BCTMark: a framework for benchmarking blockchain technologies

Dimitri Saingre, Thomas Ledoux, Jean-Marc Menaud

BCTMark: a framework for benchmarking blockchain technologies

Dimitri Saingre
IMT Atlantique - Inria - LS2N
Nantes, France
dimitri.saingre@imt-atlantique.fr

Thomas Ledoux
IMT Atlantique - Inria - LS2N
Nantes, France
thomas.ledoux@imt-atlantique.fr

Jean-Marc Menaud
IMT Atlantique - Inria - LS2N
Nantes, France
jean-marc.menaud@imt-atlantique.fr

***Abstract*—Over the last years, research activities on blockchain technologies have fairly increased. Firstly introduced with Bitcoin, some projects have since emerged to create or improve blockchain features like privacy while others propose to overcome technical limitations such as scalability and energy consumption. New proposals are often evaluated with ad hoc tools and experimental environments. Reproducibility and comparison of these new contributions with the state of the art of the blockchain technologies are therefore complicated. To the best of our knowledge, only a few tools partially address the design of a generic benchmarking of blockchain technologies (e.g., load generation). In this paper, we introduce BCTMark, a generic framework for benchmarking blockchain technologies on an emulated network in a reproducible way. To illustrate the portability of experiments using BCTMark, we have conducted some experiments on two different testbeds: a cluster of Dell PowerEdge R630 servers (Grid'5000) and one of Raspberry Pi 3+. Experiments have been conducted on three different blockchain systems (Ethereum Clique/Ethash and Hyperledger Fabric) to measure their CPU consumption and energy footprint for different numbers of clients.**

***Index Terms*—Blockchain, Performance, Evaluation, Benchmarks, Reproducibility.**

I. Introduction

Since their introduction in 2008 with Bitcoin [1], blockchain technologies have been widely developed. First used for crypto-currencies, blockchains are now being implemented in many cases: sharing of computing resources [2], decentralized social networks [3], government services [4], storage solutions [5], [6], energy trading [7], [8], ...

Despite the potential of blockchain technologies in many areas, technical limitations slow their development as a possible alternative to centralized services. For example, several issues dealing with their scalability [9], [10] or energy cost [11], [12] have been identified.

Several improvement proposals have been recently made to face those issues. We can cite the examples of the reparameterization[1] proposals in the Bitcoin community (see BIP[2] 100 to 107), new consensus systems such as [13], [14] or the introduction of off-chain transactions systems like [15].

Those proposals have been mostly evaluated through debates (e.g., in the case of the BIPs) or have used ad hoc evaluations that are often not reproducible (i.e., cannot be run on systems other than the one they have been designed for). We argue that, to properly compare the performances of several blockchain systems and quantify the contribution of new proposals regarding performance issues or functionality (e.g., fault tolerance), the blockchain community needs proper tooling for reproducible experiments.

This paper presents a framework enabling reproducible research on the performances (latency, throughput, energy consumption, ...) of blockchain technologies. BCTMark (**B**lock**C**hain **T**echnologies Bench**mark**ing) is intended to be a framework which can be used to deploy, compare, and evaluate (through various scenarios) any blockchain on a large number of different infrastructures. This framework provides an abstraction of the underlying physical infrastructure and can, therefore, be used to deploy easily on any platform that supports the SSH protocol. To demonstrate this flexibility, we have deployed experiments on both a public research cluster (Grid'5000 [16]) with "classical servers" (Dell PowerEdge R630 servers) and a private "low-power" Raspberry-Pi cluster.

The author of [17] defines several criteria to define a "good" benchmark. We argue that BCTMark has features that cover each criteria defined in [17]:

- **Repeatable**: BCTMark can manage the whole lifecycle of experiments (resources reservation, deployment, load generation, metrics collection,...) and can be used to deploy the same experiment on different infrastructures. Experiments results are consistent across different run (see subsection IV-C).
- **Observable**: BCTMark embeds several components to observe both performances and impact of the system under test (CPU consumption, disk and memory usage, ...)
- **Portable**: BCTMark can be used to compare different blockchain systems or different versions of the same blockchains. Users can write a driver to use this solution to compare their new system to existing ones.
- **Easily presented**: BCTMark embeds a Grafana dashboard [18] that can be used to present the results. Metrics are also stored in a time-series database.
- **Realistic**: Network capacities (bandwidth, latency, packet loss, ...) can be described in the deployment topology to emulate real-world deployment.
- **Runnable**: BCTMark can manage the whole lifecycle

[1]Evolution of parameters like block size and emission rate
[2]Bitcoin Improvement Proposal

of the experiment (from the resources reservations on a given testbed to the metrics collection of the system under test). It makes them easier to run: the same configuration deployment can be shared with other scientists, even on different testbeds. The deployment topology itself (number of peers, network partition and capacities, ...) can be easily described in YAML, a language commonly used for configuration.

