CollabChain: Blockchain-backed Trustless Web-based Volunteer Computing Platform

Sagar Bharadwaj KS*, Samvid Dharanikota*, Adarsh Honawad*, and K Chandrasekaran

National Institute of Technology Karnataka, Surathkal, India
{sagarbharadwaj50, samvid.dharani, adarsh2397}@gmail.com, kchnitk@ieee.org

Abstract. Volunteer computing is a distributed computing model in which individuals in possession of computing resources volunteer to provide them to a project. Owing to the availability of billions of computing devices all over the world, volunteer computing can help solve problems that are larger in scale even for supercomputers. However, volunteer computing projects are difficult to launch and deploy. These platforms also force volunteers to trust the authenticity of the project owner and to blindly accept credits allotted to their contribution by the project owner. As a result, very few high-profile trusted projects are able to sustain in this system. In this paper, we present an incentivized web-based volunteer computing platform that functions as a market place to buy and sell computing power. Launching a project on the system and contributing to an existing project happens over the browser without the need for a specialized software or hardware. We introduce the application of blockchain to remove the need to trust any other party in the system. We also present a prototype implementation and solve NP-Problems as examples using the proposed prototype.

Keywords: Volunteer Computing · CPU Cycle · Blockchain · Web · Browser

1 Introduction

Volunteer computing refers to a distributed computing solution where users (called volunteers) contribute their computing power to large-scale projects requiring high throughput and longer computation time. These large-scale projects are usually, but not limited to, research problems in the areas of medicine, meteorology, mathematics and so on. Some examples are Folding@Home [4], that simulates protein folding, computational drug design, and other types of molecular dynamics and Einstein@Home [3] that searches for radiations from neutron stars.

Approximately 10 petaflops of computing power are available from volunteer computing networks [15]. Anderson et al. [9] provide further statistics on the potential of volunteer computing.

* Contributed equally

In a general volunteer computing model, volunteers pick up tasks (processes) given by a host, perform necessary computation, and submit the output back to the host. Most volunteer computing architectures today are usually supported by unpaid volunteers. They are 'volunteers' in the truest sense. These models can also be based on 'volunteers' 'selling' their resources, and the host 'buying' them. For example, a host that requires high-throughput computation can request a volunteer to run the computation in exchange for payment made to the volunteer. Such an architecture is feasible due to the fact that they do not have to invest in expensive cloud servers or any other computation platform, but can 'buy' computing resources for smaller amounts on a per-process basis. This also allows for systems to accumulate monetary reward during their idle time.

The Blockchain in its basic sense is a distributed ledger that is replicated on all nodes in a distributed system. As the name suggests, it consists of a 'chain' of blocks, that are linked to each other. These blocks contain transaction records and associated data. The hash of the contents of a block is recorded in the following block, and this forms the 'link' between the two blocks. This essentially makes the ledger immutable and append-only, because, tampering of data in a certain block would require re-computation of its hash, and the subsequent modification of this hash in the following block, and this propagates. Generation of these blocks by nodes in the blockchain network is restricted by 'consensus' mechanisms. Hence tampering with the blockchain and creating new blocks is not feasible.

Nodes need only a pair of public and private keys to participate in the blockchain network. This implies that they do not have to use any personal data such as their name, for instance, to create an identity in the network, preserving their privacy.

The blockchain is replicated over all nodes in the P2P network, and its state is made (eventually) consistent over all nodes by exchange of state-update information with its peers. The lack of a central node/server to maintain and enforce the current blockchain state ensures a decentralized environment.

Properties of blockchain such as immutability, privacy and decentralization make it an attractive solution to many development problems that require the same properties. It was initially used as the ledger for the famous Bitcoin [11], where the blockchain stored Bitcoin transaction records. Blockchain has since evolved, and saw applications in many domains not restricted to cryptocurrencies, such as healthcare, Internet of Things (IoT), cloud computing and more.

