Design and Analysis of a Client–Server Assignment Problem in Local Area Network

Chodipalli Sudhakar, Medara Rambabu, Kuna Venkata Kiran

M.Tech Final Year Student, Department of CSE, Kaushik College of Engineering, Visakhapatnam
Associate Professor HOD, Department of CSE, Kaushik College of Engineering,

Visakhapatnam
Assistant Professor, Department of CSE, Kaushik College of Engineering, Visakhapatnam

ABSTRACT:

Network is an interconnection of various systems in order to transfer the data from one system to other. Generally networks are classified into various types based on the usage and configured. Client-Server network is one among the various types of networks, where a client will always sends a request and server will always generates a response. Interactivity is the major performance factor for any type of network including client-server architecture. Here the client will have a facility to connect to the server from any location to interact with each other for data transfer. In a large scale networks the interactive performance not only depends only on client-to-server network latency but also vice versa. Generally when a client tries to connect the server for the interactivity, sometimes client may connect directly if server is not busy with any other clients but maximum there will be failures for clients sue to server busyness. So in this paper we investigate the problem of efficiently assigning the clients to server for utilizing the maximum interactivity between each other. In this paper as an extension we have also implemented the server status updation like idle, normal, busy and overloaded.

As the server is started initially it will be in idle state as there is no clients assigned till that state, once if any client is assigned with that server then the state will change to normal where the server can have the ability to access more clients also till limit less than 3. Once if the limit have exceeded by maximum clients then the state becomes overloaded. At this particular state the client can’t able to accept any more new requests by other clients and it will display the status as busy.

In this paper we have also implemented the automatic re-directing of client requests to other server when it is in busy state as an additional extension. As a proof of concept we have implemented our proposed assignment problem on a set of nodes which are connected through LAN and our experimental results clearly tells that this is the approach which will utilize maximum network resources in order to improve the performance of server.

Keywords: Distributed interactive application, Client assignment, Interactivity, Consistency, Fairness, Status

1. INTRODUCTION

Distributive interactive application provides shared sense of space, consistency, presence to its users. Distributive interactive simulations allow participants at different locations can interact through network. Thus, the interactivity of DIAs is vital for participants to have interactive experience. Interactivity is time taken to see effect of operation issued by one client to other or same client. And time taken is called as interaction time. Latency is also called as round trip time. It is the time taken by the packet to go some destination and route back to the source. Actually we cannot eliminate this from any network but we can reduce to some extent. In the real time, servers are deployed in the different location in order to reduce the network latency. So we need latency driven distribution of servers even though there is no limitation on it resources. In distributed server architecture, the state of a DIA is to be imitating across distributed servers. It is shown in figure. 1, each clients need to be assigned to the same server in order for interaction between/among them. Here client first sends operation to the assigned server. Then server will forward the operation to the other servers. The interaction time between any pair of clients must include the network latencies between the clients and their assigned servers and the network latencies between their assigned servers. These network latencies change by how clients are assigned to the servers.

With this, the interaction time is also affect by the consistency and fairness requirements of DIAs. Consistency means the same view must be created among all clients at same time to support good interaction between the clients. On other hand, fairness creates same chance of participation without considering their network condition.

Adding consistency and fairness to our DIA creates synchronization delay due to different network latencies. These synchronization delays are dependent on the assignment of clients to servers. Therefore, how to assign the clients to the servers in DIAs is vital to their interactivity performance.

In this paper, we found the problem of effectively assigning clients to servers for maximizing the interactivity of DIAs. We concentrate on continuous
DIAs that change their states on passing time. Examples of continuous DIAs include distributed virtual environments, distributed interactive simulations, and multiplayer online games.

We commence by mathematically modelling the interactivity performance of continuous DIAs under the consistency and fairness requirements.

![Fig 1. Distributed server architecture](image)

2 PROBLEM FORMULATIONS:

2.1 System Model:

We model the underlying network supporting the DIA by a graph consisting of a set of nodes \( V \). A distance \( d(u,v) > 0 \) is associated with each pair of nodes \((u,v) \in V \times V\), representing the network latency of the routing path between nodes \( u \) and \( v \). Let \( S \subseteq V \) be the set of servers in a network and \( C \subseteq V \) be the set of clients. Each client needs to be assigned to a server for sending user operations and receiving state updates. For each client \( c \in C \), we denote by \( s_d(c) \in S \) as the server that client \( c \) is assigned to.