To the best of our knowledge, only a few tools address the issue of blockchains benchmarking (see section V). These existing frameworks, while promising, do not manage aspects necessary for rigorous benchmarking like environment deployment (improving reproducibility), collection of resources usage (e.g., CPU and memory consumption) and network emulation (crucial as, for blockchains, network issues have an impact on the diffusion of new blocks).

To sum up, our contribution results in the design and the development of a framework that can be used to create experiments on performance and functionality evaluation of blockchains systems. Thanks to the design and features provided by BCTMark, these experiments can be repeated in different environments (and therefore can be shared with the scientific community for peer evaluation) and can be run in a realistic environment thanks to network emulation functionalities.

The rest of the paper is organized as follows. After a brief reminder of general concepts on the blockchain, we detail the architecture of BCTMark as well as its operation (from a user point of view). Then, we document some first experiences to illustrate the system's capabilities. Finally, after a discussion on related work, we detail the next steps in the development of BCTMark before concluding.

II. Background on blockchain technologies

A. Overview

A blockchain can be seen as a distributed data structure that allows facts (called *transactions*) to be recorded as blocks[3]. Each block has a link to the previous one (making a "chain of block," or *blockchain*). This data structure is distributed among all participants in a peer-to-peer network. This network is maintained by some peers called *miners* (Bitcoin) or *validators* (Ethereum). Those are in charge of transaction validations.

Validating transactions involves a securing process that can be seen as a *leader election*. The mechanism involved depends on the blockchain system. Probably, the most famous one is called *proof-of-work* (PoW). It involves a "cryptographic puzzle". Every block here contains a value called a *nonce*. To validate a block, a miner has to find a value for the nonce, such as the hash value of the whole block is under a certain threshold (called the *difficulty*). This threshold value varies so that, even as the hardware becomes more powerful, the throughput of the entire network remains at about one block per 10 minutes.

[3]A batch of transactions

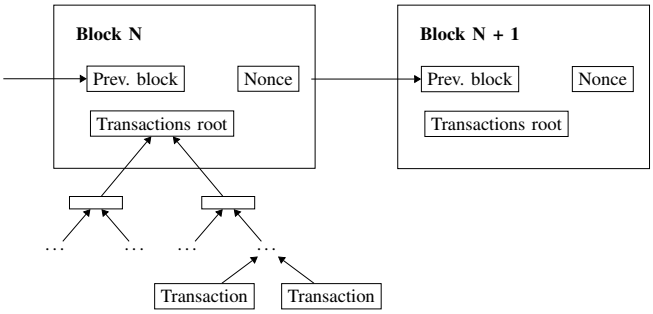

Fig. 1. A schematic view of a classical blockchain "data-structure"

Although the *proof-of-work* has been widely used with Bitcoin, it has been quite criticized for its high energy consumption [12]. Alternatives such as *proof-of-stake* (PoS) [13] or *proof-of-elapsed-time* [19] have emerged. Instead of basing its security on computational power, *proof-of-stake* systems rely on the distribution of wealth. In these systems, the probability of validating a block is proportional to the number of coins one owns (in some cases, coins can have a certain weight to avoid having a network led by the richest).

B. Smart contracts: computations on the chain

Some blockchains, like Ethereum [20], has a concept of *smart contracts*. Smart contracts are scripts written in a high-level programming language that can be deployed and executed through network transactions.

Once written, those smart contracts can be deployed on the network through a transaction containing their compiled code. On Ethereum, transactions without any recipient are used for smart-contract deployment. Once deployed, the smart contract gets an address like any "normal" accounts[4]. The contract can then be called by sending a transaction to its address containing a compiled version of a function called with desired parameters (if any).

Smart contracts offer many computational possibilities and are the backbone of any blockchain-based decentralized applications. As those contracts need to be deterministic (as every peer running the contract needs to produce the same result), they cannot have any side-effects outside the blockchain (e.g., they cannot call any Web services). As today, one of the performance limitations of the peer engines executing transactions (and so smart contracts) is that they execute all the transactions sequentially (and therefore missing the capabilities of multi-core processors). Nonetheless, work (such as [21]) is ongoing in this area.

