Presence Cloud for Mobile Presence Services in Social Network Application

Gonchi Shobha[#1], A.Anitha[*2]

[#1]*M.Tech Student, Department of Computer Science Engineering, Sri Mittapalli Institute of Technology for Women, Tummalapalem, JNTUK, AP, INDIA.*

[*2]*Asst.Professor, Department of Computer Science Engineering, Sri Mittapalli Institute of Technology for Women, Tummalapalem, JNTUK, AP, INDIA.*

[1]shobhagonchi@gmail.com

[2]anitha09@gmail.com

***Abstract*— Now a day's Social networking services on the Internet are growing and increasing number of people are using these new ways to communicate and share information. At the same time mobile phones are becoming more powerful and increasingly offer high speed Internet connectivity. Because of this people expect these social networking services to be available on their mobile device. The mobile presence service is 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 the availability 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 solve the problem, we propose efficient and scalable server architecture, called Presence Cloud, which enables mobile presence services to support large-scale social network applications. When a mobile user joins a network, Presence Cloud searches for the presence of his/her friends and notifies them of his/her arrival. Presence Cloud 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 Presence Cloud 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 Presence Cloud achieves performance gains in the search cost without compromising search satisfaction.**

***Keywords*— Presence Cloud, Social Network, GPS, Mobile presence.**

I. INTRODUCTION

Over the last few years mobile communication devices have become increasingly powerful and today many of them support applications being installed and executed on the device. Simultaneously the expansion of the third generation wide area cellular networks and other high speed wireless data technologies have made the Internet more accessible for mobile users, at prices suitable for surfing the Internet and with sufficiently low delay that even interactive packet based services are feasible. At the same time, the user experience of the web is expanding - facilitating increased collaboration and information sharing between users. Along with low cost high quality cameras, microphone, arrays etc. the web is supporting growing amounts of user generated content. This has lead to the evolution of online social networks1 and other means of fostering interaction. Over the last several years, the way that people use the web has changed, today four of the ten most visited Internet sites are social networking services [4]. The problems addressed in this master thesis concern how to simplify users' access to their social networks on the Internet when these users are using their mobile (phonebook equipped) device. The thesis also addresses how to match contacts from social networking services with contacts in the mobile device's phonebook. Additionally a prototype solution will demonstrate this concept [2]. This prototype will enhance the capabilities of a mobile phone by integrating existing social networks with the phonebook. This application is started from the phonebook so it gives the impression of being integrated with the phonebook, while at the same time provides fast access to contact information. Furthermore, the application provides a widget platform which enables users to create their own widgets, using web technologies that can access device functionality such as persistent storage and phonebook information. One of the initial design goals of the project was to try to avoid using a remotely located server and put all the logic of the application in the mobile device. An application that requires a fixed server might need potentially costly, updates of infrastructure if it rapidly becomes popular.

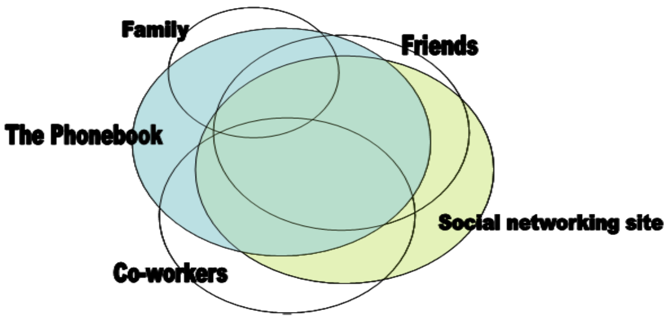


Fig. 1: Social network overlap

Social network graphs can be created using information found in different places, such as the contact lists from social networking sites, the phonebook from mobile devices, and blog rolls. By creating social network graphs it is possible to see how they overlap and use that information to synchronize the contacts on different social networks [1]. This information can then be used to enable communication with the people found in the different social networks. Fig.1 tries to illustrate how different social networks overlap. Furthermore it shows that different contact lists, such as a mobile device's phonebook or a contact list on a social networking site may contain contacts from several different social networks and therefore also overlaps.

