based on the directed search strategy. PresenceCloud ensures an one-hop search, it yields a small constant search latency on average.

2.1 Presence Cloud Overview

The primary abstraction exported by our PresenceCloud issued a scalable server architecture for mobile presence services, and can be used to efficiently search the desired buddy lists. We illustrated a simple overview of Presence Cloud in Fig. 1. In the mobile Internet, a mobile user can access the Internet and make a data connection to Presence Cloud via 3G or Wifi services. After the mobile user joins and authenticates himself/herself to the mobile presence service, the mobile user is determinately directed to one of Presence Servers in the Presence Cloud by using the Secure Hash Algorithm, such as SHA-1. The mobile user opens a TCP connection to the Presence Server (PSnode) for control message transmission, particularly for the presence information. After the control channel is established, the mobile user sends a request to the connected PSnode for his/her buddy list searching. Our PresenceCloud shall do an efficient searching operation and return the presence information of the desired buddies to the mobile user. Now, we discuss the three components of Presence-Cloud in detail below.
Fig 1. Architecture for presence cloud

2.2 Presence Cloud Server Over relay

The Presence Cloud server overlay construction algorithm organizes the PS nodes into a server-to-server overlay, which provides a good low-diameter overlay property. The low-diameter property ensures that a PS node only needs two hops to reach any other PS nodes.

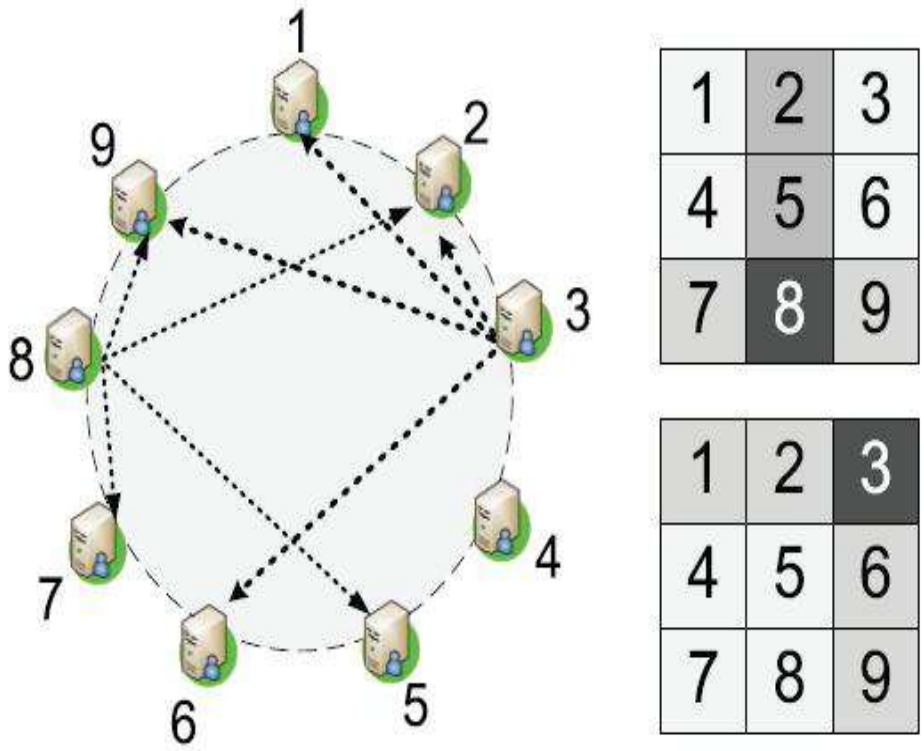

Fig 1. A perspective of Presence Cloud Server Overlay

Algorithm1. Presence Cloud Stabilization algorithm
1: /* periodically verify PS node n's pslist */
2: Definition:
3: pslist: set of the current PS list of this PS node, n
4: pslist[].connection: the current PS node in pslist
5: pslist[].id: identifier of the correct connection in pslist
6: node.id: identifier of PS node node
7: Algorithm:
8: r ☐Sizeof(pslist)
9: for i = 1 to r do
10: node ☐pslist[i].connection
11: if node.id ≠pslist[i].id then
12: /* ask node to refresh n's PS list entries */
13: findnode☐Find_CorrectPSNode(node)
14: if findnode=nil then
15: pslist[i].connection☐RandomNode(node)
16: else
17: pslist[i].connection☐findnode
18: end if
19: else
20: /* send a heartbeat message */
21: bfailed☐SendHeartbeatmsg(node)
22: if bfailed= true then
23: pslist[i].connection☐RandomNode(node)
24: end if
25: end if
26: end for

Our algorithm is fault tolerance design. At each PS node, a simple Stabilization () process periodically contacts existing PS nodes to maintain the PS list. The Stabilization () process is elaborately presented in the Algorithm. When a PS node joins, it obtains its PS list by contacting a root. However, if a PS node n detects failed PS nodes in its PS list, it needs to establish new connections with existing PS nodes. In our algorithm, n should pick a random PS node that is in the same column or row as the failed PS node.

