Vol.II/Issue III March-2013 ISSN:2278-4187

Anusandhan

Science Technology & Management Journal of AISECT University

AISECT UNIVERSITY

Where aspirations become achievements.

AISECT University : Village - Mendua, Post - Bhojpur, District - Raisen, (M.P.) India Pin - 464993
Ph.-0755-6766100, Ph.-07480-295707,
E-Mail - aisectjournal@rediffmail.com, info@aisectuniversity.ac.in
Web site : www.aisectuniversity.ac.in

RABINDRANATH TAGORE UNIVERSITY

UGC Approved Journals

Anusandhan (AUJ-AN)

- Technology & Management

Indexing and Impact Factor :

INDEX COPERNICUS : 48609 (2018)

Read / Download More Articles

DESIGNING A CHAT ROOM APPLICATION: USING PEER-TO-PEER AND CLIENT-SERVER APPROACH OF DISTRIBUTED SYSTEMS

Dr. S.K Gandhi[1], Pawan Kumar Thakur[2]
Department of Computer Science & Engineering[1, 2]
AISECT University, Bhopal (M.P)

ABSTRACT

A distributed system is an application that executes a collection of protocols to coordinate the actions of multiple processes on a network, such that all components cooperate together to perform a set of related tasks. Early distributed systems emerged in the late 1970s and early 1980s because of the usage of local area networking technologies. Internet-scale distributed systems emerged in the 1990s because of the growth of the Internet. A distributed system can be physically instructed by two ways: First, fully connected network peer- to –peer approach in which each of the nodes is connected to each other. Second, partially connected network in which a direct links exist between some but not all pairs of computers. A few of the partially connected network models are star (client server) structured networks, multi-access bus networks ring structured networks, and tree-structured networks. The purpose of this paper is to design a chat room application which is based on the architecture of distributed system. This paper used a peer-to-peer approach and a client-server one to design the chat room application.

KEYWORDS: *Distributed system, Architecture, Peer-to-peer, Client-server, Manager, Login, Online Chat, Port*

I. INTRODUCTION

A chat application can be design using different approaches of distributed system architecture. We will explain in more details two of them: a peer-to-peer approach and a client-server one. Even though different models are possible there remains some functionality in common as shown in Fig.1.1.

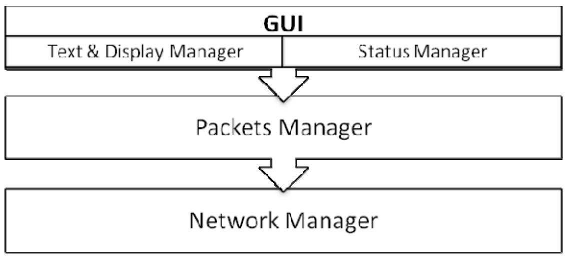

Fig.1. Architectural styles

In the above Fig. 1.1, Part A remains the same in all architectural styles which consist of GUI, text and display manager and status manager. Part B must be different in case of a peer-to-peer or client-server architecture [1]. In this paper section 3 represents the design scheme of architecture, section 4 develop a chat application use of a peer-to-peer architecture and how it work, section 5 develop a chat application use of a client-server architecture, section 6 compare the scalability and performance of both the architecture.

II. OBJECTIVES

This paper designs a chat room by using the architecture of distributed system which fulfills the following objectives:

(a) Design a Networks scheme for peer-to-peer and client server architecture.

(b) Use the peer-to- peer architecture of distributed system to design a chat room with following concepts:

- How does it work?
- Login and who is on-line?
- Ask for friend relation
- Chatting

(c) Use the client server architecture of distributed system to design a chat room with following concepts: How does it work?

- Login and who is on-line?
- Ask for friend relation
- Chatting

(d) Compare the both the architecture and find out which one architecture is best.

III. DESIGN SCHEME

To design a chat room based on the peer- to–peer and client server architecture of distributed system we will use the following managers as shown in Fig.2[1]:

(a) Network manager. A network manager is responsible for listening to the network.

(b) Packets manager. The packet manager is responsible for classifying packets between system and normal chat messages.