II. THE PROBLEM STATEMENT

In this section, we describe the system model, and the buddy-list search problem. Formally, we assume the geographically distributed presence servers to form a server to server overlay network, G = (V; E), where V is the set of the Presence Server (PS) nodes, and E is a collection of ordered pairs of V. Each PS node ni$\in$ V represents a Presence Server and an element of E is a pair (ni; nj) $\in$ E with ni; nj$\in$ V. Because the pair is ordered, (nj ; ni) $\in$ E is not equivalent to (ni; nj) $\in$E. So, the edge (ni; nj) is called an outgoing edge of ni, and an incoming edge of nj . The server overlay enables its PS nodes to communicate with one another by forwarding messages through other PS nodes in the server overlay. Also, we denote a set of the mobile users in a presence service as U = {u1; : : : ; ui; : : : ; um}, where 1 $\le$ i $\le$ m and m is the number of mobile users. A mobile user ui connects with one PS node for search other user's presence information, and to notify the other mobile users of his/her arrival. Moreover, we define a buddy list as following.

Definition 1. Buddy list, Bi = {b1; b2; : : : ; bk} of user ui $\in$ U, is defined as a subset of U, where 0 < k $\le$|U|. Furthermore, B is a symmetric relation, i.e, ui $\in$Bj implies uj$\in$ Bi.For example, given a mobile user up is in the buddy list of a mobile user uq, and the mobile user uq also appear in the buddy list of the mobile user up. Note that to simplify the analysis of the Buddy-List Search Problem; we assume that buddy relation is a symmetric. However, in the design of Presence Cloud, the relation of buddies can be unilateral because the search operation of Presence Cloud can retrieve the presence of a mobile user by given the ID of the mobile user.

Problem Statement: Buddy-List Search Problem when a mobile user ui changes his/her presence status, the mobile presence service searches presence information of mobile users in buddy list Bi of ui and notifies each of them of the presence of ui and also notifies ui of these online buddies. The Buddy-List Search Problem is then defined as designing a server architecture of mobile presence service such that the costs of searching and notification in communication and storage are minimized.

Analysis of a Naive Architecture of Mobile Presence Service: In the following, we will give an analysis of the expected rate of messages generated to search for buddies of newly arrived user in a naive architecture of mobile presence services. We assume that each mobile user can join and leave the presence service arbitrarily, and each PS node only knows those mobile users directly attached to it. We also assume the probability for a mobile user to attach to a PS node to be uniform. Let's denote the average arriving rate of mobile users in a mobile presence service. In this paper, we focus on architecture design of mobile presence services and leave the problem of designing the capacity of presence servers as a separate research issue. Thus, we assume each PS node to have infinite service capacity [3]. Hence,$\mu=\lambda/n$ is the average rate of mobile users attaching to a PS node, where n is denoted the number of PS nodes in a mobile presence service. Let h denote the probability of having all users in the buddy list of ui to be attaching to the same PS node as ui. It is the probability of having no need to send search messages when ui attaches to a PS node. Thus,

$$h = \prod_{Bi} 1 \;[\![1/n]\!] \;=n^{\wedge}Bi$$

The expected number of search messages generated by this PS node per unit time is then $(n - 1) \times (1 - h) \times\mu$.

III. DESIGN OF PRESENCE CLOUD

Presence Cloud 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, we refine the ideas of P2P systems and present a particular design for mobile presence services. The three key components of Presence Cloud are summarized below:

•Presence Cloud server overlay organizes presence servers based on the concept of grid quorum system [29]. So, the server overlay of Presence Cloud 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 neighbours.

•Directed buddy search is based on the directed search strategy. Presence Cloud ensures a one-hop search; it yields small constant search latency on average.

Presence Cloud Overview: The primary abstraction exported by our Presence Cloud is used to construct 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. 2. In the mobile Internet, a mobile user can access the Internet and make a data connection to Presence Cloud via 3G or Wi fi services. After the mobile user joins and authenticates him/her 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 [5]. 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 Presence Cloud shall do an efficient searching operation and return the presence information of the desired buddies to the mobile user.

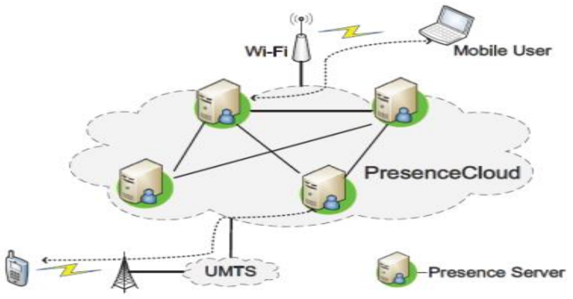


Fig.2. An overview of Presence Cloud

Presence Cloud Server Overlay: 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. The detailed description is as follows. Our Presence Cloud is based on the concept of grid quorum system [29], where a PS node only maintains a set of PS nodes of size O(√n), where n is the number of PS nodes in mobile presence services. In a Presence Cloud system, each PS node has a set of PS nodes, called PS list, that constructed by using a grid quorum system, shown in Fig. 3 for n=9. The size of a grid quorum is ⌈√n⌉×√n⌉. When a PS node joins the server overlay of Presence Cloud, 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⌈√n⌉ × ⌈√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 Presence Cloud server overlay.

