Modelação e Validação de Sistemas Concorrentes

Project

Pedro Valente
Nº 50355

Rafael Gameiro
Nº 50677

Introduction

This work consisted in the implementation of a simplified version of the Bitcoin mining operation, known as Proof-of-work. Our task was to develop a server and client that would trade messages between them, executing an operation like the one done in the Bitcoin protocol. The client would be responsible to make requests to the server with a message M, and a value N, while the server would concatenate the M with all n between 0 and N, generate a hash value and compare between them. The largest hash generated and used n would be sent back to the client. Our program should follow a set of criteria and in the remaining of this report we will address our design choices and explain if our implementation does follow the imposed requirements.

Implementation

The project was done using the Go language. To meet the required criteria, a set of mechanisms from go were used, as some logic to correctly execute the requested operations.

The program starts when the server is instantiated in a specific port, given as an argument. The server will then listen to requests sent to same IP address and port the server is in. After receiving a request to establish a connection with the server, a go routine will be created to take care o the request, making the server available to future incoming requests.

The go routine will receive the message sent by the client and process it. If some error happens while receiving or processing the message, an error is thrown, and the connection is closed. After receiving the request, a go routine will start to create a work pool of workers to attend the request inside the message.

The number of workers allotted to a request is based on N, however it is not uniform in order to keep the server responsive. Up to a point the number of assigned workers scales linearly with N (number of workers = N / 40 + 1), this ensures that small requests are processed quickly. Past a certain threshold (N > 300), the number of workers is based on log(N), this way overwhelmingly large requests will not hog too many resources, keeping the system fair and responsive.

After determining how many workers should be used, the interval between 0 and N will be split evenly among the workers. Then, each worker can start generating hash values and comparing the generated hashes to determine which one, inside their assigned interval, is the largest one.

To aggregate the hash computed by the different workers, each one will send its hash through a channel to a go routine that will receive the values and compare them, determining which is the largest. After receiving a value from each worker and comparing them, a response will be sent to the client, containing the largest generated hash and the nonce used to generate it.

After doing experimental evaluations on the program, we could conclude that our server is correct, by sending back to the client the largest generated hash between 0 and N. Furthermore, in case the connection between the client is closed, the server correctly reacts to it, by ignoring the hash generated by each client, terminating the request.