ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

Presence Server For Mobile Ubiquity Services In Presence Cloud

D. Dhiravida Vasantha Nayaki[1], G. Mangayarkarasi[2]

M.E. CSE (Final year), K.Ramakrishnan College of Technology, Trichy, Tamil Nadu, India[1]

Assistant Professor, K.Ramakrishnan College of Technology, Trichy, Tamil Nadu, India[2]

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 into 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

Because of the ubiquity of the Internet, mobile devicesand cloud computing environments can provide presence-enabled applications, i.e., social network applications/services, worldwide. Face book, Twitter Foursquare Google Latitude, buddy cloud and Mobile Instant Messaging (MIM) are examples ofpresence-enabled applications that have grown rapidly inthe last decade. Socialnetwork services are changing theways in which participants engage with their friends on theInternet. They exploit the information about the status ofparticipants including their appearances and activities tointeract with their friends. Moreover, because of the wideavailability of mobile devices (e.g., Smartphones) thatutilize wirelessmobile network technologies, social networkservices enable participants to share live experiences instantly across great distances. For example, Facebookreceives more than 25

Billion shared items every month andTwitter receives more than55million tweets each day. Inthe future, mobile devices will become more powerful, sensing, and media capture devices. Hence, we believe it isinevitable that social network services will be the nextgeneration of mobile Internet applicationsA mobile presence service is an essential component ofsocial 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 mobileusers. The presence information includes details about amobileuser'slocation, availability, activity, device capability, and preferences. The service must also bind the user'sID to his/her current presence information, as well asretrieve and subscribe to changes in the presence informationof the user's friends. In social network services, eachmobile user has a friend list, typically called a buddy list, which contains the contact information of other users thathe/she wants to communicate with.

The mobile user's status is broadcast automatically to each person on thebuddy list whenever he/she transits from one status to theother. For example, when a mobile user logs into a socialnetwork application, such as an IM system, through his/hermobile device, the mobile presence service searches for and notifies everyone on the user's buddy list. To maximize amobile presence service's search speed and minimize thenotification time, most presence services use server cluster technology. First, we examine the server architectures of existingpresence services, and introduce the buddy-list search problem in distributed presence architectures in large-scale geographically data canter. The buddy-list search problem is a scalability problem that occurs when a distributed presenceservice is overloaded with buddy search messages. Then, we discuss the design of PresenceCloud, a scalableserver-to-server architecture that can be used as a building block for mobile presence services.

The rationale behind the design of Presence Cloud is to distribute the information of millions of users among thousands of presence server's on the Internet. To avoid single point of failure, no single presence server is supposed to maintain service-wide global information about all users. Presence Cloud organizes presence servers into a quorum-based server-to-server architecture to facilitate efficient buddy list searching. Italso leverages the server overlay and a directed buddysearch algorithm to achieve small constant search latency and employs an active caching strategy that substantiallyreduces the number of messages generated by each search for a list of buddies. We analyse the performance complexityof PresenceCloud and two other architectures, a Meshbasedscheme and a Distributed Hash Table (DHT)-basedscheme. Through simulations, we also compare the performanceof the three approaches in terms servers on the Internet.

The design of Presence Cloud, a scalable server-to-server architecture that can be used as a building block for mobile presence services. The rationale behind the design of Presence Cloud 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. Presence Cloud 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 an active caching strategy that substantially reduces the number of messages generated by each search for a list of buddies.

Analyze the performance complexity of Presence Cloud 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 Presence- Cloud achieves major performance gains in terms of reducing the number of messages without sacrificing search satisfaction. Thus, Presence Cloud can support a large-scale social network service distributed among thousands of servers on the internet.

Presence Cloud 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. Presence Cloud 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 contribution is that analyzes the scalability problems of distributed presence server architectures, and defines 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 Presence Cloud. 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.

