Benchmarking the Internet Computer

Douglas Bouchet

2023-02-13

Contents

1 Abstract

This paper presents a study on the use of blockchain technology as a platform for performing heavy computational tasks such as machine learning. Through a series of tests, the paper attempts to identify the capabilities of blockchain for such applications. The results of the research indicate that the size of transactions is limited to ten thousand values and that there is a trade-off between redundancy and pace of execution. Furthermore, the length of the model has an influence on the pace of the model. These results provide insights into the potential of blockchain for heavy computational tasks and can be used to inform future research and development in this area.

2 Introduction

The emergence of blockchain technology and its associated smart contracts has revolutionized the way we think about data storage and computing. As a (student) researcher interested in the potential of blockchain technologies, I sought to explore the ability of blockchain to provide a transparent and secure environment for performing heavy computational tasks, such as machine learning. This paper aims to investigate if blockchain can indeed be used for such purposes. In order to see if the blockchain can then be used to help with heavy computational tasks, we will take an already existing protocol to perform this task, the Federated Learning. We will then see how it is possible to modify it by adding the blockchain. We will detail the smart contracts used, as well as the interactions between the different agents involved in the machine learning task. Finally we will test our new framework using diablo, a program that allows to submit transactions to a blockchain, and to measure the performance of the latter, in response to these transactions. For more realism, the nodes constituting our blockchain will be emulated using machines made available by the AWS service

DO we add these ? ——- The expectations of some fundation such as Dfinity project Can be questionable, as performaces of some BC can be quite limited

---

3 Method

3.1 The need for computational resources in Machine Learning

Recently, many machine learning models have been published. These models have become very popular mainly due to their performance. We can quote models such as DALL-E, or GPT-3. The training of these models requires a huge amount of data and computing power. If we wanted to train GPT-3 using one of the most powerful GPUs currently available, the NVIDA v100, the time required would be about 300 years (also making the optimistic assumption that all the training data could fit in the GPU RAM). There is therefore a need for techniques to increase computing power in order to generate these huge models.

3.2 The three main goals of scaling up the computational power

When designing a system able to provide such an increase of power, the following three aspects should be taken care of by this system, as they are the ones that make it currently feasible.

1. **Provide reasonable performances**. Indeed, the purpose of such a system is to provide an increase in computing capacity. The availability of sufficient computing power is therefore one of the main factors.

2. **Reward those that participates in the system**. What would be the interest for the participants to work for free, or without guarantee of payment?

3. **Shouldn't be specific to machine learning**. This system should be able to adapt to any type of task, so as not to be too specific, and potentially limited in its evolution or adaptation.

3.3 Federated Learning, an existing system to provide computational resources

Federated Learning is a system that allows to involve multiple devices in the training of a model. The idea is to distribute the training data to the different devices, and to train the model on each of them. The model is then aggregated to obtain a single model. This system is therefore able to provide a significant increase in computing power. In this project, we will consider a simplified version of federated learning. We assume that each participants already has the data needed to train the model. Our version consists of the following steps:

1. The model is initialized on the server
2. The model is sent to each participant
3. Each participant compute one model update
4. Each participant send the updated model to the server
5. The server update the model using some aggregation rules
6. Repeat step 2 to 5 for a given number of iterations, or until some convergence criteria is met

This system has some limitations, such that the rewarding of the participants is not guaranteed, or the possibility for new participants to join the process.

3.4 Advantages of using a blockchain to enhance this protocol

Using a blockchain to coordinate workers brings several advantages, which might help to solve the three main goals of scaling up the computational power.

1. Goal 1 "provide reasonable performances": Using a blockchain allow in theory anyone to join the learning process. This can provide an unbounded increase in computing power. The only limitations would be the number of workers involved in the learning, their power, but also the maximum number of participant the blockchain could handle.

2. Goal 2 "reward those that participates in the system": Using a blockchain provide an easy way to reward the participants. In order to to so, we could use smart contract, paying in gas participants that did a correct job - i.e learn a correct model. This has the advantage that anyone could actually see the code executed when rewarding the participants, as the smart contract code can be read from the blockchain. This is a very important point, as it can bring confidence to the participants, increasing the trust in the system, and potentialy motivate participants to join the process.

3. Goal 3 "shouldn't be specific to machine learning": Redundancy protocol in blockchain is a measure taken to ensure the continuity of the blockchain network by replicating the data across multiple nodes. This is done to ensure that if one node fails, the data on the other nodes can be used to prevent any disruption to the network. This also guarantee that data cannot be tampered. This implies that using a blockchain in combination with federated learning isn't only reserved to machine learning problem, but any other type of problems, involving heavy computational tasks.

slides which answer the 3 goals

3.5 Including BC in federated learning, a naïve approach

Now let's see how we can add a blockchain to our current federated learning protocol. The base case is the same as in the federated learning scenario, but now we can add a smart contract between the server and the participants. The smart contract is published by the server on the blockchain, and it will coordinate the participants. Our new protocol in order to do one model update is the following:

1. Server sends the model weights to the smart contract

2. The smart contract sends the same model weights to a pool of participants choosen among all participants. The number of participants in the pool (the ones that will perform the same model update) is called the **redundancy**

3. By assumption, each participant has the training data, so each one can perform one model update

4. Each participant send the updated model weights, along their public key to the smart contracts. The public key is sent as the smart contract need to associate each model submission to a worker in order to potentially pay them later.