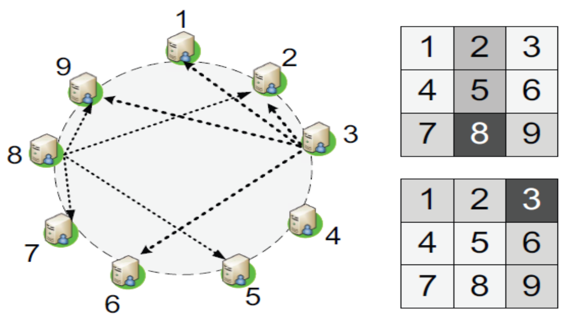


Fig.3. A perspective of Presence Cloud Server Overlay

Fig.3 illustrates an example of Presence Cloud, in which the grid quorum is set to ⌈√9⌉*⌈√9⌉. In the Fig. 3, the PS node8 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. We now show that each PS node in a Presence Cloud system only maintains the PS list of size O(√n), and the construction of Presence Cloud using the grid quorum results in each PS node can reach any PS node at most two hops.

One-hop Caching: 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 no demaintains 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 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-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. More specifically, it should be easy to see that, each PS node maintains roughly 2(⌈√n⌉−1)×u replicas of presence information, due to each PS node replicates its user list at most one hop away from itself. Here, u is denoted the average number of mobile users in a PS node.

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 [7]. Moreover, the broadcasting message can be piggybacked in a buddy search message for saving the cost.

IV. COST ANALYSIS

In this section, we provide a cost analysis of the communication cost of Presence Cloud 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 [6]. 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 – 1message 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 $2 \times (\lceil\sqrt{n}\rceil - 1)$ 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 straightforward to know that the communication cost of searching buddies and replicating presence information in Presence Cloud is Pcost = Qp = $2\times(\lceil\sqrt{n}\rceil-1)$. 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 node searched 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 $2\times(\lceil\sqrt{n}\rceil-1)$ messages (when $b >> 2(\lceil\sqrt{n}\rceil-1)$). When all mobile users are distributed equally among the PS nodes, which is considered to be the worst case, the Pcost = $4\times(\lceil\sqrt{n}\rceil-1)$.

TABLE I
Presence Architecture Comparison

	Mesh	PresenceCloud	DHT-based
Search	$O(n)$	$O(\sqrt{n})$	$O(b \times \log n)$
Replicas	$O(\lvert U \rvert)$	$O(\sqrt{n} \times u)$	$O(u)$
Latency	one hop	two hops	$\log n$ hops
Message Size	b	b/n	b/n
Message Reply Hops	$O(1)$	$O(1)$	$O(\log n)$
Maintenance Overhead	$O(n)$	$O(\sqrt{n})$	$O(\log n)$

V. SIMULATION RESULTS

We first evaluate and compare the three server architectures by considering the total buddy searching messages metric. We instantiated a server network of 256 PS nodes in our simulator, and ran a number of experiments to investigate the effect of scalability of PS nodes on involved searching messages. More precisely, we varied the user arrival rate from 100 per second to 8,000 per second to explore the relation between user arrival rate and the total searching messages. In this test, the number of buddies is set to 100.

Fig.4 depicts the total number of searching message transmissions during simulation time (1,800 seconds) under various rates of user arrival patterns (100 to 8,000 per second). We show that for a given number of PS nodes, the total number of searching messages is dominated by the user arrival rate (λ) significantly. In Fig.4, the total number of searching messages significantly increased as the user arrival rate increased. We could see that Presence Cloud outperforms all other designs. Mesh-base and Chord both require an enormous number of messages for searching buddy lists in higher user arrival rates. However, vast message transmissions may limit the scalability of the server architecture in mobile presence services.

