DISTRIBUTED INNOVATIVE GAME WITH BASIC SCENARIO

A PREPRINT

Gerasimov Valeriy
Innopolis University
v.gerasimov@innopolis.ru

April 29, 2019

ABSTRACT

This paper is about the creation of a new fully distributed game. The idea of the game is totally new and it will be presented here. Also, you will find implementation notes and history of this project.

1 Introduction

In this chapter, I will explain my tries to create distributed application during the spring course of Distributed Systems given by prof. Paolo Ciancarini at 2019.

All starts with the idea of distributed poker application. I was afraid of RMI concepts in programming and at the same time, a have some experience at web programming and HTTP servers. So I begin with it using Python Django stack. And it takes me almost 1.5 days to create registration server, server for a distributed part, front-end for the user interface as plain HTML pages written in angular. And at the end of those 1.5 days, I have only weak registration using central server functionality.

At that moment I still did not want to go deep in Java to use RMI, because my previous experience with Java was quite bad. So I have found Python alternative for RMI, PyRO(Python Remote Objects). And start a blank project with it. And I have ended with registration only by several hours, and all this without using central point(that would be changed at the end of the project).

At that moment I had working distributed registration in the lobby. And I start to think about the game. Initially, I was looking for a solution, where I can select one referee that will manage the game, and then try to distribute trust from the single referee to all players. And at that moment a have several ideas about this:

- Referee select key

  The referee selected key. Distribute it among players. Then this key is used as a seed in random functions, to generate the same cards in a poker game. I get this idea from other distributed games, this seed was used for NPC in distributed games, their decisions were made according to seed, so all players have the same result, and no network was needed for it.

  This solution was good, since the referee is needed only 1 time per game, and all other time trust is distributed across players.

  But obviously, this is not working with card games, since everyone will know the cards of his opponent. It was the main reason, why I did not choose it.

- Referee select key and regularly change it

  After the failure of the previous idea I have was looking for an idea, when the referee is important in the game and regularly change the seed.

  This idea is better than previous because players can't predict future results, and can not do bets according to it.

  At the same time, if the referee's node will crash, then players can continue to play themselves.

  Even the fact the players can't know future cards in the deck, they still can get current cards of each player. So method with the same seed choosing by the referee was failed here.

- Referee have full control over the players

  This was an idea when players select referee, and then he has full control among players. This idea works if at the end of the game players would share an opinion, whether the referee gives someone dishonest advantage. So referee to get respect points should be quite fair.

  Of course, this idea has a single point of failure, if the referee fails, then the game is over and players just can return spent points, select new referee and start a new game.

  That was not the best idea but I have stopped here for a while.

So I have implemented a version of distributed poker that have single referee, that was selected every round. And after this, I found out that project requirements says to have shared data across players, and supports at least 2 crashes in the system. And my current version by that time did not meet those requirements. So i start to look for existed distributed card games solution, and find "Mental Poker" concept with next possibilities:

- Shuffling cards using commutative encryption.

  In this solution, each player have the same deck, but it is encrypted, so no one can get actual values of cards in it. If the player wants to get 2 cards from it, it has to get keys of them from other players, and if others share(after analyzes whether the player can get cards at the moment), then he knows the value of the card, but at the same time, others still do not.

  This solution requires commutative encryption. It means that the order of encryption and decryption does not matter if the keys of encryption and decryption are the same for accordance operation.

  This solution has a disadvantage in performance. Players have to share the deck several times before it becomes shared and ready for the game.

- A non-shuffling poker protocol

  This solution does not have deck distributed across players. Instead, players generate new cards, ensure that there are no collisions, and share an encrypted version of cards. So players can see that by a certain moment player have some card, but can not see it until player share its key. And at the end of the game, all players can check the fairness of each player.

  This solution requires additive homomorphic encryption so:

$$E(c1) * E(c2) = E(c1 + c2) \tag{1}$$

I begin to implement the first variant because commutative encryption can be made just by plus, minus xor and other basic operations.

But during implementation, I have realized that if someone crashes then other players cannot restore their cards since they would need for lost keys stored on the crashed node.

At that moment I was looking for a concept when lost keys can be restored by the cooperation of remaining or part of remaining players(for example when 3 players keys together can give the key of the fourth player). But this idea has no success.

Then I start to implement plain poker with a fully open world, to meet failure tolerance requirements.

But during implementation, I was thinking about the idea, when even some uncertainty was invented in game. And in the end, I got it.

In a poker game, players choose cards for themselves. In my game players choose just integer numbers.

Initially, I want to sum several integers by the same player and compare it to the others, and the player has the most value wins. But this solution has no uncertainty.

Then I modify operation. I get not only a sum but the sum under modulo(% operation). But if modulo will be known value the game still will not be interesting.

So then in my system, every player shares their candidate to modulo value and the average will be selected. In this situation, the player who will select modulo value last will have an advantage, since he already knows all other values. So I thought that in this case, players can share not modulo values but their encrypted version. So they show that some value was chosen(and other players can prove that it really did after they get keys), and everybody gets, and after all, players share their encrypted values, then they can share keys.

So that was a description of how this project was created. Which problems were found and which of them was solved.

