JSMiner - 512 Final Project Report

Howard Chung, Kent Jiang

April 22 2015

1 Introduction

Bitcoin is a digital cryptocurrency made up of processed data blocks used for online and brick-and-mortar purchases. Because bitcoins are limited and their value is determined by market forces, bitcoins are also traded like stocks on various exchanges. The purpose of Bitcoin mining is to create new bitcoins as well as ensure that all participants have a consistent view of the bitcoin data.

JSMiner is an attempt to engage in bitcoin mining in a distributed manner–namely, through visitors' web browsers. Traditionally, bitcoin mining has been done through standalone programs, which generally require administrator permissions to run. By moving the computation to the browser, a visitor merely needs to visit a web page in order to begin mining.

2 Implementation

JSMiner consists of two components:

- The first is a client application, consisting of HTML and JavaScript.
- The second is a Node.js server which coordinates the work done by the clients, and serves as a "bridge" between the clients mining in the browser and the bitcoin network.

In this implementation, bitcoin mining is done through a pool. Due to the currently high difficulty of mining, "solo" mining without a pool can take years in order to find a block. By mining in a pool, the variance is reduced. The pool assigns work (shares) that are easier than the work required to find a block. When someone in the pool is successful in mining a "real" block, the pool distributes rewards among participants based on the number of shares they submitted. The server connects to a mining pool using the Stratum TCP protocol. This allows the pool to "push" new work. For example, when a miner anywhere in the world finds a block, the pool needs to notify all participants of this change and assign new work. When our server receives this "push" from the pool, it recalculates the block header that clients should hash.

The server has three HTTP endpoints:

- /: The root of the site serves the UI, and allows users to begin mining

- /work: This returns a JSON-encoded job that the client should use to begin mining
- /submit: This endpoint allows the client to submit a successful nonce to the server.

Clients, on connection to the server, get the current block header from the server and begin hashing. Each client selects a random nonce to start hashing with, reducing the possibility of clients doing "overlapping" work. Each time the client fails, it increments the nonce by 1 and tries the hash again. Thus, the client can attempt 4.2 billion hashes (the nonce is a 32-bit integer) before getting new work from the server. When a client successfully finds a nonce that produces a low enough hash (lower than the target value), it uses the submit endpoint to send the nonce to the server.

The server then forwards this work to the pool, which credits it toward the user's earnings.

3 Difficulties

Connecting this application to the live bitcoin network proved challenging. The client-side hashing example we began with used the old "getwork" protocol, which relies on polling the server for new work (once a second). We had to research the Stratum protocol and write the code to convert the Stratum data into a block for the client to mine.

In addition, it was difficult to test actual share submission, since bitcoin mining on CPUs is so slow that it would take a very long time in order to find a valid share.

4 Performance

By using WebWorkers, the JavaScript miner achieved speeds of approximately 80Khash/s, considerably slower than a C-implemented miner.

5 Extensions

At this stage, bitcoin mining on CPUs is largely inefficient. While possibly profitable for users if run on "free" electricity (from an institution or similar), it has largely been superseded by superior technologies. ASICs (Application-Specific Integrated Circuits) have proven to be much more efficient in terms of hashes per watt, and currently make up the bulk of the bitcoin mining hash power.

A further optimization would be the use of WebSockets in order to allow the server to push new work to clients. While the Stratum protocol takes care of the pool pushing work to the server, the server and client currently do not have a two-way connection. This would lead to slightly improved efficiency, since any work done by clients before the server is able to notify them becomes "stale" and is thus wasted.

Another interesting direction to go in would be to try to utilize the GPU using WebGL. Traditionally, GPUs have proved to be far more efficient at the SHA256-hashing required in bitcoin mining. However, the point is likely moot as both CPUs and GPUs have been superseded by ASICs. This idea might be useful if applied to "altcoins" such as Litecoin, Darkcoin, and others designed to be "ASIC-proof", through high memory bandwidth requirements.