![Fig 2. Interaction paths in a network](image)

The clients interact with one another through their assigned servers. Specifically, when a client \( c_i \) issues an operation, the effect of the operation is presented to another client \( c_j \) through the following process. First, \( c_i \) sends the operation to its assigned server \( s_d(c_i) \). Then, \( s_d(c_i) \) forwards the operation to \( c_j \) assigned server \( s_d(c_j) \) if they are different. Finally, \( s_d(c_j) \) executes the operation, possibly after some artificial synchronization delay, and then delivers the resultant state update to \( c_j \). In the above interaction process, the paths from \( c_i \) to \( s_d(c_i) \), from \( s_d(c_i) \) to \( s_d(c_j) \) and from \( s_d(c_i) \) to \( c_j \) are involved. Similarly, if \( c_j \) issues an operation, the same three paths in the network are involved in the interaction process for \( c_i \) to see the effect of the operation. Therefore, we refer to the concatenation of these three paths as the interaction path between \( c_i \) and \( c_j \).

For example, in the network of Fig. 2, suppose clients \( c_1 \) and \( c_2 \) are assigned to server \( s_1 \), and client \( c_3 \) is assigned to server \( s_2 \). Then, the interaction path between \( c_1 \) and \( c_2 \) is indicated by the dotted line, and the interaction path between \( c_1 \) and \( c_3 \) is indicated by the dashed line.

The length of the interaction path between two clients \( c_i \) and \( c_j \) represents the network latency involved in their interaction, which is given by \( d(c_i, s_d(c_i)) + d(s_d(c_i), s_d(c_j)) + d(s_d(c_j), c_j) \). Note that the length of the interaction path from a client \( c_i \) to itself is \( 2d(c_i, s_d(c_i)) \), i.e. the round-trip time between \( c_i \) and its assigned server \( s_d(c_i) \), which represents the network latency involved for \( c_i \) to see the effect of its own operation.

2.2 Consistency and Fairness Models:

In continuous DIAs, the progress of the application state is typically measured by the time elapsed since the initial state of the application. We shall call it the simulation time. For instance, the simulation time of multiplayer online game records the time elapsed in its virtual world. In the distributed server architecture, each server and client has a copy of the application state and its associated simulation time. The simulation times of all servers and clients should advance at the same rate. However, they do not have to be synchronized, i.e. their readings do not have to be the same at the same wall-clock time. Normally, the simulation time of a client lags behind the simulation time of its assigned server due to the network latency of delivering state updates from the server to the client.

The consistency requirement for continuous DIAs is to ensure that all clients share the same view of the application state when their respective simulation times reach the same value. This is automatically guaranteed among the clients assigned to the same server because they all inherit the application state from their assigned server through state updates. Nevertheless, the application states seen by the clients assigned to different servers may not be identical at the same simulation time if the application states maintained by their assigned servers are not consistent. Since the state of a
continuous DIA changes due to both user operations and time passing, to ensure consistency among the application states at the servers, each user operation must be executed by all servers at the same simulation time.

The fairness requirement is to ensure that all clients have the same chance of participation regardless of their network conditions. This is particularly important for applications where users compete with each other. In essence, fairness is concerned with the order of executing user operations. For example, a participant would gain an unfair advantage in an air combat game if an action performed by him takes effect before another action performed earlier by a different participant. To guarantee fairness in continuous DIAs, the order of operation execution must be the same as the operation issuance order at the clients based on the simulation time. In addition, the time interval between the issuances of two operations in terms of simulation time must also be preserved between the executions of these operations. This entails executing each operation at the server at a simulation time that is of a constant lag behind the simulation time of the operation issuance. Integrating both consistency and fairness requirements, we get the following criterion: all operations must be executed by all servers at simulation times that are of a constant lag behind the operation issuances.