II. DESIGN OF PRESENCECLOUD

The past few years has seen a veritable frenzy of researchactivity in Internet-scale object searching field, with many designed protocols and proposed algorithms. Most of theprevious algorithms are used to address the fixed objectsearching problem in distributed systems for differentintentions. However, people are nomadic, the mobilepresence information is more mutable and dynamic; anew design of mobile presence services is needed toaddress the buddy-list search problem, especially for the demand of mobile social network applications. Presence Cloud is used to construct and maintain distributed server architecture and can be used to efficientlyquery 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:

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

- Presence Cloud server overlay: It organizes presence servers based on the concept of grid quorum system. So, the server overlay of PresenceCloud has abalanced load property and a two-hop diameternode degrees, where n is the number ofpresence servers.

- One-hop caching strategy: It is used to reduce thenumber of transmitted messages and acceleratequery speed. All presence servers maintain cachesfor the buddies offered by their immediate neighbours.

- Directed buddy search**:** It is based on the directed searchstrategy. 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 ascalable server architecture for mobilepresence 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 toPresenceCloud via 3G or Wifi services. After the mobileuser joins and authenticates himself/herself to the mobilepresence service, the mobile user is determinately directedto one of Presence Servers in the Presence Cloud by using the Secure Hash Algorithm, such as SHA-1. The mobileuser opens a TCP connection to the Presence Server (PSnode) for control message transmission, particularly for thepresence information. After the control channel is established,the mobile user sends a request to the connected PSnode for his/her buddy list searching. Our PresenceCloudshall do an efficient searching operation and return thepresence information of the desired buddies to the mobileuser. Now, we discuss the three components of Presence-Cloud in detail below.

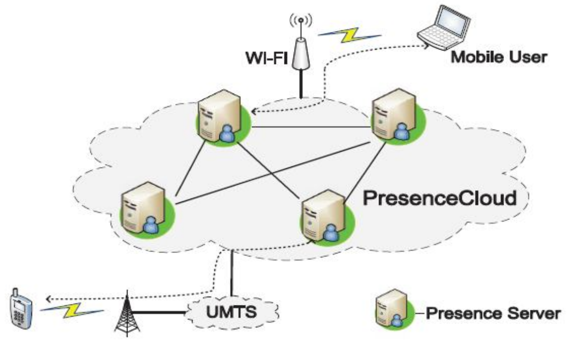

Fig 1. Architecture for presence cloud

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.

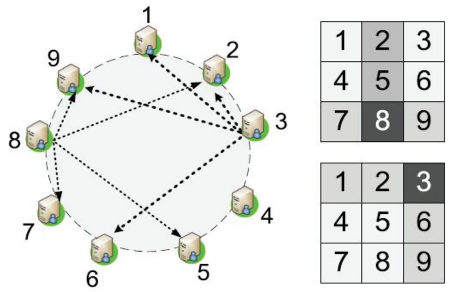

Fig 2. Presence cloud server overlay

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.

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.

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

3. When the PS node q receives a B, then retrieves eachbi from B and searches its user list and one-hopcache 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.

One-Hop Caching

To improve the efficiency of the search operation, PresenceCloud requires a caching strategy to replicate presenceinformation 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. 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 neighbouring PS nodes for getting updated quickly.

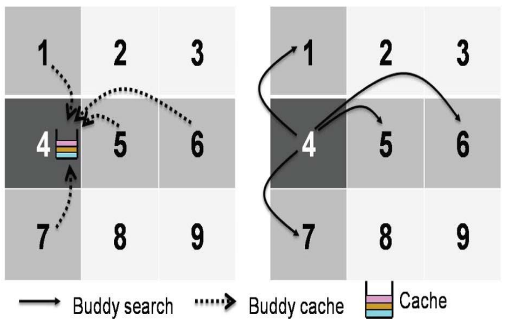

Fig 3. Directed buddy search

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, PS node 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 PS node 5. However, due to the one-hop caching strategy, PS node 6 has a buddy cache that contains these user lists of PS node {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 amobile 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. Consequently, this one-

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

hop caching strategy ensures that the user's presence information could remain mostly up-todate and consistent throughout the session time of the user.