This paper proposes a novel architecture of browser-based volunteer computing platform that employs blockchain to provide a trusted environment for the volunteers and incentivizes them to devote their computational resources. Without the need of any specialized application, except just the web-browser that is pre-installed on almost all commercially available personal computers, anyone can deploy the platform and use it with ease.

The paper starts with an introduction to volunteer computing and blockchain concepts in general. The following section explores other research works related to volunteer computing. We then define the design goals that our volunteer

computing platform must implement in the third section. The fourth section describes the architecture of our volunteer computing platform along with a sample implementation. The fifth section explains the flow of control through whilst using the platform. The final section displays the results of some our tests and drawbacks of our architecture.

2 Related Work

Many researchers and organizations have attempted to create browser-based volunteer computing platforms with varying levels of complexity that cater to general or specific use-cases. The earliest popular volunteer computing project was perhaps SETI@home (Search for Extra Terrestrial Intelligence) proposed by Anderson [8]. The project consisted of a high frequency feed from a radio telescope whose signals could be analysed to determine the presence of extra terrestrial life. Although the project failed to identify extra terrestrial life, it paved the way for *Public-resource computing*. The same group launched a volunteer computing platform called *BOINC* (Berkeley Open Infrastructure for Network Computing) [7]. It was the first platform that allowed participants to volunteer for a number of scientific projects that had massive computation requirements. BOINC projects use a centralised server complex centered around a relational database that stores descriptions of applications, platforms, versions, workunits, results, accounts, teams, and so on. The BOINC project has a large entry barrier for addition of projects. Around 30 projects use BOINC today [6]. Every project must maintain a server complex containing a database server and a web server. BOINC is thus only used to contribute computation power and seldom to request it. The volunteers will also have to spend significant amount of time downloading and setting up the BOINC client. BOINC is also based on a trusted 'credit' system. The volunteers trust the project host and receive credits after completing their share of work. Our design removes the barrier of entry for addition of new tasks, decentralizes the system and also eliminates the need for trust between exchanging parties.

Golem [5] is an incentivized computing market place with based on blockchain. However the tasks that the volunteers on the system can perform are limited to computations such as computer graphics rendering and certain machine learning algorithms. Thus the computations that can be performed are not generic and the platform cannot be used to request computation power for generic tasks.

BOID [1] is another blockchain-backed social computing platform where volunteers get paid in custom BOID cryptocurrency tokens for the resource that they contribute. It is currently in Alpha phase.

Sarmenta and Hirano [13] have proposed a Java based applet, Bayanihan, for a web-based volunteer computing system that consists of worker and watcher clients that connect to a server (via the URL in the browser) that serves 'problems' to the volunteers (clients). Note that Bayanihan is only a framework that can be used to build platforms on top of it with the mentioned underlying architecture. In contrast, our paper is a full-fledged platform.

Ong et al. [12] proposed a volunteer computing platform using a client-broker-host model, based on Java. Clients submit tasks that are distributed to the hosts via the broker, which handles task collection and distribution. Similar to our architecture, the hosts are incetivized by the client based on the amount of data they process/compute on.

Specific to image processing computation, a client-server model is proposed by Zorrilla et al. [16] where the clients are users on a social media service, who run a given algorithm on images from the social media provided by the server. The platform uses JavaScript, similar to our architecture, and spawns threads to perform computation in the browser in the background. It is worth noting that the more recent works on browser-based distributed computation have used JavaScript over Java. One key reason is due to the lack of the need of a separate Java Virtual Machine on JavaScript-based browsers, that needs to be installed on a system to view Java-based web-pages.

Turek et al. [14] also proposed a volunteer computing approach, based on a similar client-server model, for a specific use-case: Web-crawling. Servers first download the content of the web-pages required to be crawled. They then send the content to volunteers and collect the results computed by them. The volunteers initially obtain the algorithm to be run on the web-page content. They then query the server to obtain tasks. On receiving the content, they process it with the given algorithm and return the information to the server.