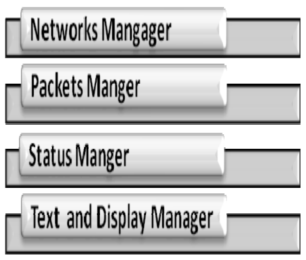

Fig. 2 Networks Managers

(c). Text and display manager. The text and display manager is responsible for normal chat messages received from the packet manager.

(d). Status manager. This is responsible for status, login and friend messages.

The most important thing is to run all these modules in separate threads because it is really important to deal with and display messages when the network manager is sending another one, etc. Fig. 3(a,b) showed the global shape of the two architectures: client server architecture and peer-to-peer architecture[2].

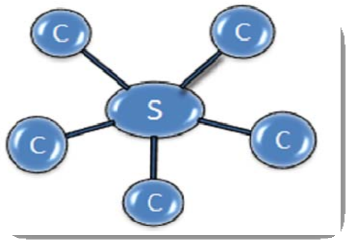

Fig.3(a) Client Server Architecture

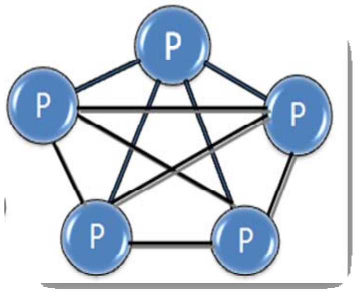

Fig. 3.(b) Peer-to-Peer Architecture

We would try to explain more how these two architectures can be developed and how we can deal with some aspects of the process such as login, discovery, friend relationships etc in the next section of this paper.

IV. FIRST APPROACH: PEER-TO-PEER ARCHITECTURE

The first possibility to develop a chat application is the use of a peer-to-peer architecture of distributed system[3]. A fully connected network peer to peer architecture is a network in which each of the nodes is connected to each other as shown in Fig. 1.3.

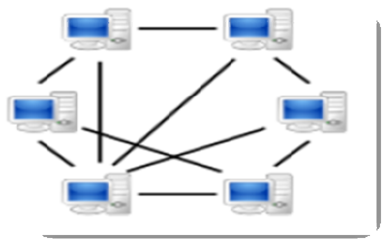

Fig. 4 Peer-to-peer architecture of distributed system (Wikipedia.org)

(a) How does it work?

We are working with a pure peer-to-peer network which is composed by peers and links between them only. We do not want to use the functionalities of super-peers to keep this system as basic and simple as possible. Every node must maintain two separated lists[4].

(i). One list containing names of friends (List<Friend1, Friend2, Friend3, ...,>).

(ii). Another list containing all online peers that we know (List<[User1, IP, Port], [User2, IP, Port], [User3, IP, Port], ...>).

The friends-list is modified when we add or remove a friend. This process will be explained in more details in our next section. The online-list is managed by the status manager[3]. In a first time, we add peers into this list during the login phase. But, after that, we can use an eventually perfect failure detector (EPFD) to remove peers when they leave or they crash[5].

(b) Login and who is on-line?

We take an example with five peers (nodes) already connected on the network we are the sixth one arriving on the network as shown in Fig.1.5 and let's make some assumptions:

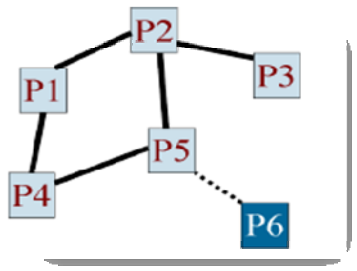

Fig. 5 Example sixth one (node) arriving on the network

We assume that the discovery process is made easier thanks to a public place where connected peers can publish their point-of-entrance in the network. Whatever happens we need to know at least one entry point.We assume that there exists a (well managed) DHT (Distributed Hash Table) containing all credentials of registered people of this chat service.

Peer 6 chooses Peer 5 as entry point. Peer 6 will send a join message to Peer 5 : **JOIN(Peer6, Password, IP, Port)** Figure 6 shows the behaviour of every node when a JOIN message arrives[6].