C. Public vs. Private Blockchains

Blockchain technologies can be divided into two categories: public and private blockchains. Public blockchains, such as Bitcoin and Ethereum (with its Ethash [22] engine), have no identified users. One can join or leave the network at any time without the need for any authorization. Security protocols of

[4]The significant difference between an account controlled by a contract and one by a human is the presence of its code

those public blockchains need to be enforced to face potential Byzantine faults. Proof-of-work is an example of a consensus system for public blockchains.

Private blockchains have different security models. They aim to identify participants, especially for the block validators. These are designated in the protocol so that no one else can validate the block. These blockchain systems, such as Ethereum (with its *Clique* engine) and Hyperledger Sawtooth (with its *Proof of Elapsed Time* system, based on the *Intel SGX* enclave), have different consensus engines. These engines have better performances (due to the different security models considered) but offer a lower degree of decentralization.

We have illustrated in this section the variety of technologies behind the term *blockchain*. According to Google Scholar, the number of publications concerning the term *blockchain* was 9 510 in 2017, 25 700 in 2018, 33 000 in 2019 and 35 000 in 2020 (at the moment where this paper was written). This trend tends to illustrate a gain of interest on blockchain technologies. However, to the best of our knowledge, only a few tools exist to evaluate this growing number of publications. To address this lack, we introduce BCTMark, a framework for benchmarking blockchain technologies.

III. BCTMark

This section presents BCTMark, our solution for benchmarking blockchain technologies. We first introduce how BCTMark can be used to run existing experiments and how developers/scientists can integrate new blockchain systems to be tested. Then, we detail its architecture and underlying components.

A. Usage

From a user's point of view, the workflow of an experiment performed with BCTMark proceeds as described in Figure 2.

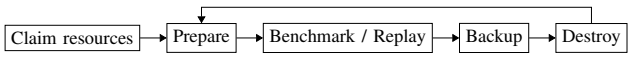

Fig. 2. The experiment workflow of BCTMark

The first step is to *claim* resources on which to deploy the experiment. BCTMark is intended to be portable to manage repeatable experiments. Experiments can be deployed on any infrastructure that supports SSH connections. Some research testbeds (like Grid'5000) require users to book resources before using them. This reservation phase can be addressed by BCTMark. As shown in Listing 1, the deployment topology can be described in a YAML file. This provided example can be used to deploy on a local device 1) an Ethereum network with one bootnode and two peers, 2) one benchmark worker (used to generate loads), 3) a "dashboard" server that hosts both the monitoring stack and the load generator master (that coordinate workers, see subsection III-B for details on load generation). In this case, the *claim* phase will only start the required virtual machines.

Once the infrastructure resources claimed, BCTMark can *prepare* the experiment by deploying the required components (i.e., download and install dependencies, copy configuration files...). For each *role* (see Listing 1), there is corresponding component to be deployed. The monitoring stack (*dashboard* role) and the benchmarking workers (*bench_worker* role) are common to many experiments. Users can define their roles to deploy their blockchain network. In the example in Listing 1, we need two roles to deploy an Ethereum network: bootnodes[5] and peers.

After deployment, users can run the benchmark themselves. BCTMark provides two possibilities to do this: an ad hoc load generation and a one based on previous traces (for more details on the implementation choices, readers may refer to subsection III-B). Once the benchmark has ran, the results can be backed-up, and the environment destroyed/cleaned for another experiment.

```
deployment:
  vagrant:
    backend: virtualbox
    box: generic/debian10
    resources:
      machines:
        - roles: ["dashboard"]
          flavour: tiny
          number: 1
        - roles: ["ethgethclique:bootnode"]
          flavour: tiny
          number: 1
        - roles: ["ethgethclique:peer"]
          flavour: tiny
          number: 2
        - roles: ["bench_worker"]
          flavour: tiny
          number: 1
```

Listing 1. Configuration example for local deployment with Vagrant

From a developer's point of view, all the following necessary actions must be implemented to integrate a new blockchain to be tested:

- *Deployment*: write a new Ansible playbook (cf. subsection III-B) that specify how to deploy, backup and delete the system;
- *Metric collection*: write a Telegraf plugin (cf. subsection III-B) to gather system-specific metrics (e.g., block emission rate) if not already available through HTTP web services (BCTMark can collect metrics exposed at given HTTP endpoint);
- *Adhoc Load generation*: write functions that correspond to an interaction one can have with the system (e.g., how to send a transaction, how to call a smart contract, ...);

[5]Bootnodes are peers that have an address known by everyone in the network. New peers can connect to those bootnodes to get the address of other peers in the network

- *Reproducible Load generation*: implement functions to backup transactions (and serialize those) and functions to replay a given serialized transaction.