2.3 Minimum Achievable Interaction Time:

Given a client assignment, the minimum achievable interaction time meeting the above criterion of consistency and fairness is the maximum length of interaction paths between all client pairs, i.e.

$$D = \max_{c_i, c_j} \{d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + (s_A(c_j), c_j)\},$$

(1)

This is achievable by synchronizing the simulation times at all clients and setting the offset of each server s’s simulation time relative to all clients’ simulation time at

$$D - \max_{c'} \{d(c', s_A(c')) + d(s_A(c'), s)\},$$

(2)

We present an example to illustrate the minimum achievable interaction time. In the network shown in Fig. 3a, three clients $c_1, c_2, c_3$ are assigned to servers $s_1, s_2, s_3$, respectively. A straightforward strategy of time setting is to make the simulation times at all clients synchronized, and also make the simulation times at all servers synchronized (Fig. 3b). The simulation time of clients must lag behind the simulation time of servers due to the network latency of delivering state updates. Suppose that at simulation time 0, one of the three clients issues an operation. If $c_i$ issues the operation, its operation first reaches server $s_1$ at simulation time six, as shown in Fig. 3b. Then, the operation is forwarded to the other two servers, which receive it at simulation time seven. As can be seen, the latest possible simulation time for a server to receive the operation is 11 (i.e. the time for server $s_2$ to receive the operation if it is issued by $c_3$). Therefore, the constant lag for operation execution must be at least 11. If the operation is executed at simulation time 11, all the clients receive and display the resultant state update at simulation time 11, and consequently, the interaction time is 11. On the other hand, if the offset between each server and all clients is set according to (1), the delivery of operation and state update is shown in Fig. 3c. It can be seen that regardless of its origin, the operation can be received and executed by all servers by simulation time nine. Thus, the resultant
interaction time is nine, which equals the maximum interaction path length.

2.4 Problem Statement:

The client assignment problem for maximizing the interactivity of continuous DIAs is formulated as follows:

Given a set of servers $S$ and a set of clients $C$ in a network, find a client assignment that minimizes the maximum length of interaction paths between all client pairs, i.e., to minimize

$$D = \max_{c_i, c_j \in C} \{d(c_i, s_A(c_i)) + d(s_A(c_i), s_A(c_j)) + (s_A(c_j), c_j)\}$$

(3)

2.5 Further Considerations:

In this paper, we focus on efficient usage of server which are available and the associated delays involved in the interaction between clients. Thus, the above problem formulation has not taken into consideration the processing delays at the servers. In general, the processing delays at the servers are easier to improve than the network latencies. A busy server can always be better provisioned (e.g., by forming a server cluster) to meet the capacity requirements and reduce the processing delay. We shall discuss in how to deal with server capacity constraints in our proposed client assignment algorithms if server capacities are limited.

3. Heuristic Algorithms:

In this section, we present heuristic client assignment algorithms. The computation of these algorithms is based on the server capacity limitation between clients and servers.

3.1 Client-Server Assignment:

First algorithm is called client-server assignment server which assign client to the available server based on the server registered order. Here each server have capacity limitation up to 3 client per server. After that limit, client redirect to the next available server. The computation complexity for each client assignment is $O(m \times n)$ where $m$ indicates number of server registered and $n$ indicates number of clients registered In the database. In this algorithm dynamic changes is possible i.e. any client join or leave at any point of time. In this algorithm we use onprocess to know number of clients assigned to server, and initially each server assigned to zero. We have status initially in the idle state keep on changing until server reached overload state. In this algorithm we have three states of the server are idle, normal, overload. In our system idle indicates no client to the server. Normal indicates one or more client assigned to the server. Overload indicates that server reached the maximum client assignment constraint (ie 3 we kept for our project).

**Initial algorithm**

1. $cl \leftarrow \{cl_1, cl_2, \ldots, cl_n\}$; // the set of registered client to the database
2. $sv \leftarrow \{sv_1, sv_2, \ldots, sv_n\}$ // the set of servers kept open for clients
3. All onprocess of each server is assigned to zero;
4. status of each server $\leftarrow \{\text{idle, normal, overload}\}$;
5. Initially every server is initialized with zero;
6. whenever clients want to assign to server check database for registered server
7. if no server found
8. Client will get alert “server is busy with work, please try after sometimes”
9. else assignment$(cl_m \in cl)$; here $cl_m$ indicates any registered client

3.2 Status change algorithm:

In this algorithm, status of the server used to change to different states depends on the constraint we kept. Here status initially in the idle. Whenever new client assigned to the server then changes to the normal. Once it reached to given condition ie 3 then status of the server reaches to the overload. After this server cant handle any client from that point.