5. The smart contract then set the elected model as the model with the most bids from participants.

6. The elected model is then sent to the server, which can update its current model. This makes one epoch

7. The steps 1 to 6 are then repeated for a chosen number, or until some convergence on loss is reached.

This explanation was focus on only one pool/group of participants, but in a real case there would be several pools running in parallel, allowing for a quicker model training, which is what this protocol is aiming at.
There is however a big problem with this protocol. As the smart contract is located on a blockchain, everything that is actually written to the smart contract, can be read by anyone (assuming we are using a public blockchain). This means that it is actually possible for a participant to not actually perform the model update, but still get paid. Let's illustrate this with a simple example. Suppose we have a worker(W_0) that sends its updated model weights $M_{W_0}$to the smart contract. The transaction would then be (M_{W_0}, Pk_0). Some evil worker (W_1) can read this transaction, and steal the model weights M_{W_0}. It can then send the following transaction (M_{W_0}, Pk_1). The smart contract has no way to know if the model weights M_{W_0} were actually learned by W_0 and W_1. If this model is the elected one, then both W_0, W_1 will be paid. Indeed it's actually possible for someone to get paid, without performing the computation. Let's now see how we can solve this issue.

3.6 Solving the model stealing issue

In order to solve the model stealing issue, we will use a **commit-reveal** protocol. It involves two steps: a sender first commits to a message by providing a cryptographic hash of it, without revealing the actual message. Then, the sender reveals the message to prove they are the one who committed to it.
Let H be a one-way cryptographic hash function. This involve that it is easy to compute $H(x)$, but it is hard to compute x from $H(x)$.
Our commit-reveal schema will be the following:

- Commit: the worker W_i sends $H(M_{W_i}|Pk_i)$ to the smart contract
- Reveal: the worker W_i sends M_{W_i}, Pk_i to the smart contract

After the reveal, the smart contract can check that $H(M_{W_i}|Pk_i)$ is equal to the commit. If it is the case, then the submission is considered valid. Otherwise its rejected. The elected model is computed on valid submission only.
One important thing to add: during the reveal phase, no more commits are accepted as during the reveal we send the model in clear. If new commits were still accepted, it would be possible for someone to read some of the submitted models, and create a commit with them. The commit would indeed be valid, therefore our protocol wouldn't prevent anymore from model stealing.

3.7 Dealing with smart contract limited memory

- soem numbers for max var size - idea: split the models In order to deal with these limitations, we will not send the model weights in one transaction during the reveal phase, but rather send chunks of it. Now suppose that we can write our model as following:

$$M = M_1|M_2|...|M_n \tag{1}$$

We modify the reveal as following:

1. worker i sends $H(M_{W_{i,1}})$, which is the hash of the first chunk of the model
2. he can then send the hash corresponding to the next chunk: $H(M_{W_{i,2}})$ and so on.

The smart contract create only one variable for the chunk received (for each worker). This variable contains the hash of the first chunk, which is computed by the smart contract. The model chunk can be stored on an external storage, to not use memory in the smart contract. Upon receiving the next chunk, it update the previous hash by hashing it with the new chunk. The new chunk is also added to the external storage. This operation is repeated for every new chunk received from this worker. The last chunk received is also concatenated with the public key of the worker.
At the end, for each worker we have the complete model in an external storage, and the following hash on the smart contract

$$\begin{aligned} Hash_{W_i} &= H(...(H(H(M_{W_{i,1}})|M_{W_{i,2}})|...)|M_{W_{i,n}}|Pk_i) \\ &= H(M_{W_{i,1}}|M_{W_{i,2}}|...|M_{W_{i,n}}|Pk_i) \\ &= H(M_{W_i}|Pk_i) \end{aligned} \tag{2}$$

The second line is obtained using the Merkle-Damgard theorem. We see that using this construction $Hash_{W_i}$ during reveal protocol can be compared to the one send during the commit protocol.
We don't need to modify the commit, as we only send an hash, which is of small constant size. Using this construction, we can therefore reduce the memory used on the smart contract. In fact, we are only storing hash on the smart contract, which are 32 bytes long. - total expected memory usage on the smart contract

4 Learning a model using blockchain result and observations

- small intro about aws + diablo

4.1 First limitation, txs size

- explain protocol for testing - add graph - add explanation

4.2 The Redundancy/Pace tradeoff

- term definition - add graph - add explanation

4.3 Minimum pace depending on model length

- Do this, as it could provide some idea about time needed to train a model - Fix redundancy, why 8 - add graph - add explanation

4.4 Expected training time for some classical models

- add tab - add analysis

5 Conclusion

- extend what was said in the abstract - opening to small ml model + transfert learning

6 References

- any ref to number used in the report - link to github repo