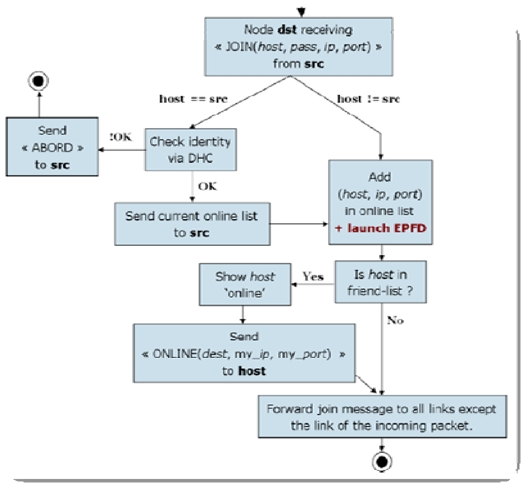

Fig6 Behavior model

In addition to that, we can imagine a way to avoid Join messages to loop forever in the network with a simple 'id' feature or something. Thanks to this login process, we can retrieve the list of already connected people and launch our EPFD on every host to detect when they leave or they crash. We are now ready to begin chat sessions with connected friends.

(c) Ask for friend relation

When the application that is running and the user fully logged-in, we can simply search in the online list to friend new friends. Then, we can directly send an Ask-Friend message to the IP address and the port of the new friend[5]. This kind of message is treated by the packets manager. If the new friend agrees with this new relationship, he replies with a Friend- Ok message to the original peer. If the new friend not wants to begin a relationship with the peer, he simply never replies to the message. This process is very simple thanks to the fact that we are in a peer-to-peer network, where every node can act as a server, client or both[7]. The communication between peers can be done directly thanks to the information contained in the online-list.

(d) Chatting

In the same way than the friend messages, peers can simply exchange normal chat messages with each other thanks to the fact that they have all the information to join other peers. We can easily understand here the importance of the packets manager. Actually, when a packet arrives in a node, the packet manager can simply read the header of this packet to know if it is a system notification (friends, status, etc.) or a real chat message (containing text to display).

V. SECOND APPROACH: CLIENT-SERVER ARCHITECTURE

The second possibility to develop a chat application is the use of client-server architecture of distributed system as shown in Fig. 1.7.

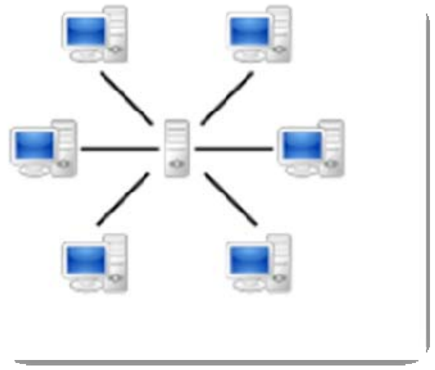

Fig. 7. A centralized server-based system of nodes (Wikipedia.org).1.5.1 How does it work?

We think that second architecture is simpler to achieve because this solution is more centralized and then, easier to understand. Fig1.8 shows the main modules of a client and the server. We assume that the server is unique on the network.

An important thing to notice is that the server might use a reliable multicast to contact every node connected to him. This kind of behaviour will happen every time a (new) node is connecting or disconnecting[8].

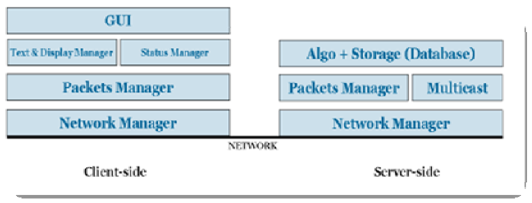

Fig. 8 Modules of a client and the server
1.5.2 Login and Who is on-line

The login phase is simple it consists in simple messages send to the (only one) server on the network. Everything is centralized so the server has just to check in a local database if the permissions of the user are correct[4]. If they are:

Firstly, the server replies with a Join Ok message to the client. The message contains the list of connected friends.

Secondly, the server sends a notification to every node which has the new node in its friend list. Here, we can understand the usage of the multicast module. A global scenario:

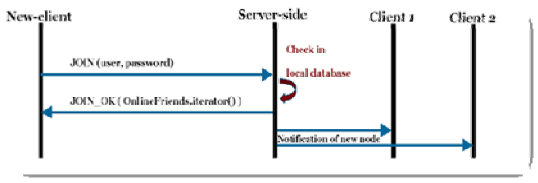

Fig. 9 Login Phase 1.5.3 Ask for friend relation

The process of ask for a friend relation is simpler than in the peer-to-peer architecture as shown in Fig.10.