2.3 One-hop caching strategy

To improve the efficiency of the search operation, Presence Cloud requires a caching strategy to replicate presence information of users. In order to adapt to changes in the presence of users, the caching strategy should be asynchronous and not require expensive mechanisms for distributed agreement. In Presence Cloud, each PS node maintains a *user list* of presence information of the attached users, and it is responsible for caching the *user list* of each node in its PS list, in other words, PS nodes only replicate the *user list* at most one hop away from itself. The cache is updated when neighbours establish connections to it, and periodically updated with its neighbours. Therefore, when a PS node receives a query, it can respond not only with matches from its own *user list*, but also provide matches from its caches that are the user lists offered by all of its neighbours.

2.4 Directed buddy search

We contend that minimizing searching response time is important to mobile presence services. Thus, the buddy list searching algorithm of Presence Cloud coupled with the two-hop overlay and one-hop caching strategy ensures that Presence Cloud can typically provide swift responses for a large number of mobile users. First, by organizing PS nodes in a server-to-server overlay network, we can therefore use one-hop search exactly for queries and thus reduce the network traffic without significant impact on the search results. Second, by capitalizing the one-hop caching that maintains the user lists of its neighbours, we improve response time by increasing the chances of finding buddies. Clearly, this mechanism both reduces the network traffic and response time. Based on the mechanism, the population of mobile users can be retrieved by a broadcasting operation in any PS node in the mobile presence service. Moreover, the broadcasting message can be piggybacked in a buddy search message for saving the cost.

Directed buddy search algorithm:
1. A mobile user logins PresenceCloud and decides the associated PS node, q.
2. The user sends a Buddy List Search Message, B to the PS node q.
3. When the PS node q receives a B, then retrieves each bi from B and searches its user list and one-hop cache to respond to the coming query. And removes the responded buddies' from B.
4. If B = nil, the buddy list search operation is done.
5. Otherwise, if B =nil, the PS node q should hash each remaining identifier in B to obtain a grid ID, respectively.
6. Then, the PS node q aggregates these b(g) to become a new B(j), for each g Sj. Here, PS node j is the intersection node of Sq intersection Sg. And sends the new B(j) to PS node j.

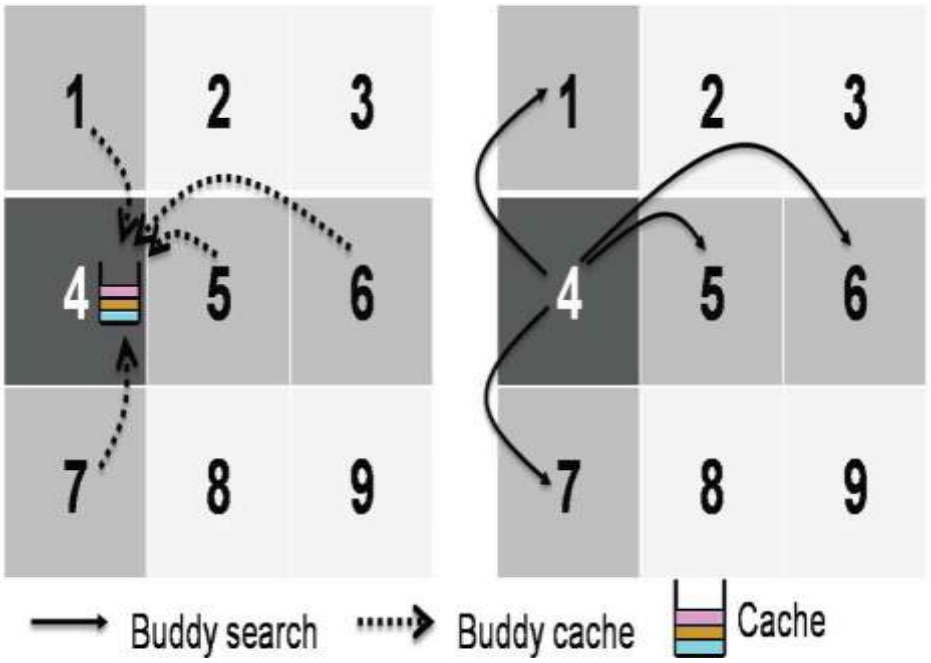

Fig 2. Directed buddy search

When a PS node 4 receives a Buddy List Search Message, B (1; 2; 3; 4; 5; 6; 7; 8; 9), from a mobile user, PSnode 4 first searches its local user list and the buddy cache, and then it responds these searched buddies to the mobile user and removes these searched buddies from B. These removed buddies include the user lists of PS node {1,4,5,6,7}.