A developer/researcher would benefit from the design of BCTMark as a framework to easily integrate its new blockchain technology to be tested. Indeed, BCTMark already provides:

- *Deployment*: portability of deployment on several testbeds that support SSH;
- *Network emulation*: latency, bandwidth limits, ...;
- *Metric collection*: collection of metrics related to the infrastructure (e.g., CPU usage);
- *Load generation*: distribution of the load to generate among workers.

Only specific interactions with the blockchain to be tested need to be implemented.

B. Architecture

To avoid reinventing the wheel, BCTMark is based on the state-of-the-art industry-proven tools. Altogether they empower researchers, allowing them to provision computing resources, deploy blockchain peers, generate load (based on an history to reproduce or according to a given scenario), and collect metrics relating to peers' performance and energy consumption. The architecture of BCTMark is illustrated in Figure 3.

Deployment. BCTMark can deploy the entire experiment stack: system under test, monitoring system, and load generators.

Deployment does not require any agent installation on the machines. They are managed through SSH. A *playbook* defines the configuration to be deployed, which takes the form of configuration files in YAML format. Those configuration files make it possible to specify the desired deployment in a relatively explicit, documented, and repeatable way. BCTMark also provides an abstraction layer of the underlying infrastructure. The deployment topology can be described in a relatively high-level point of view, portable on different testbeds. That makes experiments portable on various infrastructures such as Vagrant (local deployment), Grid'5000 and Chameleon.

To manage deployment, BCTMark uses EnosLib [23] (an open-source library to build experimental frameworks) and Ansible [24] (a software that allows to manage deployment of configuration on a cluster). These two components enable self-describing, reproducible deployments.

Metrics management. Metrics about the server (CPU, memory consumption, HDD usage, ...) and blockchains (number of blocks produced, hashrate, ...) are collected, stored and displayed by Telegraf [25], InfluxDB [26] and Grafana [18] in time-series, respectively.

Telegraf natively allows the collection of server metrics through many plugins written in Go. New ones can be developed to manage the collection of data on deployed blockchain peers. Current experiments on Ethereum deployment use the HTTP plugin from Telegraf to collect metrics through the Ethereum HTTP API.

Network Emulation. One strength of BCTMark is its ability to describe simply the desired network to emulate. Users can describe in the YAML deployment configuration file several groups of peers and emulate any desired network condition between them. The current characteristics of the network that can be emulated are the percentage of packet loss, network delay, and network rate (i.e., bandwidth). A use case of this feature could be to study the effect of a sudden network partitioning or merge on a blockchain system. Under the hood, BCTMark uses EnosLib that applies the desired network rules using the Linux command TC.

Load generation. BCTMark supports two ways to generate workloads[6]: an *ad hoc* load generation (based on Python scripts) and a load generation based on an history. The first one uses Locust [27], a load generator written in Python. The user needs to specify, through Python methods, any interaction a user can have with the system under test (e.g., sending a transaction to someone or deploying/calling a smart contract). Locust will then use those methods to generate random loads.

The second way to generate load is based on a provided history. BCTMark can extract the history of a peer in the system and serialize it in a YAML file containing all the transactions. To reproduce the history, it can split the transactions between different workers, create the number of accounts needed to replay it, and let the workers re-run the transactions. This way, we can aim to replay transactions issued from the *mainnet* of a targeted blockchain system.

Energy consumption. BCTMark does not embed any energy monitoring tools. However, as it enables the deployment of experiments on any kind of testbeds, it can be used to deploy systems on clusters where the energy consumption is monitored. We have already tried this by deploying experiments on the SeDuCe [28] cluster (see subsection IV-A). It is part of the Grid'5000 testbed and is monitored with both energy and thermal sensors.

IV. Experiments

In this section, we illustrate BCTMark's capabilities through three experiments. The first one demonstrates its capacity to deploy experiments on different testbeds, the second one its capacity to compare two blockchain systems and the third one, its usage for smart-contract performance evaluation. Those experiments use two different testbeds (both having power measuring capacities):

1) A Raspberry-pi 3+ cluster. Each node has a quadcore Cortex-A53 ARMv7 CPU and 1GB of RAM.
2) Grid'5000 [16] Ecotype: A Dell PowerEdge R630 cluster. Each node has two Intel Xeon E5-2630L v4 (Broadwell, 1.80GHz, 10 cores/CPU) CPU and 128 GiB of RAM. Grid'5000 is a large scale public research testbed containing several clusters. Ecotype is one of those clusters, located in Nantes (France).