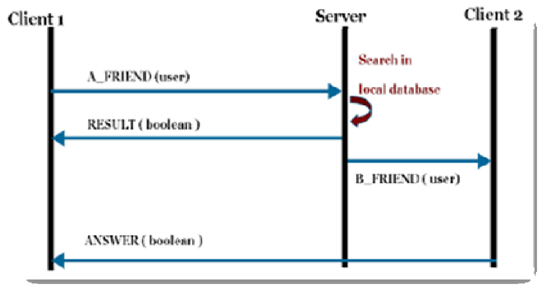

Fig.10 Ask for a friend relationship

In above Fig.1.10 note that the server replies with a Result message to indicate to the client if the searched user exists or not. The reply to the invitation of relationship is an Answer message.

Chatting The process is really simple again. When there is a friend who is online, you can begin a chat session with him. You always send your message to the server with its destination and the content[9]. Then, the server just forwards the content to the destination. As in the peer-to-peer architecture, the packets manager has a very important role to classify system notifications and real chat messages.

VI. COMPARISON AND DISCUSSION

Our first peer-to-peer approach is really good for scalability because at no point of the architecture, the number of node is important. We do not have to change anything if we use this chat application with 5 people or 50 people. The only thing in relation with the number of node is the list of online people[10]. We can imagine a better data structure to store this information (local hash-map, etc.) But with the high volume of space we have in nowadays computers. Even if a list of 1000000-entries is not a problem to store. Once again, if we need to search in this list very often, we can imagine a better data structure than a list.

Our second client-server approach an application like that is not really effective. The major problem is the fact that everything is centralized and we cannot imagine a number of nodes growing up without a failure of the central server. In this case, the entire chat application is down. Also, we should add other servers to be able to reply to all the queries in a respectable delay.

VII. CONCLUSION

The peer-to-peer is not the best one for the login performance. Actually, we have to wait until our Join message reaches the farthest node to be fully connected to the network. If we have a global network with a high latency and a shortest path from us to the farthest node of about 50 hops, we must wait 50 times the mean latency of the network to be sure that everybody on the network is aware of our presence. The client-server architecture provide much better performance for the login phase thanks to the small number of hops needed to join all peers. This is the advantage of a centralized server. But to conclude we think that the drawbacks of client server are bigger than the advantages.

REFERENCES

[1] Kundan Singh and Henning Schulzrinne, *"Peer-to-peer internet telephony using sip"*, Department of Computer Science, Columbia University.

[2]. Yang, B. H. & Garcia-Moline, *"Designing a Super-Peer Network"*, Standford University, February 2002.

[3] Sinha P.K, *"Distributed Operating Systems Concepts and Design"*, Prentice-Hall of India private Limited, 2008.

[4]. W. Siong Ng, B. Chini, K. Lee Tan. *"Best Peer: A Self-Configurable Peer-to-Peer System."* In Proc. of the 18th Int. Conf. on Data Engineering, California, 2002.

[5]. Kwok, S., Lui, S., A License Management Model for Peer-to-Peer Music Sharing. International Journal of Information Technology & Decision Making, September, vol. 1, #3, pp. 541-558,2002.

[6]. Lang, K. and Vragov, R. "*Using Experimental Methods to Evaluate the Effectiveness of Different Pricing Mechanisms for Content Distribution Over Peer-to-Peer Networks.*" Americas Conference on Information Systems. 2005.

[7]. Lechner, U. and Hummel, J. "*Business Models and System Architectures of Virtual Communities: From a Sociological Phenomenon to Peer-to-Peer Architectures"*, International Journal of Electronic Commerce, 6, 3,2002. pp.41-53.

[8] E. Bonsma, C. Hoile. *"A distributed implementation of the SWAN peer-to-peer look-up system using mobile agents"*. In Proc. of the AAMAS'02 Workshop on Agents and Peer-to-Peer Computing, Italy, 2002.

[9] Bibliography containing references on *Distributed Computing* can be found at: ftp:ftp.cs. umanitoba.ca/pub/bibliographies/Distributed/Osser.html

[10]. Bibliographies containing references on *Distributed Systems* can be found at: www.wikipeida.org.