Then, PS node 4 can aggregates b and b to become a new Bð6Þ and sends the new B to PS node 6. Note that the ps list Id of PS node 6 is {3,4,5,9}. Here, PS node 4 also aggregates b and b to become a new and sends the new Bð5Þ to PSnode 5. However, due to the one-hop caching strategy, PS node 6 has a buddy cache that contains these user lists of PSnode {3,9}, PS node 6 can expeditiously respond the buddy search message . Consequently, the directed searching combined with both previous two mechanisms, including Presence Cloud server overlay and one-hop caching strategy, can reduce the number of searching messages sent.

Our caching strategy does not require expensive overhead for presence consistency among PS nodes. When a mobile user changes its presence information, either because it leaves Presence Cloud, or due to failure, the responded PS node can disseminate its new presence to other neighbouring PS nodes for getting updated quickly.

III. COST ANALYSIS

A cost analysis of the communication cost of PresenceCloud in terms of the number of messages required to search the buddy information of a mobile user. Note that how to reduce the number of inter server communication messages is the most important metric in mobile presence service issues. The buddy-list search problem can be solved by a brute-force search algorithm, which simply searches all the PS nodes in the mobile presence service. In a simple mesh-based design, the algorithm replicates all the presence information at each PS node; hence its search cost, denote by QMesh, is only one message. On the other hand, the system needs n _ 1 messages to replicate a user's presence information to all PS nodes, where n is the number of PS nodes. The communication cost of searching buddies and replicating presence information can be formulated as Mcost = QMesh +RMesh, where RMesh is the communication cost of replicating presence information to all PS nodes. Accordingly, we have Mcost = O(n).

In the analysis of Presence Cloud, we assume that the mobile users are distributed equally among all the PSnodes, which is the worst case of the performance of Presence- Cloud. Here, the search cost of Presence Cloud is denoted as Qp, which is messages for both searching buddy lists and replicating presence information. Because search message and replica message can be combined into one single message, the communication cost of replicating, Rp (0).

It is straight forward to know that the communication cost of searching buddies and replicating presence information in Presence Cloud is Pcost .However, in Presence Cloud, a PS node not only searches a buddy list and replicates presence information, but also notifies users in the buddy list about the new presence event. Let b be the maximum number of buddies of a mobile user. Thus, the worst case is when none of the buddies are registered with the PS nodes reached by the search messages and each user on the buddy list is

located on different PS nodes. Since Presence Cloud must reply every online user on the buddy list individually, it is clear that extra b messages must be transmitted. In the worst case, it needs other messages When all mobile users are distributed equally among the PS nodes, which is considered to be the worst case, the Pcost.

IV. CONCLUSION

A scalable server architecture that supports mobile presence services in large-scale social network services. Presence Cloud achieves low search latency and enhances the performance of mobile presence services. Total number of buddy search messages increases substantially with the user arrival rate and the number of presence servers. The growth of social network applications and mobile device computing capacity to explore the user satisfaction both on mobile presence services or mobile devices. Presence Cloud could certificate the presence server every time when the presence server joins to Presence Cloud. The results of that Presence Cloud achieve performance gains in the search cost without compromising search satisfaction.

REFRENCES

[1]. R.B. Jennings, E.M. Nahum, D.P. Olshefski, D. Saha, Z.-Y. Shae, and C. Waters, "A Study of Internet Instant Messaging and ChatProtocols," IEEE Network, vol. 20, no. 6, pp. 16-21, July/Aug. 2006.

[2]. Z. Xiao, L. Guo, and J. Tracey, x"Understanding Instant Messaging Traffic Characteristics," Proc. IEEE 27th Int'l Conf. Distributed Computing Systems (ICDCS), 2007.

[3]. Chi, R. Hao, D. Wang, and Z.-Z. Cao, "IMS Presence Server: Traffic Analysis and Performance Modelling," Proc. IEEE Int'Conf. Network Protocols (ICNP), 2008.

[4]. Instant Messaging and Presence Protocol IETF Working Group, http: //www.ietf.org/html.charters/impp-charter.html, 2012.

P.Reshma did her B.Tech in cse, jntu Kakinada and pursuing her M.Tech in CSE from Dhanekula Institute of engg. And technology, jntu kakinada. Her area interest is moblile computing and software engg.

Dr.B.srinivasarao did his B.Tech in CSE form Nagarjuna University,M.Tech in CSE from JNTU Anantapur and PhD in CSE from Acharya Nagarjuna University . Currently working as Professor and HOD in Dhanekula Institute of engg. And Technology.