Although not a volunteer-computing application itself, Merelo-Guervós and García-Sánchez [10] proposed a browser-based distributed computing model for evolutionary computation.

3 Design Goals

Figure 1 shows the architecture of the volunteer computing platform that we have built.

We ensure that the architecture we have proposed adheres to the following design goals:

- *Easy Deployment*: Deployment refers to the steps needed to be taken to make a software system available for use. We aim at building a system which involves minimum effort to deploy. Existing volunteer computing platforms generally have a large deployment overhead, especially for the party seeking computation resources - the task submitter in our case.
- *Decentralization*: Most volunteer computing platforms depend on a central entity to monitor executors, collect results and distribute credits. This means the system is prone to single point of failure problems. We wish to remove centralized entities and make sure there is no single point of failure or a potential bottleneck in the system.
- *Platform independence*: Software systems in the volunteer computing domain are dependent on hardware resources and the underlying platform. Authors have developed clients separately for different platforms. However,

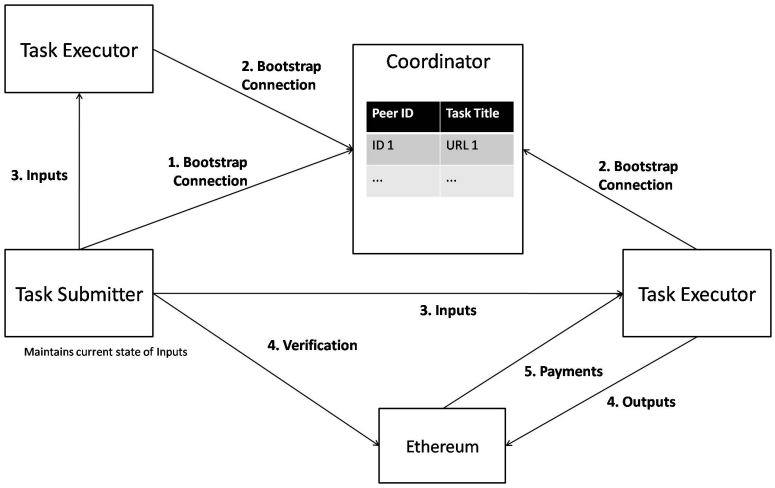

Fig. 1. Architecture

we propose a system that is independent of the platform and underlying hardware.

- *Trustlessness*: All volunteer computing systems existent today are reliant on the assumption of presence of trust between participating entities. Submitters rely on executors to not attack the system and executors trust the submitters to distribute unbiased credits proportional to the amount of computation. This trust model is justified as long as nothing of real value is exchanged. However, such an assumption cannot be made when it comes to an incentivized platform. We propose a system where the concept of trustlessness is achieved using blockchain.
- *Privacy*: Privacy refers to a one's choice of withholding private information about oneself from a system. The system developed, although involves incentivization must be capable of payments without the need to discover the individual's personal credentials.
- *Elimination of software and hardware overhead*: Existing volunteer computing platforms mandate the Task submitters to maintain dedicated hardware to sustain the system. We wish to eliminate these software and hardware overheads.
- *Performance*: The system should ideally provide a linear increase in performance with respect to the number of executors.
- *Scalability*: The system should be scalable with respect to the number of nodes as well as the number of tasks it can handle.

The current architecture that our platform uses implements almost all the mentioned design goals to a great extent. However, scalability and decentralization are two design goals that can be improved upon in further works.

4 Proposed Architecture

Two types of nodes participate in the underlying P2P network:

- *Task Submitter*: Any node in the system that wishes to submit a task to the system
- *Task Executor*: Any node in the system that wishes to execute a task submitted to the system. It contributes computing resources to the system and gets incentivized for it.

