Auction System

Group 3: Peter Hösch, Sena Tarpan, and Yun Ye

1 Introduction

Our goal in this project is to design an auction system which enables a seller to sell a single product to a group of buyers. The buyers can place bids on the product. The product will be sold to the buyer who placed the highest bid at a predefined end date for the price of said last bid.

2 Project requirements analysis

2.1 Features

The seller defines a start and end date for the auction, describes the product as well as a starting bid. The buyers get informed about the the details of the product, the end date and the current highest bid every time said bid changes. Nodes can join or leave the auction at any time. The system should be able to recognize if a buyer goes offline or if a server crashes and take appropriate action. The server should be able handle high amounts of buyers by delegating work to supplemental servers if needed.

Table 1. Methods and Parameters of Servers and Clients.

	Server	**Client**

Methods	self_identification()
	server_logic()
	request_join()
	broadcast()
	validation_token()
	inter_server_communication()
	reply_to_client()
	election()
	update_data()
		self_identification()
		client_logic()
		request_join()
		raise_bid()

Parameter	**float** highest_bid **list** bid_history **dict** customer_list **dict** server_list **tuple** proxy_server_address	**tuple** proxy_server_address **float** highest_bid **float** current_bid **string** own_cookie **boolean** winner

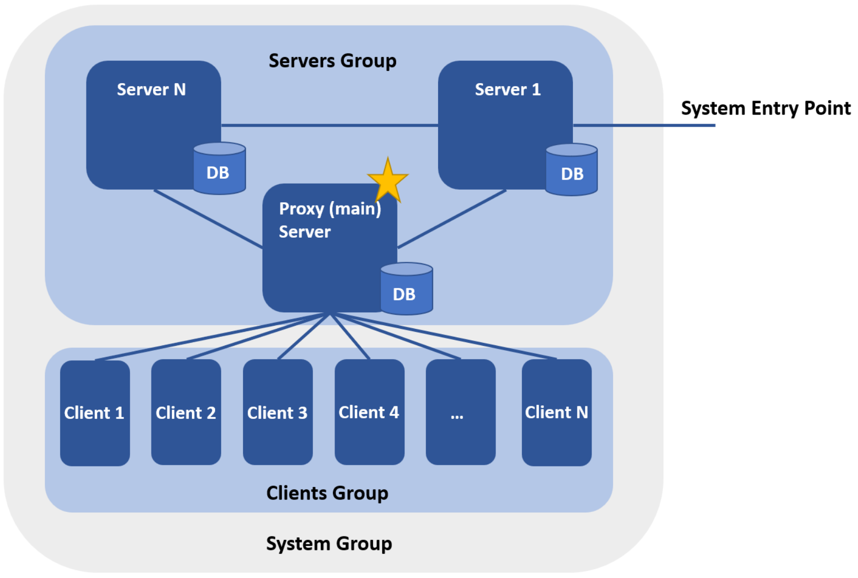

Fig. 1. Architecture diagram

2.2 Implementation

The system will be implemented as a many servers-many clients design. The servers are the seller who functions as the main server as well as supplemental servers that provide fault tolerance and scalability. The clients on the other hand works like a thin client machine that provide merely an interface with a very restricted logic and data functionality. Before the clients are accepted as one of the buyers, a validation cookie will be assigned to the client in case some malicious party tries joining the Auction. Should the main server stop responding, the remaining servers will vote for a new main server from among their group. Otherwise, the supplemental servers exist to take bids, aggregate them and transfer that data to the main server. For this purpose, each server will be connected to a number of clients. The clients only communicate with this server, not directly with the main server or with each other. All servers maintain their own database and regularly synchronize with each other. To ensure that we become aware of lost connections or crashes, all participants in the auction are required to send a heartbeat signal in regular intervals to the server. If this signal doesn't arrives in a certain timeframe, the system will assume that the node that should have sent it is no longer available and proceed to initiate actions to replace them. Bids will be placed by using TCP connections from a client to a server to ensure that the bid will be reliably transported, while the servers will use UDP with ordered reliable multicast for the host discovery process, to inform all interested clients in a raised bid, and to regularly synchronize the current time which is important due to the time critical nature of an auction.

If multiple bids with the same value get placed during a short timeframe the system should be able to determine which one was first and as such counts.

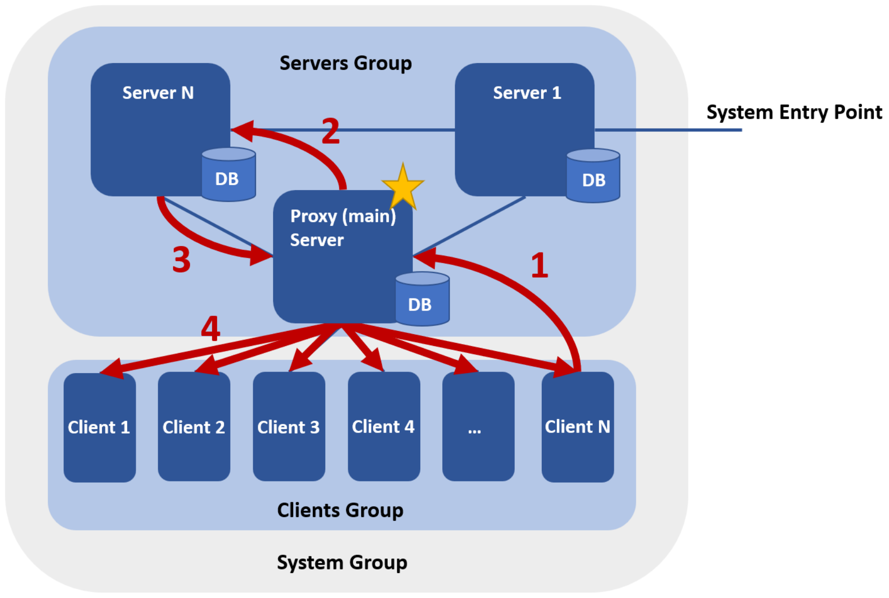

Fig. 2. Raise Bid Example: 1. The Client send a json to Proxy Server through TCP. 2. The Proxy distributes the request to one of the server or handles the request itself. 3. The server sends the result to Proxy. 4. The Proxy informs all the participants through UDP.