A Scalable Server Architecture Using PresenceCloud for Social Network Applications

Vijay Kumar A S
Dept of Computer Science and Engineering
AMC Engineering College
Bangalore, India
viju.arehalli@gmail.com

Mrs. Jeevitha R Assistant Professor
Dept of Computer Science and Engineering
AMC Engineering College
Bangalore, India
jeevitha.cute@gmail.com

***Abstract*- Social network applications are becoming increasingly popular on mobile devices. A mobile presence service is an essential component of a social network application because it maintains each mobile user's presence information, such as the current status (online/offline), GPS location and network address, and also updates the user's online friends with the information continually. If presence updates occur frequently, the enormous number of messages distributed by presence servers may lead to a scalability problem in a large-scale mobile presence service. To address the problem, we propose efficient and scalable server architecture, called PresenceCloud, which enables mobile presence services to support large-scale social network applications. When a mobile user joins a network, PresenceCloud searches for the presence of his/her friends and notifies them of his/her arrival. PresenceCloud organizes presence servers into a quorum-based server-to-server architecture for efficient presence searching. It also leverages a directed search algorithm and a one-hop caching strategy to achieve small constant search latency. We analyze the performance of PresenceCloud in terms of the search cost and search satisfaction level. The search cost is defined as the total number of messages generated by the presence server when a user arrives; and search satisfaction level is defined as the time it takes to search for the arriving user's friend list. The results of simulations demonstrate that PresenceCloud achieves performance gains in the search cost without compromising search satisfaction.**

Keywords*- *Social networks, mobile presence services, distributed presence servers.

I. INTRODUCTION

Because of the ubiquity of the Internet, mobile devices and cloud computing environments can provide presence-enabled applications, *i.e.*, social network applications/services, worldwide. Facebook, Twitter, Foursquare, Google Latitude, and Mobile Instant Messaging (MIM), are examples of presence-enabled applications that have grown rapidly in ways in which participants engage with their friends on the Internet. They exploit the information about the status of participants including their appearances and activities to interact with their friends. Moreover, because of the wide availability of mobile devices (e.g., Smart phones) that utilize wireless mobile network technologies, social network services enable participants to share live experiences instantly across great distances. For example, Facebook receives more than 25 billion shared items every month and Twitter receives more than 55 million tweets each day.

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 an 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. Currently, more than 500 million people use social network services on the Internet. Given the growth of social network applications and mobile network capacity, it is expected that the number of mobile presence service users will increase substantially in the near future. Thus, a scalable mobile presence service is deemed essential for future Internet applications.

In the last decade, many Internet services have been deployed in distributed paradigms as well as cloud computing applications. For example, the services developed by Google and Facebook are spread among as many distributed servers as possible to support the huge number of users worldwide. Thus,

we explore the relationship between distributed presence servers and server network topologies on the Internet, and propose an efficient and scalable server to server overlay architecture called PresenceCloud to improve the efficiency of mobile presence services for large-scale social network services.

First, we examine the server architectures of existing presence services, and introduce the buddy-list search problem in distributed presence architectures in large-scale geographically data centers. The *buddy-list search problem* is a scalability problem that occurs when a distributed

Presence service is overloaded with buddy search messages. Then, we discuss the design of PresenceCloud, a scalable server-to-server architecture that can be used as a building block for mobile presence services. The rationale behind the design of PresenceCloud is to distribute the information of millions of users among thousands of presence servers on the Internet. To avoid single point of failure, no single presence server is supposed to maintain service-wide global information about all users. PresenceCloud organizes presence servers into a quorum-based server-to-server architecture to facilitate efficient buddy list searching. It also leverages the server overlay and a directed buddy search algorithm to achieve small constant search latency; and employs

active caching strategy that substantially reduces

the number of messages generated by each search for a list of buddies. We analyze the performance complexity of PresenceCloud and two other architectures, a Mesh-based scheme and a Distributed Hash Table (DHT)-based scheme.

Through simulations, we also compare the performance of the three approaches in terms of the number of messages generated and the search satisfaction which we use to denote the search response time and the buddy notification time. The results demonstrate that PresenceCloud achieves

major performance gains in terms of reducing the number of messages without sacrificing search satisfaction. Thus, PresenceCloud can support a large-scale social network service distributed among thousands of servers on the Internet.