Note that the executors in our architecture are not true 'volunteers', i.e., they seek some gain from the work that they have contributed. Henceforth, the Task Submitter will be referred to as 'Submitter' and the Task Executor as 'Executor'.

4.1 Coordinator

A coordinator node is also present in the network along with the Submitters and Executors. Nodes initiates their connection to the network by connecting to the coordinator. Every peer to peer system requires a bootstrap connection to a seed node through which it discovers other peers in the system. The coordinator in our system acts like the seed node. The coordinator has two tasks:

- To maintain a database of all the projects utilizing the volunteer computing platform. This enables any Executor node to discover all projects and choose the project of its liking.
- To act as a seed node and let peers discover themselves to initiate connections among themselves. The coordinator acts as a brokering connection initiator.

After the nodes have connected to the coordinator, which only acts like a bootstrapping seed node, they no longer need to communicate with the coordinator, except for heart beats to confirm the node is still alive. The Executor node obtains the 'peer ID' (can also be thought of as a 'task ID') of the Submitter of a particular task through the coordinator's database. Note that the same Submitter node can have different peer IDs for different tasks that they submit to the network, i.e., the peer ID is unique for a task. This ensures brevity in the number of IDs that the coordinator has to store in memory.

Once the Executor has discovered the corresponding Submitter, it does not communicate through the coordinator. Instead, a P2P connection is established between the Submitter and the Executor directly. The Submitter generates a unique URL for its task, and the Executor establishes this connection by accessing the URL in a browser. The task and the its inputs are sent through this connection.

The decoupling of the coordinator from the task of channeling input to executors ensures that the coordinator is no longer the bottleneck. It inadvertently implies that the connection established between the submitter and executor remains intact in spite of failures at the coordinator's end. Such bottlenecks and single point of failure problems, which are prevalent in most existing volunteer computing architectures, are avoided here.

4.2 Task Submitter and Executor

The Submitter creates a task to be distributed along with a set of inputs on which the task/process is to be run, and then submits it to the network. The task comprises of a JavaScript method which conforms to a specified format of input parameters and return values.

Once the connection between the Submitter and Executor is established as stated above, the Submitter provides inputs to the Executors in a batch round robin fashion. It sends a configurable batch of inputs at once to each Executor. The task of dividing an input into independent units is delegated to the user. This decision is made as user of the system is the only actor who can decide the best execution plan for a task.

The Executors complete their tasks independently by executing a procedure defined by the Submitter on their assigned inputs. The output of the Execution, however, is not sent directly to the Submitter. This is to avoid two types of malicious node behaviour:

- Withdrawing payments - The Submitter can choose not to proceed with paying the Executor even after it receives valid outputs.
- Forgoing legitimate computation - The Executor can avoid computing outputs using the given task and instead send garbage output to the Submitter in hopes of getting paid.

The output computed by the Executor is sent to the blockchain to prevent the above behaviour. In addition to sending the task method and set of inputs to the Executor, the Submitter also sends a set of 'pre-computed' outputs to the smart contracts. The following sections explains the role of blockchain and smart contracts in detail.

4.3 Blockchain Incentivization Mechanism

The blockchain ensures the following:

- Payment for the volunteers for successful completion of work
- Honest behaviour of submitters and volunteers

Smart contracts are used to realize the same.

The Submitter, when it creates a task, is mandated to compute the outputs for a randomly chosen subset of the inputs. This subset is very small compared to m, the size of the inputs. These outputs are computed on a per-batch basis, as in, one set of outputs per batch. The batch size determines the degree of legitimate values the Submitter gets. It then invokes the smart contract, providing it with the vector of hashes of the computed outputs as a parameter, along with the price it is ready to pay. The Submitter also makes the payment to the smart contract. These hashes are recorded on the blockchain.

The Executor, on successfully completing the computation, provides the vector of hashes of outputs to a smart contract method. The smart contract then

verifies if the expected outputs (computed values from the Submitter) are present in the outputs given by the Executor. If so, the smart contract then releases payments to the executor and the executor then sends over the outputs to the Submitter directly via its connection.