2 Rules

As you can read in the introduction, I have a long way before getting the final idea. And this chapter describes its key rules and features.

Before every round registration, the phase has to proceed, but this is not important to then steps of the game. So we assume that at the beginning of the game N players have each other addresses and all of them know that the game has been just started.

From the beginning, each player starts to share M numbers with each of his opponents. These numbers are public, so everyone knows them.

After this players imagine one more number and share them at the same moment, so no one has an advantage because he knows all other results beforehand, so one of the game points is to predict last numbers of other players.

After the moment every player knows about M+1 numbers of each player (this means (M+1)*N numbers in total), the result of each player can be computed by next formula:

$$Result_i = (\sum_{m=1}^{M} r_{im})/C$$

$$C = (\sum_{i=1}^{N} l_i)/N$$

Where:

- Result is the result of Ith player
- r is the mth public number of ith player
- C is the common for all players number that is mean among all numbers of the players
- l is the last number of ith player

2.0.1 States of the game

Currently I distinct 2 states of the game:

- Numbers sharing state. When players publish their M numbers
- Last number sharing. In this state every player have to imagine his number without any information about last number of other players

After sharing last numbers the game is terminated.

2.0.2 Future works

The game, its idea, and rules came from imagination. No game design or math analysis has been conducted. So I have no information, what advantage can players get if they will wait for results of other players' public numbers before publishing their ones. Maybe the strict system of moves can be introduced. Anyway, before further development, such analysis has to be proceed.

3 Big picture

3.1 Main features

In this game I have next features as main:

- P2P registration. This means that everybody can create a lobby without any centralized entity(except web).
- Absolute fault tolerance. So every peer can fail and the last become winner.

3.2 Problems and solutions

During development, I have met several problems and this section describes which one I have met and how I have solved them.

3.2.1 Sharing secret info

As you can read in the rules section, the game have a state where everybody shares their numbers at the same moment(i.e. make self-decision without information about other ones). In case we have a centralized instance, then it would be easy. But in the distributed setting, it is not an obvious task.

To solve it I use zero-knowledge proof concept. Where instead of the number we publish its encrypted version. When all peers get encrypted version of numbers(at that moment they will know that each player already has done its decision and it cannot be changed), peers share their encryption keys. So everyone can see all information and compute results.

3.2.2 Sharing the same information

Currently, every peer basically sends a new number to each other player, like in broadcast systems. But each player that receives results can not check whether the sender sends the same information to each peer and not trying to fraud the system.

A possible solution is to send a check request to all other peers to get acceptance.

3.2.3 The origin of sender

When sending any info happens, the sender use next notation:

$$otherPlayer.send(info, senderName)$$

It means that sender just says: "I am really I, and my name is <some name>", and the receiver says: "Ok, i believe you". This is a big hole in security.

But fortunately this is easy to solve by SSL protocol, where private and public key pair exists.

4 Implementation

In this section we discuss architecture, technologies used and algorithms implemented.

4.1 Assumptions

But before the implementation, here some assumptions will be listed, that is not always the case in real life, but very seldom to keep it in mind to develop prototype.

4.2 Architecture

To implement prototype the all to all distributed scheme is used. So everyone can share information to every other node.

4.3 Technologies

Since I have enough experience with Python programming language, I use it. For remote objects, I use PyRO4(Python Remote Objects) library.

4.3.1 Network reliability

This point means that if some Communication errors occurred then this is local problems of that peer. Maybe it has crashed or problems with internet provider occurred.

4.3.2 Network is homogeneous

This means that if the first peer cannot send information to the second, then no other peers can connect to the second one. In real life, the WEB can fail partially, so this case is possible but too seldom

4.3.3 Peer crashes forever

This means that if some peer fails then it cannot recover itself and reconnect to the game.

4.4 Algorithms

4.4.1 Registration

The registration way is pretty straight forward and can have security issues. Every player can create a lobby, then it will get its address. So new player knowing the address of creator can access to his lobby and then get his new address. So the third player can connect to the arbitrary player. And the player that already in the lobby will share information, that new player is connected.

The peer that create the lobby at the moment, when the 4th player(the number can be specified) connect to the lobby, sends an initial signal about the start and the game begins

4.4.2 Failure tolerance

To detect failures our peer catch-all communication errors, if it got one, it will delete failures node and notify other players that some fail occurred, then all others also check that and also will notify all the others. And after each healthy peer will check N times(each time when got error or notify about error from other players)

5 Conclusion

While working on this project we got a new idea of the game, design distributed solution of this game and create a working prototype of one peer in "peer to peer" system. New concepts of remote objects were learned and practiced, and I got experience in developing distributed systems as well. But the current solution has several issues and this section is about future works on this project.

5.1 Future works

5.1.1 User interface

The current implementation is just a command line interface and it is not convenient to share information of other players state and ask to input the self player's one. Creating user interface would make it easy.

5.1.2 Security

In the section about problems was listed possible solutions, with them, the system would be in a ready state for real plays. For now, it is just a prototype for debugging and testing.

5.1.3 Fault tolerance

In the current system, the peer can detect that some other peer crashes, also peer can delete it from self lists. Since that it is also able to realize that it is the last one and proclaim itself as the winner. But no information is showed to the user in this case.