The contribution of this paper is threefold. First, PresenceCloud is among the pioneering architecture for mobile presence services. To the best of our knowledge, this is the first work that explicitly designs a presence server architecture that significantly outperforms those based distributed hash tables. PresenceCloud can also be utilized by Internet social network applications and services that need to replicate or search for mutable and dynamic data among distributed presence servers. The second contribution is that we analyze the scalability problems of distributed presence server architectures, and define a new problem called the buddy-list search problem. Through our mathematical formulation, the scalability problem in the distributed server architectures of mobile presence services is analyzed. Finally, we analyze the performance complexity of Presence Cloud and different designs of distributed architectures, and evaluate them empirically to demonstrate the advantages of PresenceCloud.

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; a new 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 a distributed server architecture and can be used to efficiently query the system for buddy list searches. PresenceCloud consists of three main components that are run across a set of presence servers. In the design of PresenceCloud, we refine the ideas of P2P systems and present a particular design for mobile presence services. The three key components of PresenceCloud are summarized below:

- *PresenceCloud server overlay* 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 with $O(\sqrt{n})$ node degrees, where n is the number of presence servers
- *One-hop caching strategy* 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 neighbors.
- *Directed buddy search* is based on the directed search strategy. PresenceCloud ensures an one-hop search, it yields a small constant search latency on average.

III. PRESENCECLOUD OVERVIEW

The primary abstraction exported by our PresenceCloud is used to construct 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

PresenceCloud in Fig.1. In the mobile Internet, a mobile user can access the Internet and make a data connection to PresenceCloud via 3G or Wi-Fi 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 PresenceCloud by using the Secure Hash Algorithm, such as SHA-1. The mobile user opens a TCP connection to the Presence Server (PS node) for control message transmission, particularly for the presence information. After the control channel is established, the mobile user sends a request to the connected PS node 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 PresenceCloud in detail below.

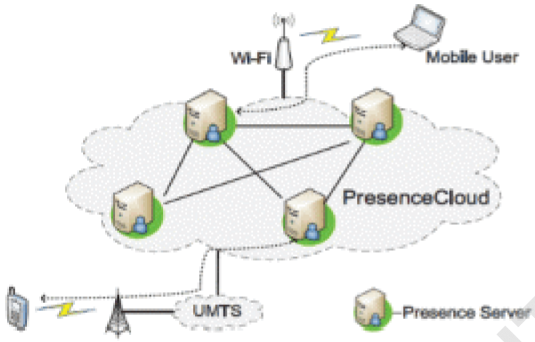

Fig 1: An overview of PresenceCloud

IV. PresenceCloud Server Overlay

The PresenceCloud 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. The detailed description is as follows.

Our PresenceCloud is based on the concept of *grid quorum system*, where a PS node only maintains a set of PS nodes of size $O(\sqrt{n})$, where n is the number of PS nodes in mobile presence services. In a PresenceCloud system, each PS node has a set of PS nodes, called *PS list*, that constructed by using a grid quorum system, shown in Fig.2 for n=9. The size of a grid quorum is $\sqrt{n}\times\sqrt{n}$. When a PS node joins the server overlay of PresenceCloud, it gets an ID in the grid, locates its position in the grid and obtains its PS list by contacting a root server1. On the $\sqrt{n} \times \sqrt{n}$ grid, a PS node with a grid ID can pick one column and one row of entries and these entries will become its PS list in a PresenceCloud server overlay. Fig.2 illustrates an example of PresenceCloud, in which the grid quorum is set to $\sqrt{9} \times \sqrt{9}$. In the Fig. 2, the PS node 8 has a PS list {2,5,7,9} and the PS node 3 has a PS list {1,2,6,9}. Thus, the PS node 3 and 8 can construct their overly networks according to their PS lists respectively.

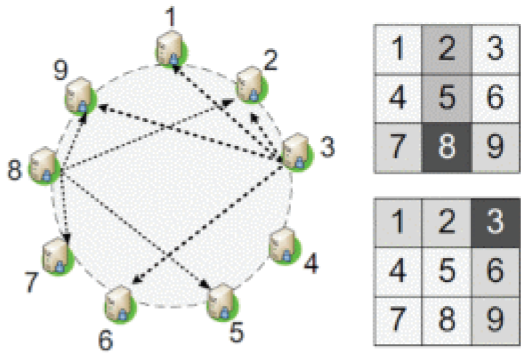

Fig2: A perspective of PresenceCloud Server Overlay

PresenceCloud 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

7: Algorithm:

8: $r \leftarrow$ 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

In Algorithm1 the function Find_CorrectPSNode() returns corrects list entry ie findnode:id is equal to *pslist*[*i*]*: id*. The function RandomNode(*node*) is designed to pick a random PS node that is in the same column or row as the failed PS node, *node*. This function can retrieve substituted nodes by asking the existing PS node in PS list.