Since the Executor has no way to know which input of the batch given to it has its output pre-computed, it has to execute all of it to get paid, thereby ensuring that the executor will not submit garbage outputs for verification and falsely be rewarded.

The batch size directly determines the probability of the Submitter receiving a valid output. If the batch size is small, there the number of inputs per batch decreases, and the Submitter would have to spend more of its computing power to pre-compute the outputs at the benefit of receiving more valid outputs. Conversely, if the batch size is large, then the Submitter would have to perform lesser computation at the risk of receiving wrong outputs from a malicious Executor.

S. No.	**Type**	**Hardware**	**%age Batches**	**Browser**	**Time**
1	Serial	Intel Core i7-4510U CPU	100%	Mozilla Firefox 64.0	198.64 seconds
2	Serial	Intel Core i7-4510U CPU	100%	Google Chrome 71.0	88.53 seconds
3	Distributed	Intel Core i7-4510U CPU	63.63%	Google Chrome 71.0	56.95 seconds
		Intel Core i5-6200U CPU	36.36%	Google Chrome 71.0	
4	Distributed	Intel Core i7-4510U CPU	54.54%	Google Chrome 71.0	52.06 seconds
		Intel Core i5-6200U CPU	36.36%	Google Chrome 71.0	
		Qualcomm Snapdragon 650 hexa-core (4x1.4GHz + 4x1.8GHz)	9.09%	Firefox (Android 6.0)	

Table 1. Run-times with different browsers

5 Flow of Control

1. The submitter joins the network by connecting to the coordinator (P2P Bootstrap).
2. The submitter writes the task to be executed in the given format and also prepares the set of inputs (and also divides them into assignable batches) on which this is supposed to be run.
3. The submitter then executes the task on a subset of these batches and obtains output.
4. The submitter then decides the reward he is willing to pay, and invokes the smart-contract to record the pre-computed outputs (a hash of the pre-computed outputs, for practical purposes; one hash is computed for one

batch) along with the task ID and reward, and a payment is made to the contract. The task 'function' is then given to the coordinator.
5. The executor also joins the network by connecting to the coordinator (P2P Bootstrap).
6. The executor then picks a task from the list of tasks displayed to it by the coordinator.
7. Once the executor picks a task, its connection with the coordinator is terminated. A connection is initiated directly with the corresponding submitter.
8. The submitter then provides the executor with a few batches of inputs corresponding to that task that are still not computed.
9. The executor runs the task function on these inputs and computes the set of outputs.
10. The executor then provides the hashes of the computed batches to the same smart-contract.
11. The smart-contract compares the hashes that have been provided by the submitter (pre-computed outputs) with the hashes given to it by the executor. If all the pre-computed outputs' (in the range of batches given to that executor) hashes are present in the list of hashes given by the executor, the smart-contract then pays the executor with the reward amount as specified by the submitter.
12. If the executor is still connected to the submitter, a new set of batches are given to the executor and the above steps repeat.
13. If the hashes do not match in step 11, then no payment is made as the outputs computed (some or all) by the executor are wrong.

6 Results

To test our system, we executed a simple scenario where the submitter wants to solve the integer factorisation problem. Factorisation does not have a known polynomial solution and is thus an NP-Problem. The problem of finding all factors of n can be distributed among executors by dividing the range of numbers having possible factors (1 to $\sqrt{n}$) into sub ranges. The executors can then launch the factor searching process in their own sub-ranges. For the purpose of demonstrating preliminary results, we make an attempt at factorising a 21 digit (in base 10) number with 67 bits. The submitter function is a simple 8 line JavaScript code which is run in a distributed fashion by all executors in parallel. We have divided our input into sub-ranges as follows:

```
{
  "Input": [
    {
      "start": 1,
      "end": 100000000,
      "num": 123456789123456800000
    },
    {
```

```
      "start": 100000001,
      "end": 200000000,
      "num": 123456789123456800000
    }, // and so on
}
```

Our implementation also provides sample scripts to generate input batches in the above format. The following is the code for the 'task' given to the executors:

```
factor = [];
for(var k = input.start; k <= input.end
    && k * k <= input.num; k++) {
        if(obj.num % k == 0) {
                factor.push(k);
                factor.push(obj.num / k);
        }
}
return {Factor: factor};
```

We record several observations in table 1.

