A Computational Study of Consensus Efficiency

Implementation of Blockchain Framework

Richard Simon

A proposal presented for the study of consensus efficiency as it relates to the degree of Masters of Science in Computer Science

Computer Science
Southern Methodist University
United States
15 May, 2018

Contents

1 Specific Aims

Blockchain Technology is poised to change the field of computer science forever. It is theorized that this tech will touch near every industry in some way sooner rather than later. With this in mind, it is more important than ever to understand, in depth, the mechanisms that make blockchain possible. Specifically, understanding the method by which a given system achieves distributed consensus - the most important duty of a blockchain network - yields a great depth of understanding of the system's performance, and more importantly, its viability in given use cases. To date, this is an area lacking in a specific type of analysis. Many schemes have been devised to lessen the computational burden placed on the nodes that make up this system, with some achieving more success in practice than others, however, a system for testing and profiling these different implementations is yet to be created.

The specific aim of this study is to remedy this, to create a framework which will aid in the study of the efficiency of consensus.

2 Abstract

By definition, blockchain technology operates as a distributed system, meaning that its component machines, or nodes, do not share resources, or a centralized clock, and do their work in a vacuum apart from any other components of the system, save for incoming and outgoing messages from the other constituent components. This presents a set of challenges that appear in any distributed system, which have been studied at length. However, because of the nature of a blockchain's specialized task, some efficiency bottlenecks have yet to be studied; namely, the consensus protocols that are seminal in allowing for decentralized transactions. The most famous of which, of course, being Proof of Work.

Consensus algorithms(PoW specifically) present a major challenge in the form of replicate work. Because, by design, each node on the network is working to add the next block to the chain(excluding a select few robust implementations) whenever one is discovered, the work done in parallel by the other nodes on the network is for not. Imagine 10 people sitting in a room trying to solve rubix cube puzzles, and whenever one person finishes their puzzle, the other 9 participants have their cubes taken away, and everyone starts on a new puzzle. This would be immensely frustrating. In

the world of distributed systems, however, the problem has a compounding factor: latency. In our rubix cube analogy, this would be akin to, when a person solves their puzzle, each remaining participant's cube is confiscated, but only after an amount of time has lapsed first. That is, seconds or even minutes pass while the participants continue working on a puzzle that has been solved, only for them to find out that this work was unnecessary.

For a system such as Bitcoin, or any other major cryptocurrency, this waste is tolerable, as the security of the network is paramount, and miners are ultimately rewarded for their efforts when they do eventually create a block. Conversely, for a private system, such as a blockchain that maintains healthcare or accounting records, this waste is expensive for those operating the network, and detracts from the operating potential of the network itself.

More problematic yet for this type of use case, faster "block times", or the target time it takes for one block to be created(more on this later), create more waste, as the number of latency periods increase, and so does the overall waste of the network. This is a big problem for currency applications, and record keeping systems alike, as many such systems require speedy transactions, certainly faster than the target block time of Bitcoin, which is 10 minutes.

As mentioned before, robust implementations of consensus exist that alter the dynamic of mining, and instead of pitting miners against one another to solve a computational puzzle, create a more deterministic, cooperative mining network. This paradigm shift lends itself nicely to private blockchain applications. While these implementations do eliminate computational waste, they certainly aren't without flaw. The aim of this endeavor is to help better understand the shortcomings of various consensus methods, and to minimize the work associated with doing so.

3 Methodology

Today, shortcomings of various blockchain implementations are identified only after deployment. Because the systems are distributed, rather than centralized in one environment, it is often difficult to predict problems that arise until they are burdensome to the system. This has been the case with issues related to scaling, introduction of different types of hardware into global mining pools, security vulnerabilities, and more. When dealing with centralized systems, simulation of the environment is easily achieved. For example, modern debuggers and profiling tools discover errors and

vulnerabilities in software easily. For software that is intended to be used on multiple hardware platforms, emulators exist to identify potential issues with that portability. For blockchain technology, such tools are hard to come by, if they even exist at all. Because of the looming ubiquity of the technology, this needs to change.

Currently, it is possible to simulate the lifecycle of a distributed application in a centralized environment, however, this comes at the cost of time spent refactoring large segments of code, and the overhead of a scaffolding to simulate the different components of the application.