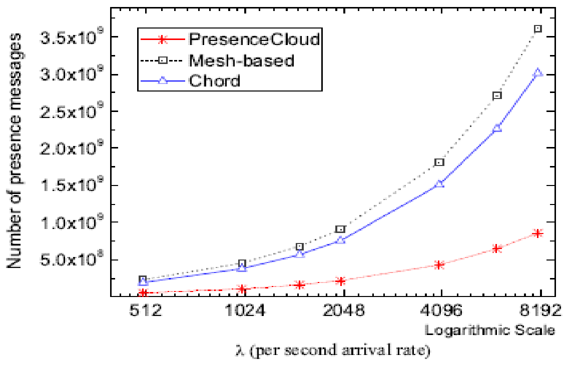


Fig. 4. The total message transmissions during simulation time (1,800s). (The x axis of this figure is in logarithmic scale)

Fig.5 shows the average number of searching message transmissions during simulation time (1,800 seconds) under various rates of user arrival patterns (100 to 8,000 per second). As shown in Fig. 8, the average number of searching message transmissions is independent of user arrival pattern. Increasing the rate of user arrival pattern does not increase the average searching message transmissions. Our Presence Cloud requires the least message transmissions. But mesh-based requires O(n) searching complexity (note that the number of PS node is set to 256), the experimental results fit our analysis in the Section 5. Chord-based design performs second highest message transmissions per searching operation. However, if the server architecture is not designed well, the scalability problem of servers may limit itself to scale more than thousands size, hence a poor server architecture may not support a very large number of servers.

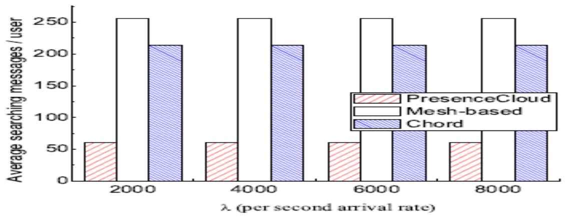
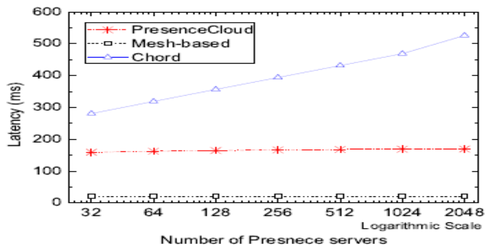


Fig. 5. The average message transmissions per searching operation

As shown as Fig.6, for Presence Cloud, the buddy searching latency grows gently with the number of PS nodes. However, the buddy searching latency of mesh-based design is significantly better than Presence Cloud. The reason is that, by using the mesh-based design, every PS node can retrieve all desired buddy information in its current replica and return the presence information of buddy to user in one hop RTT, and the one hop RTT is quite small in our assumption. Presence Cloud, on the other hand, needs to retrieve all available replicas from its neighbours, which affects the buddy search time. Although the mesh-based design achieves a faster buddy search time and a higher replica hit ratio than Presence Cloud, it sacrifices the scalability of the server architecture in mobile presence services. Under the Chord-based design, a search operation may need to visit a logarithmic number of PS nodes to find the buddies of users.



buddy until 240. As shown in Fig. 7, in all designs, the buddy searching latency is not impacted by the number of buddies. The search latency is dominated by the diameter of the overlay, thus the buddy searching latency does not grows with the number of buddies. Clearly, it is a trade-off, the

Fig.6. Average buddy searching latency vs. number of PS nodes (The x axis of this figure is in logarithmic scale)

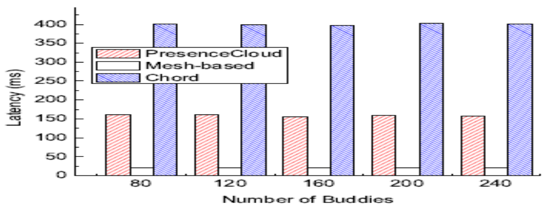


Fig. 7. Average buddy searching latency vs. number of PS nodes

We also studied the buddy searching latency of server architecture designs while varying the number of buddies. Fig. 7 depicts the buddy searching latency with the addition of

experiment results show that mesh-based design performs best search satisfaction, but suffers heavily communication cost.

However, our Presence Cloud reduces the significantly communication cost without sacrificing search satisfaction extremely.

VI. CONCLUSION

In this paper, we have presented Presence Cloud, a scalable server architecture that supports mobile presence services in large-scale social network services. We have shown that Presence Cloud 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 Presence Cloud achieves major performance gains in terms of the search cost and search satisfaction. Overall, Presence Cloud 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, http://www.google.com/intl/enus/latitude/intro.html.
[5] Buddy cloud, http://buddycloud.com.
[6] Mobile instant messaging, http://en.wikipedia.org/wiki/Mobile instant messaging
[7] 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, 2006.
[8]C.Chi, R.Hao and D.Wang,"Ims presence sarver".