The heart beat messages overhead is another performance factor in distributed server overlays. Anyone distributed server overlay architecture requires heart beat message to maintain the connectivity of each server for recovering system from server failure. However, in order to reduce

the maintenance maintain the connectivity of each server for recovering system from server failure. However, in order to reduce the maintenance overhead, PresenceCloud piggybacks the heart beat message in buddy search messages for saving transmission costs.

V. ONE-HOP CACHING

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 PresenceServer [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 neighbors establish connections to it, and periodically updated with its neighbors. 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 neighbors.

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 PresenceCloud, or due to failure, the responded PS node can disseminate its new presence to other neighboring PS nodes for getting updated quickly. Consequently, this one-hop caching strategy ensures that the user's presence information could remain mostly up-to-date and consistent throughout the session time of the user.

VI. DIRECTED BUDDY SEARCH

We contended that minimizing searching response time is important to mobile presence services. Thus, the buddy list searching algorithm of PresenceCloud coupled with the two-hop overlay and one-hop caching strategy ensures that PresenceCloud 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 neighbors, 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 node in the mobile presence service. Moreover, the broadcasting message can be piggybacked in a buddy search message for saving the cost.

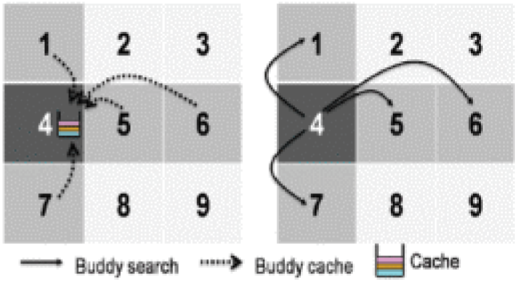

Fig3: An example of buddy list searching operations in PresenceCloud

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 $\boldsymbol{B} \neq nil$, the PS node q should hash each remaining identifier in $\boldsymbol{B}$ to obtain a grid ID, respectively.

6) Then the PS node q aggregates these $b(g)$ to become a new $B(j)$, for each $g \in Sj$. Here PS node j is the intersection node of $Sq \cap Sg$. And sends the new $\boldsymbol{B}(j)$ to PS node j.

VII. COST ANALYSIS

In this section, we provide 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 Q_{mesh}, 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 $Mcost = Q_{mesh} + R_{mesh}$, where R_{mesh} is the communication cost of replicating presence information to all PS nodes. Accordingly, we have $Mcost = O(n)$.

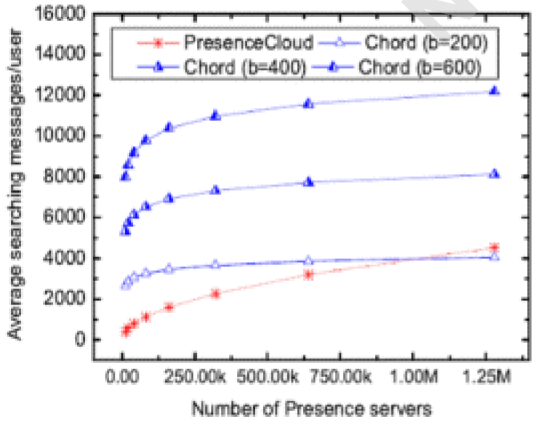

Fig4: Average searching messages vs. very large number of PS nodes

VIII. DISCUSSIONS

A number of issues require further consideration. Our current PresenceCloud does not address the communication security problem, and the presence server authentication problem, we discuss the possible solutions as follows. The distributed presence service may make the mobile presence service more prone to communication security problems, such as malicious user attacks and the user privacy. Several approaches are possible for addressing the communication security issues. For example, the Skype protocol offers private key mechanisms for end-to-end encryption. In PresenceCloud, the TCP connection between a presence server and users, or a presence server could be established over SSL to prohibit user impersonation and man-in-the-middle attacks. This end-to-end encryption approach is also used in XMPP/SIMPLE protocol.