Clearly, distributing the work over several computers reduced the computation time owing to parallel execution. The time required to finish computation decreases inversely with the number of devices. An interesting observation is that the Chrome's V8 engine runs the given JavaScript code faster than the Mozilla's Gecko Engine. We also used a mobile device running the Firefox browser. The Android 6.0 mobile device obviously took more time than a laptop computer, but it was nevertheless able to partake in the voluntary computing platform and yield results. This shows that even mobile devices can participate in the platform.

We also took up the hashing problem to demonstrate our system. The problem is similar to cryptocurrency mining where the executors compute hashes for nonces in the given range. The hash function considered is a Javascript implementation of the String.hashCode() function in Java. The results are shown in Figure 2. Two Gigahashes were computed in total in total by the system. The nonces required to generate two Gigahashes were divided into 100 subranges. 10 of these subranges were grouped together into a batch. Thus there were a total of 10 batches.

Figure 3 shows how the hash rate of the system varies with respect to the number of executors. As can be seen from the chart, the increase in performance is not always linear with respect to the number of executors. There are regions in the graph where the hash rate remains constant even when the number of executors increases. For instance, the hash rate does not increase when the number of executors increases from 5 to 9. The reason for this behavior is explained by the distribution of the batches given in Table 2. A total of 10 batches is distributed between all the available executors. The runtime of execution is measured as the time elapsed between the first executor joining the system and the last executor finishing the computation. As a result, the hash rate is limited by the last executor to finish. Thus, it is dependent on the the maximum number of batches given

