Intermediate Report I

Mauraders Map, CS 5412
Haotian Pan(hp343), Yize Li(yl2376), Hang Chu(hc772)
March 11, 2014

1. Accomplished Goals

1.0 Introduction

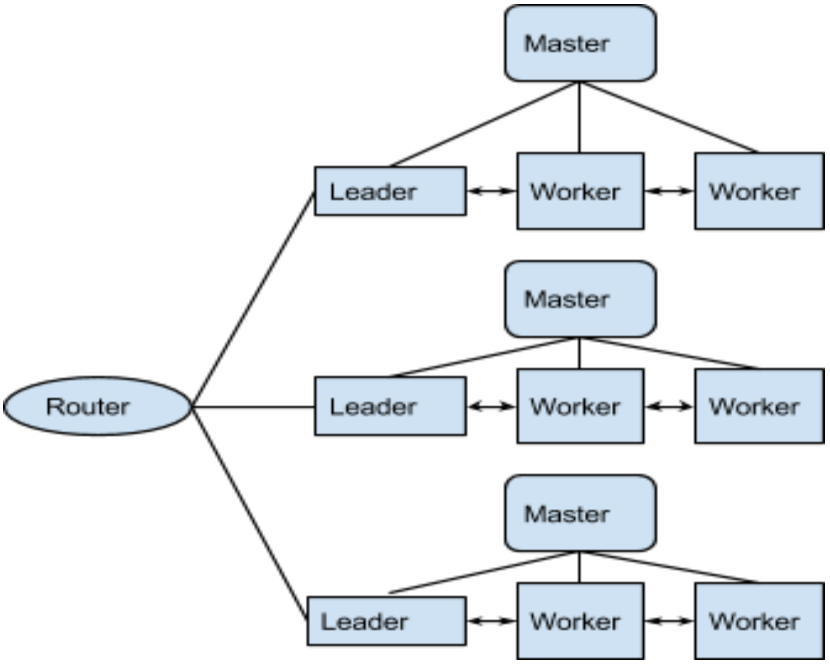
We have basicly followed the milestones proposed before, but have also made some changes to our schedule.

The figure below shows our structure of the cloud we are going to design in this project.

(1) We assume a router who is in charge of assign connecting clients to different leaders. In this case, the router is assumed to be absolutely stable, and it is also doing the load balancing as well.

(2) We use masters to manage server groups, and any group consists of three mutually replicated servers, one leader and two workers. This master is delivering some information(not all) of clients between leader and workers. For instance, we are planning to send client IDs in the crashed leader server to its two replica worker servers, to ensure dropped clients will get higher priority(e.g. by locks) when reconnecting if there are other new clients trying to connect at the same time.

(3) At bottom leverl we use three servers who are actually generated from the same file to do the replica. Upon launched, they will all connect to the master and one of them will be the leader and all the coming clients will connect to it. During the process of the game, every leader and worker will share the same data of the clients with the help of the master.



1.1 Milestone 1

1.1.1 Create the game map

Discription: According to our game model, the map is 2D. It can be either manually crafted, or can be retrieved from real-world map such as Google Maps.

Current status: We completed the game model, with a 2D manually crafted map.

1.1.2 Create game GUI

Description: We will implement a webpage-based GUI for gamers, which includes: a map display with real-time player status, buttons for taking actions such as move or attack (we will also try locating by GPS to substitute manual moving control).

Current status: We completed the GUI, with a BattleCity-like interface. We plan to change the tank images into zombies/humans and redefine the fire range, though we believe the how game itself looks does not matter in this project.

1.1.3 Traditional single server-client framework

Description: We will build a simple gaming framework, using only one server. Every user is connected to this server.

Current status: We completed a single server-client framework. The client access the game via a web browser, the communication between the client and the server follows the http protocol.

1.1.4 Testing maximum response speed

Description:We will test the real-time performance of the simple framework.

Current status: We have not evaluated the response speed in milliseconds but the response is quite fast, with stable instantaneous performance where human eyes could not detect delay.

1.2 Milestone 2 (Partially)

1.2.1 Improve communication efficiency

Description: In the previous stage we only test response speed, in this stage we will improve communication efficiency to meet the 100 ms requirement.

Current status: Our current implementation is quite efficient with response time far less than the 100 ms requirement, so we consider this goal accomplished.

1.2.2 Multi server support

Description: Only one server cannot be called a cloud. We will implement strategies to enable our framework working at multiple servers. Different game instances will be hosted on different servers.

Current status: Our framework now allows more than one server group, we divide existing servers into a master and a group of workers. The master can assign jobs to an idle worker when the previous active server crashes. We consider this goal partially accomplished as we have not implemented the router server for assigning players into different master-worker groups, now we just store the ip addresses of the server groups in the clients, and they will go through the ip addresses list to find a server group which is not heavy loaded.

1.3 Milestone 3 (Partially)

1.3.1 Error tolerance (replicating servers)

Description: In this case, we will duplicate the data of each server to two other redundant servers. Changes in any one of the servers would cause the other two to synchronize. This ensures that clients would not receive bad data or crash after one server crashed.
Current status: Our design of error tolerance is based on a master-leader-worker structure for servers we constructed for the servers. Every time a game instance is created, the master sever will assign one of the servers in the backup list to become the new leader and take charge of game holding. When the master crashes, one of the workers will take over immediately and become the new master. We have implemented the master-leader-worker structure but have not completed the worker fork up master process, so we consider this goal partially accompllished.