III. PERFORMANCE EVALUATION AND COST ANALYSIS

- King-topology**:** This is a real Internet topologyfrom the King data set. The King data set delay matrix is derived from Internet measurements using techniques. It consists of 2,048 DNS servers. The latencies are measured as RTTs between the DNS servers.

- Brite-topology: This is an AS topology generated by the BRITE topology generator using the Waxman model where alpha and beta are set to 0.15 and 0.2, respectively. In addition, HS (size of one side of the plane) is set to 1,000 and LS (size of one side of a high-level square) is set to 100.

PERFORMANCE METRICS

Performance Metrics Within the context of the model, we measure the performance of server architectures using the following three metrics:

- Total searching messages: This represents the total number of messages transferred between the query initiator and the other PS nodes .Our experiments, since it is widely regarded to be critical in a mobile presence service.

- Average searching messages per-arrived user**:** The number of searching messages used per arrived user. Moreover, thismetric is independent of user arrival pattern.

- Average searching latency: This represents that average buddy searching time for a joining mobile user. This metric is acritical metric for measuring the search satisfaction of mobile presence service.

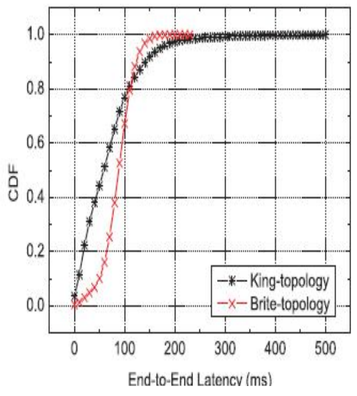

Fig 4. System performance

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

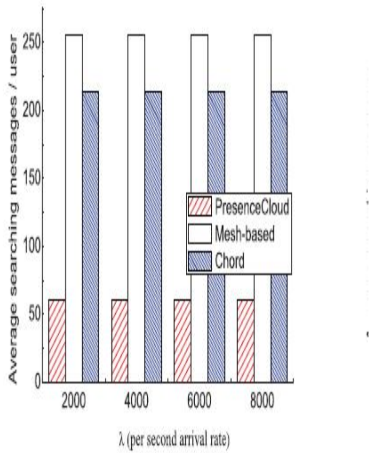

Fig 5. Performance of presence cloud

3.1 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 interserver 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 PS nodes, 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

ISSN(Online): 2320-9801
ISSN (Print): 2320-9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization) Vol.2, Special Issue 1, March 2014

Proceedings of International Conference On Global Innovations In Computing Technology (ICGICT'14)

Organized by

Department of CSE, JayShriram Group of Institutions, Tirupur, Tamilnadu, India on 6th & 7th March 2014

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 Chat Protocols," IEEE Network, vol. 20, no. 6, pp. 16-21, July/Aug. 2006.
[2] Z. Xiao, L. Guo, and J. Tracey, "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.
[5] Extensible Messaging and Presence Protocol IETF Working Group, http://www.ietf.org/html.charters/xmpp-charter.html,2012.
[6]Open Mobile Alliance, "OMA Instant Messaging and Presence Service," 2005.
[7]P. Saint-Andre, "Interdomain Presence Scaling Analysis for the Extensible Messaging and Presence Protocol (XMPP)," IETF Internet draft, 2008.
[8]X. Chen, S. Ren, H. Wang, and X. Zhang, "SCOPE: Scalable Consistency Maintenance in Structured P2P Systems," Proc. IEEE INFOCOM, 2005.
[9] Facebook, http://www.facebook.com, 2012.
[10]Twitter, http://twitter.com, 2012.
[11]Foursquare, http://www.foursquare.com, 2012.
[12]GoogleLatitude, http://www.google.com/intl/enus/latitude/intro.html, 2012.
[13]Buddy cloud, http://buddycloud.com, 2012.
[14] MobileInstantMessaging,http://en.wikipedia.org/wikI/Mobile_insnt messaging.
[15]Gobalindex,http://www.skype.com/intl/en- us/support/userguides/p2pexplained, 2012.