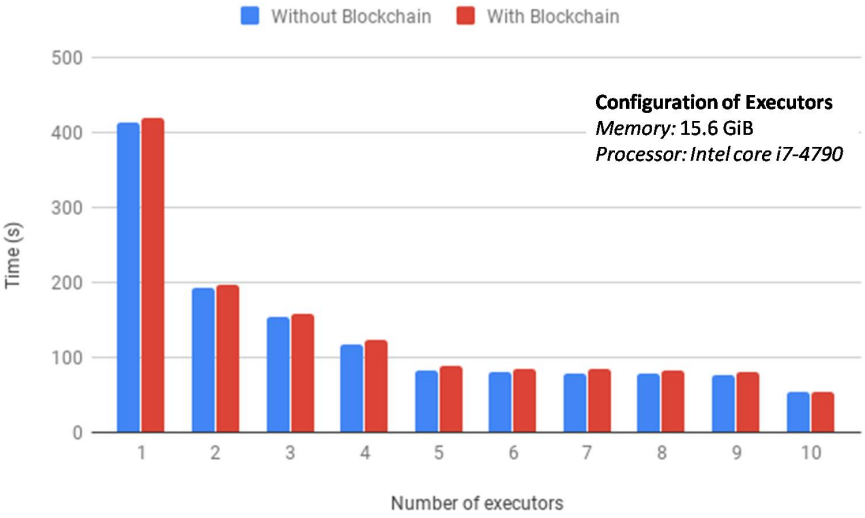

Fig. 2. Running Times for the hashing problem

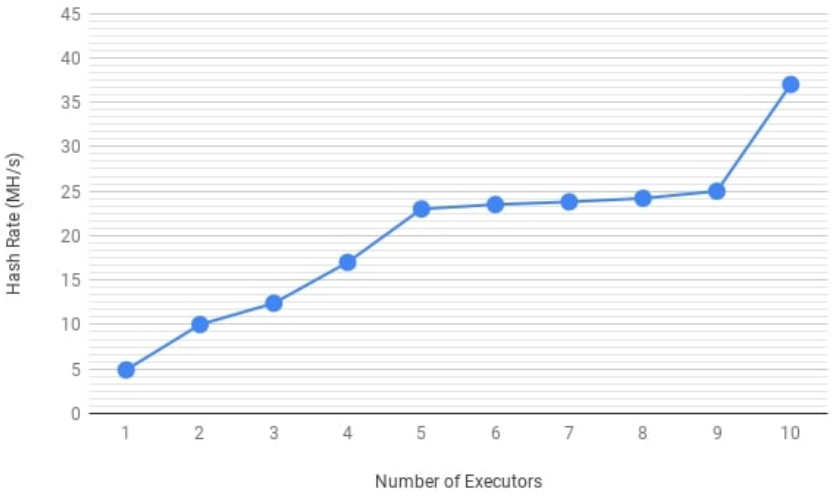

Fig. 3. Increase in Hash rate with number of executors

Executors	Distribution of batches (number of executors X batches)	Maximum Batches
1	1 X 10	10
2	2 X 5	5
3	(1 X 4) + (2 X 3)	4
4	(2 X 3) + (2 X 2)	3
5	(5 X 2)	2
6	(4 X 2) + (2 X 1)	2
7	(3 X 2) + (4 X 1)	2
8	(2 X 2) + (6 X 1)	2
9	(1 X 2) + (8 X 1)	2
10	(10 X 1)	1

Table 2. Distribution of Batches among executors

to an executor. As shown in Table 2, the maximum number of batches given to an executor remains 2 even when the number of executors increases from 5 to 9. However, there is a slight increase in the hash rate when executors increase from 5 to 9. This is because, only the fastest executors finish fast enough to fetch another batch from the submitter. Thus in case of 5 executors, the fastest 4 executors fetch the next batch. The runtime is limited by the slowest of the fastest four. In case of 9 executors, only the fastest executor fetches the next batch. Thus, it completes faster.

7 Analysis

Design Goal	**Technology Used**
Trustlessness	Blockchain
Easy Deployment	Browser
Decentralisation	PeerJS
Platform independence	Browser
Privacy	Ethereum / Metamask
Performance	Architecture
Scalability	Architecture

Table 3. Design Goals

In this section we explore how the design goals proposed in Section III were met. Table 3 summarizes the technologies used to meet these design goals.

- *Easy Deployment*: The system is easy to deploy. The user will need nothing more than a browser to contribute or request computing power. In order to get incentivized, setting up an Ethereum wallet is required. An Ethereum wallet can be set up in a very short time with the help of tools such as the Metamask wallet browser extension.
- *Decentralisation*: Our design aims at eliminating any form of centralization. The purpose of the volunteer computing system is to impede dependency on a central entity for both computation and incentivization purposes. Our

current design uses a coordinator that acts a seed node in the Peer to Peer network, which is inevitable. After establishment of a connection between the submitter node and executor nodes, the coordinator is no longer involved and a direct P2P connection is established between the nodes.