3.1 Implementation

The goal of this project is to implement a solution that will establish a method to easily simulate a distributed application, while eliminating the need for the refactoring and additional programming described above. The solution will be implemented as follows:

3.1.1 Scaffolding

The most time consuming part of refactoring distributed code to run centrally is setting up a scaffolding to simulate a distributed network. This is the starting point from which the framework is be built. The scaffolding itself is implemented on top of OpenMPI, where the simulated nodes run as separate processes. The separate processes, making use of the MPI interface, can then run on separate hardware as easily as they can in a centralized environment. For the purposes of the build, SMU's Maneframe II cluster is used as a means to distribute the workload over different pieces of hardware, as it would occur in a live distribution. The "master process"(process ID of 0) will spawn the environment, keep analytics(more on that later), and simulate user interaction with the blockchain in as organic a manner as possible.

3.1.2 Mining

For mining, the process of maintaining consensus in the system, to be simulated as it would in a live distribution, the solution will spawn a process to represent each simulated node on the network. This will require several steps, which will vary from implementation to implementation(of the consensus protocol used). The first step(assuming a gossip protocol is used to disseminate the node's messages) is to establish connections with neighboring nodes in the network. This

will allow for the spread of transaction and mining information throughout the network. Next, the mining node will need access to the current copy of the blockchain. For this to happen, the node will make requests to its neighbors for their copies of the blockchain. In keeping with the original implementation of the blockchain, the longest chain(the one which has the most work put into it) will become the requesting node's copy. This node will then broadcast that copy to its neighbors, ensuring that their copies are all up to date. Finally, the node will make requests to neighboring nodes for the pools of transactions that they are working with. Because transactions are sent to nodes around the network, they are not perfectly distributed, and each node will have different transactions to work with. After this setup has finished, the node will then start grouping transactions into potential blocks, and working on the hash puzzle to mine the next block.

3.1.3 Simulated Latency

To keep the simulation honest, the hardware connectivity of the host system will have to be restricted, as M2 uses gigabit and infiniband connections, which will vastly outperform latency and bandwith metrics of a deployed network. This difference would affect the quality of the data collected for analytics of the network implementation.

3.1.4 Analytics

As mentioned above, the master process keeps certain data about the simulated runs of the network. The most important data in the present implementation is the amount of wasted CPU(or GPU) time due to latency. Nodes will achieve this by timestamping each block as it is created. When a node receives an interrupt message signalling that a block has been mined, the node will compare the timestamp of the block to its system clock, and calculate how many clock cycles have been spent in futility. While the cluster on which the simulation will run is distributed, the clocks are synchronized closely enough that it will not greatly impact data collection(differences of milliseconds or less). This is the primary metric to be collected for early implementations of the framework, however additional metrics may be added based on the configuration of the simulation(see below).

3.1.5 Customization

As outlined in the abstract, the purpose of the solution is to add flexibility in simulating distributed systems that utilize blockchain technologies. To accomplish this, the system described herein must be agile in its simulation ability. There are 2 basic ways to achieve this. The first, and most easily implemented, is to document the build well enough that an end user may copy and alter modules of this project to fit their needs. However, this does not well solve one of the premises mentioned above: it makes simulation and profiling time consuming and cumbersome.

To solve this, the design choice has been made to incorporate an existing technology that can inject user code in the appropriate places. XSLT, a popular tool used in parsing XML data, is incorporated, which affords end users an easier method of customization. The XML format to be parsed will be organized so that each parameter for execution, from the number of participant nodes, to the simulated network latency, is fully customizable to suit the needs of the end user.

Additionally, 2 methods will be provided for an end user to customize their network implementation. The first, and easiest, will be to allow the user to specify paths to the corresponding files in their source code, along with a path to any makefiles needed to compile the application correctly. The second, more cumbersome choice, is to allow the user to include their mining code directly in the XML file. XSLT will differentiate between these 2 options by taking paths as a tag parameter, where code will be included as the body of a tag. The fully customizable nature of the framework will make it attractive to developers for testing cases in which they are unsure if they have an optimal solution. For example, scaling is a prominent issue in this type of application, as distributed systems incur much overhead when scaling. Parameters that might be useful in this test case are the number of participant nodes, number of blockchain client users (simulated), and network latency parameters.

4 Future Steps

In its present state, the solution's feature set is not complete. In order for the product to meet the needs of potential developers, it, too, must be developed with care, and tested to exhaustion. This will require many intermediate steps, not the least of which is completing the customization engine. From there, the engine will need to be tested with the consensus protocol that is currently hard-coded into the solution. Once complete, formal testing of alternate consensus implementations

will be possible, and necessary. I expect this project to span several semesters to be a fully tooled solution, and as such will be completing code sprints over the course of the summer and into the coming semesters. The primary objective for the coming sprint is to complete an XSLT based customization engine for the project, which will then enable me to add other mining implementations, and test those.