The presence server authentication problem is another security problem in distributed presence services. In centralized presence architectures, it is no presence server authentication problem, since users only connect to an authenticated presence server. In PresenceCloud, however, requires a system that assumes no trust between presence servers, it means that a malicious presence server is possible in PresenceCloud. To address this authentication problem, a simple approach is to apply a centralized authentication server. Every presence server needs to register an authentication server; PresenceCloud could certificate the presence server every time when the presence server joins to PresenceCloud. An alternative solution is PGP web of trust model, which is a decentralized approach. In this model, a presence server wishing to join the system would create a certifying authority and ask any existing presence server to validate the new presence server's certificate. However, such a certificate is only valid to another presence server if the relying party recognizes the verifier as a trusted introducer in the system. These two mechanisms both can address the directory authentication problem principally.

IX. EXISTING SYSTEM

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 a 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.

Several IETF charters have addressed closely related topics and many RFC documents on instant

messaging and presence services have been published, e.g.,XMPP, SIMPLE. Jabber is a well-known deployment of instant messaging technologies based on distributed architectures.

It captures the distributed architecture of SMTP protocols. Since Jabber's architecture is distributed, the result is a flexible network of servers that can be scaled much higher than the monolithic, centralized presence services. 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, 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.

CONCLUSION

In this paper, we have presented PresenceCloud, a scalable server architecture that supports mobile presence services in large-scale social network services. We have shown that PresenceCloud achieves low search latency and enhances the performance of mobile presence services. In addition, we discussed the scalability problem in server architecture designs, and introduced the buddy-list search problem, which is a scalability problem in the distributed server architecture of mobile presence services. Through a simple mathematical model, we show that the total number of buddy search messages increases substantially with the user arrival rate and the number of presence servers. The results of simulations demonstrate that PresenceCloud achieves major performance gains in terms of the search cost and search satisfaction. Overall, PresenceCloud is shown to be a scalable mobile presence service in large-scale social network services.

REFERENCES

[1] Facebook, http://www.facebook.com.
[2] Twitter, http://twitter.com.
[3] Foursquare http://www.foursquare.com.
[4] Google latitude.
[5] Buddycloud, http://buddycloud.com.
[6] Mobile instant messaging,
a. http://en.wikipedia.org/wiki/Mobile instant
b. messaging.
[7] R. B. Jennings, E. M. Nahum, D. P. Olshefski,
a. D. Saha, Z.-Y. Shae, and C. Waters, "A study
b. of internet instant messaging and chat
c. protocols," IEEE Network, 2006.
[8] Gobalindex, http://www.skype.com/intl/en-
a. us/support/user-guides/ p2pexplained/.
[9] Z. Xiao, L. Guo, and J. Tracey, "Understanding
a. instant messaging traffic characteristics," Proc. of IEEE ICDCS, 2007.
[10] C. Chi, R. Hao, D. Wang, and Z.-Z. Cao, "Ims
a. presence server: Traffic analysis and
b. performance modelling," Proc. of IEEE ICNP,
c. 2008.
[11] Instant messaging and presence protocol workinggrouphttp://www.ietf.org/html. charters/impcharter.html.
[12] Extensible messaging and presence protocol ietfworkinggrouphttp://www.ietf.org/html. charters/xmpp-charter.html.
[13] P. Bellavista, A. Corradi, and L. Foschini, "Ims-based presence service with enhanced scalability and guaranteed qos for interdomain enterprise mobility," IEEE Wireless Communications, 2009.
[14] A. Houri, E. Aoki, S. Parameswar, T. Rang, , V. Singh, and H. Schulzrinne, "Presence interdomain scaling analysis for sip/simple," RFC Internet-Draft, 2009.
[15] M. Maekawa, "A pn algorithm for mutual exclusion in decentralized systems," ACM Transactions on Computer Systems, 1985.
[16] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (SHA1)," RFC 3174, 2001.
[17] M. Steiner, T. En-Najjary, and E. W. Biersack, "Long term study of peer behavior in the kad DHT," IEEE/ACM Trans. Netw., 2009.
[18] K. Singh and H. Schulzrinne, "Failover and load sharing in sip telephony," International Symposium on Performance Evaluation of Computer and Telecommunication Systems, July 2005.