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 4, September 2014

Mobile Presence Services with Scalability Using Concurrency in Programming

D Abdus Subhahan[1], P Lakshmi Samyuktha[2]

PG Student, SVEC, JNTUA University, Tirupati, India[1]

Assistant Professor, SVEC, JNTUA University, Tirupati, India[2]

ABSTRACT*:* 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. Web architectures are an important asset for various large-scale web applications, such as social networks or e-commerce sites. Being able to handle huge numbers of users concurrently is essential, thus scalability is one of the most important features of these architectures. Multicore processors, highly distributed backend architectures and new web technologies force us to reconsider approaches for concurrent programming in order to implement web applications and full scalability demands. While focusing on different stages of scalable web architectures, we provide a survey of competing concurrency approaches and point to their adequate usages.

KEYWORDS: Mobile presence services, Concurrency, PresenceCloud, distributed presence servers.

1. INTRODUCTION

Presence enabled applications can be provided by mobile devices and cloud computing environments, i.e., social network applications/ services, worldwide. From the last decade, presence enabled applications are growing rapidly. Some of the examples are Facebook [1], Twitter [2], Foursquare, Google Latitude, buddycloud [3] and Mobile Instant Messaging (MIM) [4].

An essential component of social network services is mobile presence service in cloud computing environments. It maintains an up-to-date list of presence information of all mobile users. The details of presence information includes about a mobile user's location, availability, activity, device capability, and preferences.

For example, when a user logs into a social network application, such as an Instant Messaging system, through his/her mobile device, the mobile presence service searches for and notifies everybody on the user's buddy list. Most presence services use server cluster technology [5] to maximize a mobile presence service's search speed and minimize the notification time.

Here we discuss about three contributions. First, PresenceCloud [13] is one of 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. Internet social network applications and

services can also utilize presence cloud that needs to replicate or search for mutable and dynamic data among distributed presence servers.
The second contribution is that we define a new problem called the buddy-list search problem by analyzing the scalability problems of distributed presence server architectures. The scalability problem in the distributed server architectures of mobile presence services is analyzed through our mathematical formulation.
Finally, we demonstrate the advantages of PresenceCloud by analyzing the performance complexity of PresenceCloud and different designs of distributed architectures, and evaluate them empirically. The primary abstraction exported by our PresenceCloud is used to construct scalable server architecture for mobile presence services, and can be used to efficiently search the desired buddy lists.

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 him/her 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 [6].

The Pretty Good Privacy Program (PGP) uses a "Web of Trust" model for establishing trust relationships between users. This could allow a user to indirectly and unknowingly trust others that the user does not know.

Then 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.

II. RELATED WORK

In this section, we describe previous researches on presence services, and survey the presence service of
Existing systems. Several IETF charters [7] have addressed closely related topics and many RFC documents on instant messaging and presence services (IMPS) have been published, e.g., XMPP [8], SIMPLE [9]. Jabber [10] 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 [11]. P2P-SIP [12] has been proposed to remove the centralized server, minimize maintenance costs, and keep from happening failures in server-based SIP deployment. To maintain presence information, P2PSIP clients are organized in a DHT system, rather than in a centralized server.

III. DESIGN OF PRESENCECLOUD

PresenceCloud is used to construct and maintain 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:

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 4, September 2014

3.1 PresenceCloud server overlay

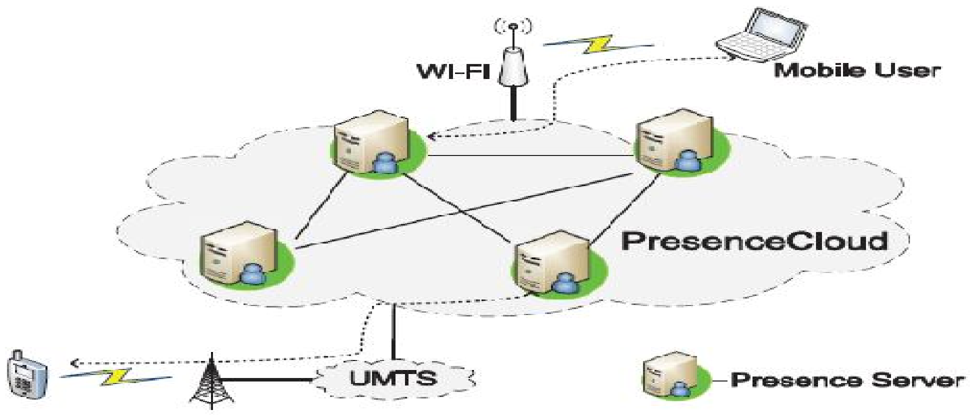

Fig. An overview of PresenceCloud

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

Our PresenceCloud is based on the concept of grid quorum system [14], 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. The size of a grid quorum is $\lceil \sqrt{n} \rceil \times \lceil \sqrt{n} \rceil$.

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 server.1 On the $\lceil \sqrt{n} \rceil \times \lceil \sqrt{n} \rceil$ 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.

3.2 One-hop caching strategy

PresenceCloud requires a caching strategy to make an exact copy of presence information of users to improve the efficiency of the search operation. 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.

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 $2(\lceil\sqrt{n}\rceil-1)\times u$ duplicates of presence information, due to each PS node duplicates its user list at most one hop away from itself. Here, u is denoted the average number of mobile users in a PS node.

3.3 Directed Buddy Search

We contend that reducing 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 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.

IV. CONCURRENCY

Concurrency is a property of a system representing the fact that multiple activities are executed at the same time. According to Van Roy, a program having "several independent activities,each of which executes at its own pace". In addition, the activities may perform some kind of interaction among them. The concurrent execution of activities can take place in different environments, such as single-core processors, multi-core processors, multi-processors or even on multiple machines as part of a distributed system. Yet, they all share the same underlying challenges: providing mechanisms to control the different flows of execution via coordination and synchronization, while ensuring consistency.

Apart from recent hardware trends towards multi-core and multiprocessor systems, the use of concurrency in applications is generally motivated by performance gains. Cantrill et al. describe three fundamental ways how the concurrent execution of an application can improve its performance:
Reduce latency A unit of work is executed in shorter time by subdivision into parts that can be executed concurrently.

Hide latency Multiple long-running tasks are executed together by the underlying system.this is particularly e.ective when the tasks are blocked because of external resources they must wait upon, such as disk or network I/O operations.

Increase throughput By executing multiple tasks concurrently, the general system throughput can be increased. It is important to notice that this also speeds up independent sequential tasks that have not been specifically designed for concurrency yet.

The presence of concurrency is an intrinsic property for any kind of distributed system. Processes running on di.erent machines form a common system that executes code on multiple machines at the same time.Conceptually, all web applications can be used by various users at the same time._us, a web application is also inherently concurrent._is is not limited to the web server that must handle multiple client connections in parallel. Also the application that executes the associated business logic of a request and the backend data storage are confronted with concurrency.

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 4, September 2014

4.1 Concurrency and Parallelism

In the literature, there are varying definitions of the terms concurrency and parallelism, and their relation. Sometimes both terms are even used synonymously. We will now introduce a distinction of both terms and of their implications on programming.

Concurrency vs. Parallelism

Concurrency is a conceptual property of a program, while parallelism is a runtime state. Concurrency of a program depends on the programming language and the way it is coded, while parallelism depends on the actual runtime environment. Given two tasks to be executed concurrently,there are several possible execution orders._ey may be performed sequentially (in any order), alternately, or even simultaneously. Only executing two different tasks simultaneously
yields true parallelism. In terms of scheduling, parallelism can only be achieved if the hardware architecture supports parallel execution, like multi-core or multi-processor systems do. A singlecore
machine will also be able to execute multiple threads concurrently, however it can never provide true parallelism.

