Distributed Computing Systems IN4391
The Dragons Arena System

Patrick Brand (patrick.brand2@gmail.com)
Raies Saboerali (r.a.a.saboerali@student.tudelft.nl)

Abstract

a description of the problem, system description, analysis overview, and one main result. Size: one paragraph with at most 150 words.

1 Introduction

(recommended size, including points 1 and 2: 1 page): describe the problem, the existing systems and/or tools about which you know (related work), the system you are about to implement, and the structure of the remainder of the article. Use one short paragraph for each.

2 Background on Application

The Dragon Arena System (DAS) is a online warfare game of WantGame BV between human players and computer controlled dragons. The game consist of a battlefield of 25x25 on which players and dragons reside. The goal of the game for the players is to kill all the dragons and survive (vice versa for the dragons). WantGame want the game to be able to support many concurrent users and has therefore decided to invest and design a distributed game engine which needs to be consistent, scalable and fault-tolerant.

As stated, the game needs to support multiple concurrent users playing on the same battlefield. This means that the system must be able to scale up in order to handle all these users and the requests they make to the servers. In order to achieve this, it is important to have multiple server nodes to handle requests of all these users. (This will be explained in more detail in the next section).

Multiple nodes means that different players may be performing actions which impacts a second player's actions. In order to detect this, the system needs to be able to detect whether another players action is still valid. A possible solution for this problem may be to use causal ordering for player actions.

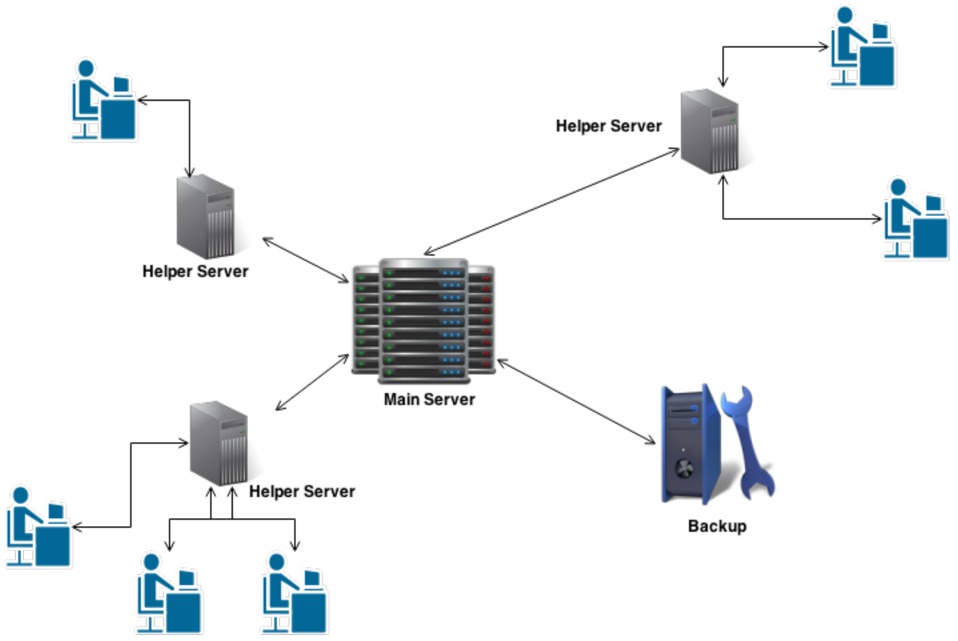
Last but mot least, players must be able to continue their game even if some server crashes. A possible approach may be, that there needs to be at least more than one copy of a running instance of the game to make sure that clients can silently reconnect to another server without disrupting the users game experience.

3 System Design

This section will first discuss how the system operates, by describing how the distributed game engine will operate. After this, the fault-tolerance will be discussed and at last the scalability components of the system.

3.1 System Operation

The general setup of the DAS system consists of Master Servers, Helpers Server, Backup Servers, and Clients. To initialize the game, one instance of the master server needs to be started. The master server will then contain the battlefield and a list of units playing on the battlefield. The battlefield on this master server only contains a set of actions which can be performed. The actual computation for these actions will not be be computed on the main server, but instead these will be computed on the helper servers. Therefore for the game to function, it is important to also initialize at least one helper server. Every time a helper server is initiated, it also registers itself on the main server. If a user starts the game on his computer, the clients then connects to main server requesting to join the game. The master server then randomly selects a helper server from the list of his helpers servers. The necessary details about the lucky helpers server is then returned to the client. The client then connects to this helper server and registers itself. The client also requests the battlefield itself in order to be able to make the correct decisions. If everything works well: the user ready to play! When a user tries to perform an action, the action is send to helper server. The helper server will then receive the request and depending on the type of action, the helper with request the required information from the main server. When this information is received, the helpers performs the required computation of behalf of the player and sends the results to main server. The server than send a message back to the client in order to inform him about the result of the action.



3.2 Fault Tolerance

Every distributed system is prone to faults, and the Dragon Arena system not an exception. First, lets look at the helper servers. As stated earlier, the clients connect to a helper server and register themselves there. When a helper server goes down, the main server notices this and removes this helper from its known helpers. If a client tries to query the helpers server when it wants to perform an action, it will detect that the helper server is down. The client then asks the main server for a new helper server. After the main server updates the client of his new helper, the client registers with the new helper and tries to execute its last action again. The new helper receives the action request and continues working as usual.

It is also possible that the main server goes down. The main server was the one containing the data about the battlefield and its participants. This is why it is necessary to have a backup server server running. In order to recover main server crashes, the system does check-pointing periodically. If during the periodic check, the main server detects that there has been changes to the battlefield, the main server send the changes to the backup server. The backup server then updates the knowledge of its backup battlefield and other relevant information.

If the main server goes down, it becomes impossible for the helper server to perform action on the main obviously. If the helpers detect such a problem, they immediately try to send the request to the backup server and asks the backup server to perform the action requested on the battlefield. The helpers also notify the backup, the failure of the main server and tell the main server that he needs to take over the tasks. Whenever the backup server gets such a notification, it promotes itself to being the main server and takes all the responsibilities of a main server on itself. The backup may be behind a few steps, because the backup is not made after each Unit action, but periodically. Updating the backup after each Unit action would cause overhead at the server.

3.3 Scalability

In order to support a large number of players on the battlefield, it is important for the system to be scalable. As stated earlies, the helper server are ones performing the computation for the main server. It is possible to add many helper server as possible which can be registered to the main server. Clients may then connect to any to these available helper servers in order to play the game.

The main server however is not scalable for distributing workload like the helper servers do. Instead, only a backup is created of the main server, which can be used if the main crashes. Since the helper servers are the ones doing the computation, it is also not necessary for the main server to have multiple instances for distributing the workload.

4 Experimental Results

4.1 Setup

During the implementation of the game, the game was mainly tested on the local machines. To simulate the distribution of the system locally, different RMI registries were used for the main classes which are supposed to run on different nodes. By doing so, the running classes need to call a different (local) server in order to make an RMI call.

4.2 Consistency Experiments

One important issue in a distributed game is keeping the game-state consistent for every client and server. This means that battlefield needs to be somehow updated for every client in order to be able to play the game. As stated in the previous chapter, the whenever a client sends a request, the required information is returned to the helper server the client is connected to. The helpers server itself does not maintain a local state of the game, but only computes results. However, the clients and battlefield do maintain a state of the game. Therefore it is important for the master server to maintain a correct state of the game.

5 Discussion

6 Conclusion

7 Appendix

References