**Status updation algorithm**

Status(onprocess)

1. if onprocess==0
2. change status--idle;
3. if onprocess<=2
4. change status--normal;
5. if onprocess==3
6. change status--overload;

3.3 Connection reset algorithm:

In this algorithm we use connection reset as the one of the functionality in the administrator in order make out the deadlock condition and make all server state to the idle and onprocess to the 0.
Connection reset functionality algorithm

Connection-reset()

1. Reset count of all servers onprocess to zero

3.4 Distributed-Modify Assignment:

Here s1 registered first, s2 registered second, s3 registered third. Here client assignment starts from s1, once given limit reached it cant accept anymore clients. Here next client will be assign to the next server that is s2 because it is registered server. But here client are not stable any one join/leave from the online system. So before assigning another client it will again look for the server for any space left if found it will assign to it otherwise that will assign to the next server.

Assignment algorithm

Assignment (clm)

1 check onprocess of server found
2 if onprocess<3 and onprocess!=3
3 then assign the particular client to that server
4 increment onprocess particular server;
5 status(onprocess);
6 else check onprocess of next server;// keep on move from first server to last one.
7. if onprocess of next server i.e., onprocess<3 and onprocess!=3
8. goto step3;

4. EXPERIMENTAL EVALUATION:

We have implemented the proposed concept on Java Platform in order to show the performance of our proposed dynamic resource allocation scheme is very accurate in sharing and storing the web site domains in a distributed server architecture dynamically for web site domain creation. For this we have used JEE container for implementing front end user interfaces as JSP, HTML as front end and for back end we have used MY-SQL as back end database.

4.1 Dynamic scenarios:

We have evaluated the performance of the proposed algorithm in static scenarios, where the client participation and network latency do not change. In practice, DIAs often support dynamic client participation such that clients may join and leave the network at any time. In addition, network latency may vary with time depending on network conditions. Distributed-Modify Assignment is more suitable for adapting the client assignment to dynamics in client participation and network latency due to its distributed and incremental nature.

4.1.1 Adaptivity to Client Joining/Leaving:

Our algorithm can be adaptable to the dynamic nature so any client can join/leave at any point of time. In our project whenever a new client come for the interactivity we look for the registered server in the order in which it is registered. Then we check onprocess of the first to last server if anything found the less than 3 then we take that server assign our client to it. After assigning we increment the onprocess of the server and change of the server. On other hand, whenever client logout of the network we used decrease of the onprocess of the particular server and change of the status of the server. It is the whole process take place and below is the algorithm present whenever client left from the our network

Algorithm for dynamic changes in the distributed server arrangement

1. If logout action is performed by the any client
2. then decrement onprocess of the particular server to which it is assigned
3. call status(onprocess)

5. CONCLUSION:

In this paper, we have analyzed the design, implementation, and evaluation of a resource management system in a distributed server architecture. Our proposed system uses first come first server allocation as main concept for resource allocation. Also we use the status of the servers as idle, normal, overloaded for the administration tasks. Our algorithm achieves both overload avoidance and load balancing for systems with multiresource constraints.

REFERENCES:


Chodipalli Sudhakar was born in Visakhapatnam Andhra Pradesh State India, 1992. His received B.tech degree from Kaushik College of Engineering, Visakhapatnam Andhra Pradesh, in 2012. He is pursuing M.tech (CSE) final year in Kaushik College of Engineering Boyapalem, Visakhapatnam Andhra Pradesh State in India.

Medara Rambabu received B.tech degree from Andhra University in 2004 and M.tech from JNTU Kakinada in 2010. He has 6 years of teaching experience. Currently, he is working as Associate Professor, HOD in CSE department of Kaushik College of Engineering Boyapalem, Visakhapatnam Andhra Pradesh State in India.

Kuna Venkata kiran received B.tech degree from MVGR vizianagaram in 2006 and M.tech from Nagarjuna University in 2009. He has 5 years of teaching experience. Currently, he is working as Assistant professor in CSE department of Kaushik college of Engineering Boyapalem, Visakhapatnam Andhra Pradesh State in India.