- *Platform independence*: Our design and implementation is independent of the platform as it runs within the execution environment of a browser. It is only limited by the availability of a web-browser on a device. Any device that can run a browser can contribute computational power including mobile phones, tablets and Desktops.
- *Trustlessness*: Existing Volunteer computing systems are all dependent on trust. They generally do not verify the outputs generated by volunteers. There is little reason for volunteers to submit erroneous values in an un-incentivized system. The volunteers also trust the task owner's authenticity. However, in an incentivized system on the public network, such a trust based model cannot be used. A node may simply submit erroneous values to get the incentive rewards. A simple verification is not sufficient as the task's owner may refuse to compensate the volunteer in spite of receiving valid/correct outputs. In such cases, reliance on a trusted third party who can mediate between the task submitter and the task executor seems necessary. However, our design eliminates the existence of any trusted entity. The execution and incentivization takes place without the need for the involvement of trust using blockchain.
- *Privacy*: The system does not require personal data. Ethereum accounts associated with a person do not have personal data attached to them, which means our system preserves user's privacy.
- *Elimination of software and hardware overhead*: Our implementation does not require additional software or hardware set up. There is no need to set up an Ethereum client or an Ethereum node. A browser and a simple browser based Ethereum wallet would be sufficient to participate in the system.
- *Performance*: The performance of the system increases with the number of executors as shown by the results.
- *Scalability*: The system is scalable with respect to both the number of unique tasks it can handle and the number of nodes. The system has no bottleneck as the connection to the coordinator persists only before a direct connection is established between the task submitter and the task executor.
- *Minimum user prerequisites*: There are no prerequisites for a task executor to contribute computational power. A task submitter can write the task in plain JavaScript. The system does not mandate any special syntax and thus the task submitter need not be familiar with the system to use it. This is generally not the case with most other volunteer computing platforms.

Note that the submitter is required to stay on-line even after delegating the process function and the inputs to the executors in order to obtain computed outputs from the executor. This could be a possible future work direction to remove this constraint.

8 Conclusion

In this paper we have proposed a novel approach of leveraging blockchain technology to build a trustless volunteer computing platform. The platform is completely browser based making it convenient for both the task submitters and executors. This removes the requirement of any additional software or hardware as is necessary in traditional volunteer computing solutions. A proof of concept of the proposal has also been implemented and tested using the open source blockchain, Ethereum. The implementation has been open sourced [2]. The platform also supports incentivization as opposed to traditional solutions because of the trustless and immutable nature of blockchain.

References

1. BOID. https://www.boid.com/
2. CollabChain. https://github.com/SagarB-97/CollabChain
3. Einstein@Home. https://einsteinathome.org/
4. Folding@Home. https://foldingathome.org/
5. The Golem Project. https://golem.network/doc/Golemwhitepaper.pdf (November 2016, Accessed: 2019-01-18)
6. BOINC. https://boinc.berkeley.edu/ (Accessed: 2019-01-18)
7. Anderson, D.P.: Boinc: A system for public-resource computing and storage. In: Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on. pp. 4–10. IEEE (2004)
8. Anderson, D.P., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: Seti@ home: an experiment in public-resource computing. Communications of the ACM 45(11), 56–61 (2002)
9. Anderson, D., Fedak, G.: The computational and storage potential of volunteer computing. pp. 73– 80 (06 2006)
10. Merelo-Guervós, J.J., García-Sánchez, P.: Designing and modeling a browser-based distributed evolutionary computation system. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1117–1124. ACM, New York, NY, USA (2015)
11. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
12. Ong, T.M., Lim, T.M., Lee, B.S., Yeo, C.K.: Unicorn: Voluntary computing over internet. SIGOPS Oper. Syst. Rev. 36(2), 36–51 (Apr 2002)
13. Sarmenta, L.F., Hirano, S.: Bayanihan: Building and studying web-based volunteer computing systems using java. Future Generation Computer Systems 15(5-6), 675–686 (1999)
14. Turek, W., Nawarecki, E., Dobrowolski, G., Krupa, T., Majewski, P.: Web pages content analysis using browser-based volunteer computing. Computer Science 14(2)), 215–230 (2013)
15. Wikipedia contributors: Volunteer computing — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Volunteer_computing&oldid=859975321 (2018, Accessed: 2019-01-18)
16. Zorrilla, M., Martin, A., Tamayo, I., Aginako, N., Olaizola, I.G.: Web browser-based social distributed computing platform applied to image analysis. In: Cloud and Green Computing (CGC), 2013 Third International Conference on. pp. 389–396. IEEE (2013)