Concurrent Programming vs Parallel Programming

Differentiating concurrent and parallel programming is more tedious, as both are targeting different goals on different conceptual levels. Concurrent programming tackles concurrent and interleaving tasks and the resulting complexity due to a nondeterministic control flow.Reducing and hiding latency is equally important to improving throughput. Instead, parallel programming favors a deterministic controlflow and mainly reaches for optimized throughput.the internals of a web server are the typical outcome of concurrent programming, while the parallel abstractions such as Google's MapReduce [Deaþ.] or Java's fork/join [Leaþþ] provide a good example of what parallel programming is about.

Parallel programming is also essential for several specialized tasks. For instance, a graphics processing unit is designed for massive floating-point computational power and usually runs a certain numerical computation in parallel on all of its units at the same time. High-performance computing is another important area of parallel programming. It takes advantage of computer clusters and distributes sub-tasks to cluster nodes, thus speeding up complex computations.

4.2 Models for Programming Concurrency

Van Roy introduces four main approaches for programming concurrency that we will examine briefly.The more important models will be studied later as potential solutions for programming concurrency inside web architectures.

Sequential Programming

In this deterministic programming model, no concurrency is used at all. In its strongest form,there is a total order of all operations of the program. Weaker forms still keep the deterministic behaviour. However, they either make no guarantees on the exact execution order to the programmer a priori. Or they provide mechanisms for explicit preemption of the task currently active, as co-routines do, for instance.

Declarative Concurrency

Declarative programming is a programming model that favors implicit control flow of computations. Controlflow is not described directly, it is rather a result of computational logic of the program.The declarative concurrency model extends the declarative programming model by allowing multiple .ows of executions. It adds implicit concurrency that is based on a data-driven or a demand-driven approach. While this introduces some form of nondeterminism at runtime,the nondeterminism is generally not observable from the outside.

Message-passing Concurrency

This model is a programming style that allows concurrent activities to communicate via messages. Generally, this is the only allowed form of interaction between activities which are otherwise completely isolated. Message passing can be either synchronous or asynchronous resulting in different mechanisms and patterns for synchronization and coordination.

Shared-state Concurrency

Shared-state concurrency is an extended programming model where multiple activities are allowed to access contended resources and states. Sharing the exact same resources and states among different activities requires dedicated mechanisms for synchronization of access and coordination between activities.The general nondeterminism and missing invariants of this model would otherwise directly cause problems regarding consistency and state validity.

V. CONCLUSIONS

In this paper we have presented a scalable server architecture called PresenceCloud that supports mobile presence services in large scale social network services.Although focusing on different stages of scalable web architectures, we provide a survey of competing concurrency approaches and point to their adequate usages**.**

REFERENCES

[1] Facebook, http://www.facebook.com, 2012.
[2] Twitter, http://twitter.com, 2012
[3] Buddycloud, http://buddycloud.com, 2012.
[4] Mobile Instant Messaging, http://en.wikipedia.org/wiki/ Mobile_instant_messaging, 2012.
[5] 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.
[6] D. Eastlake and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," IETF RFC 3174, 2001.
[7] Extensible Messaging and Presence Protocol IETF Working Group, http://www.ietf.org/html.charters/xmpp-charter.html, 2012.
[8] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence Describes Instant Messaging (IM), the Most Common Application of XMPP," IETF RFC 3921, 2004.
[9] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle, Session Initiation Protocol (SIP) Extension for Instant Messaging, IETF RFC 3428, 2002.
[10] http://www.jabber.org, 2012.
[11] Peer-to-Peer Session Initiation Protocol IETF Working Group, http://www.ietf.org/html.charters/p2psip-charter.html, 2012.
[12] K. Singh and H. Schulzrinne, "Peer-to-Peer Internet Telephony Using SIP," Proc. ACM Int'l Workshop Network and Operating Systems Support for Digital Audio and Video (NOSSDVA), 2005.
[13] Chi-Jen Wu . Jan-Ming Ho , Ming-Syan Chen , "A Scalable Server Architecture for Mobile Presence Services in Social Network Applications ," IEEE Trans. Mobile Computing, vol. 12, no. 2, pp. 386-398, Feb. 2013, doi: 10.1109/TMC.2011.263
[14] M. Maekawa, "A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems," ACM Trans. Computer Systems, vol. 3, pp. 145-159, 1985.