1.3.2 Clients should witness changes in a consistent way

Description: For this part, we may actually implement with the isis tool. We may come across some protocols to guarantee consistency between clients.
Current Status: We completed the process that the master assigns the new leader from the idle worker servers, thus when the current serving leader sever crashes a new leader will be assigned and put into work immediately. In this way, clients witness no fluctuation.

1.3.4 Maintain 100 ms delay limit

Description: Assume that we have made several changes from milestone 4.2.1 in order to meet the requirements, we still want to check and maintain a 100 ms delay limit.
Current Status: Our current implementation is able to maintain the delay limit.

2. Things to do next

We are going to get these things done in priorities.

2.1 Clients reconnection (High Priority)

Up till now, if the leader servers goes down, one of the leader server will take over and make sure no clients will be affected or even dropped from the game. But reconnection should be done in a faster and better way so that no players will sense an apparent delay and reconnection should not be bothered by new players who are trying to join the game at the same time.

2.2 Worker replica (High Priority)

In our current version, the user data are stored locally. When a new worker is assigned as the leader, the user data are sent from clients. This is not a sophisticated strategy, we intend to improve this by storing a back-up user data in the master server.

2.3 Router-load balance (High Priority)

As the number of players grows, only one master-worker group of servers is not enough. We should make our system scalable while balancing load. We are going to use a router to do that favor in the coming weeks.

2.4 Game logic (High Priority)

Up till now our game looks just like BattleCity, and neither do we have walls nor can we fire. We can only move our own tank and see other tanks moving. This is enough as some kind of “game” that we can test our server stability and replicating with, but it is still far from playable. We will change the game into a human vs. zombie appearance, with walls and attacking available.

2.5 Group (Medium Priority)

We have considered a kind of possibilty that our master, leader and worker servers could actually form into a group. This group can have no hierarchy when initialized, but should automaticly nominate a master, a leader and other workers when corresponding events happen. After that, the group members keep communicating with each other by broadcasting information throughout the group. This idea is inspired by the working mechanism of Isis2.

2.6 Worker fork up master, master fork up worker (Low Priority)

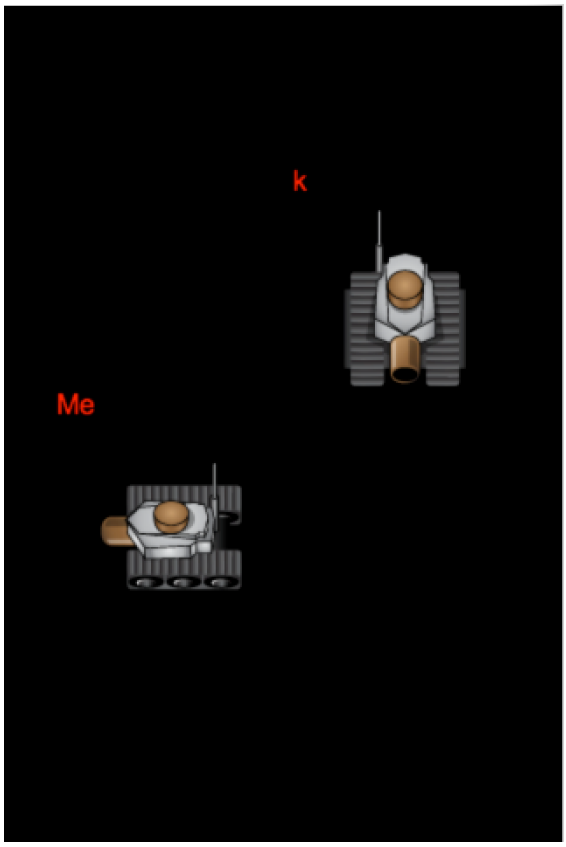
In the current model, we have not consider cases where a master could crash down. If that happens, the current leader and workers should somehow generate a new master. This can be done by immediately promoting one leader or worker to a master, or finding another existing master, or even launch a new master.

2.7 Master replica (Low Priority)

As mentioned above, we have not found a way to do master replica yet. Maybe we should reconsider the design of our system, because using something new like a “master” to achieve and manage replicas seems to bring us repeated questions.

3. Presentation

Some of our work are presented below:



The simple game screen

ZombieCloud — bash — 65×24

```
Last login: Thu Mar 13 17:45:03 on ttys000
You have mail.
Yizes-MacBook-Pro:~ yizeli$ cd /Users/yizeli/ZombieCloud
Yizes-MacBook-Pro:ZombieCloud yizeli$ node worker.js
connection successful
1
Listening for connections on port 8001
Logged in 127.0.0.1; currently 1 users.
k spawned a car!
Logged in 127.0.0.1; currently 2 users.
l spawned a car!
Yizes-MacBook-Pro:ZombieCloud yizeli$
```

ZombieCloud — node — 67×24

```
Last login: Thu Mar 13 17:45:28 on ttys001
You have mail.
Yizes-MacBook-Pro:~ yizeli$ cd /Users/yizeli/ZombieCloud
Yizes-MacBook-Pro:ZombieCloud yizeli$ node worker.js
connection successful
2
1
Listening for connections on port 8001
Logged in 127.0.0.1; currently 1 users.
k spawned a car!
Logged in 127.0.0.1; currently 2 users.
l spawned a car!
```

The consoles showing one server crashed down and another took over