We evaluated three blockchain systems:

[6]By *workload*, we mean *transactions* to be processed by the system under test

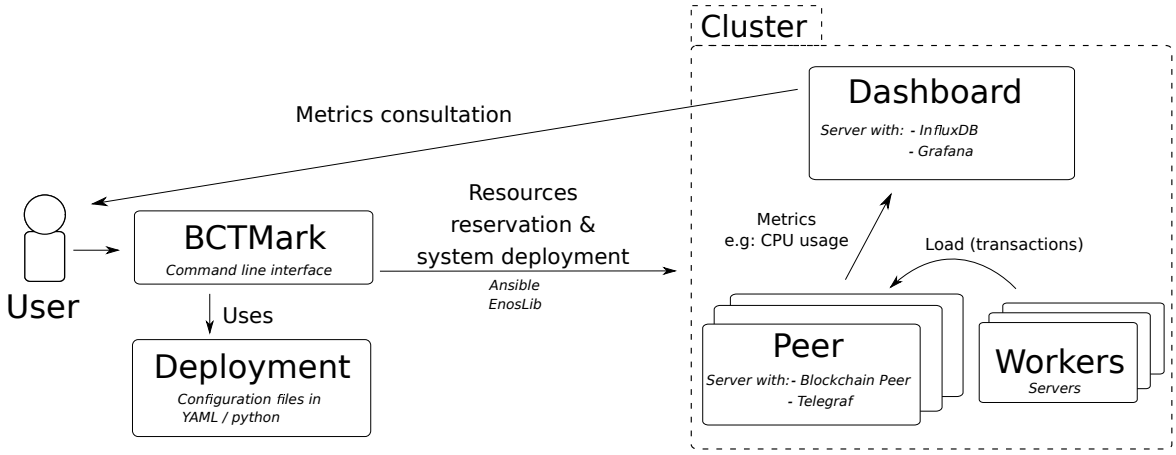

Fig. 3. BCTMark architecture

1) Ethereum Ethash, an implementation of the Proof of Work (PoW) system of Ethereum. It is the default implementation of Ethereum, used in the context of a public blockchain. In this system, every peer can actively participate to block mining.
2) Ethereum Clique, an implementation of the Proof of Authority (PoA) system of Ethereum. PoA is used in the context of a private blockchain. In this system, pre-selected and identified peers can validate blocks one at a time. It does not involve any mining.
3) Hyperledger Fabric. It is also intended for private blockchain. Peers submit transactions to special peers called *orderers*. Orderers are in charge of the ordering process of transactions. Hyperledger Fabric uses a voting-based consensus protocol.

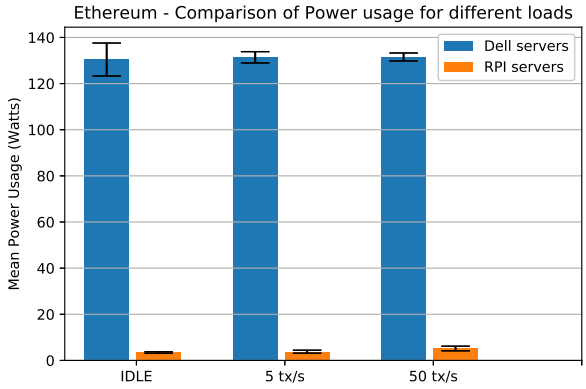

Fig. 4. Comparison of Power Usage for different loads

A. Deployment of blockchains on two different testbeds

This experiment illustrates the capabilities of BCTMark to deploy blockchains on different testbeds. We have deployed Ethereum Clique on both Raspberry Pi and Ecotype cluster under three scenarios. The IDLE scenario does not include any load generation. Peer just generate and share empty blocks. The two other scenarios include a load generation of 5 and 50 transactions per second. Load is generated by separated workers and spread randomly across peers. For both experiments, we deployed 12 peers and 6 load generator workers.

Results are presented in Figure 4. The bar plotted on the graph corresponds to the average power usage of every machines in the cluster. The error bar illustrates the standard deviation of power usage.

Those two platforms have different power draw. Power usage on the Dell servers goes from 130.4 to 131.54 watts (0.7% increase) whereas power usage on the Raspberry Pi platform goes from 3.4 to 5.2 watts (44% increase). This result was expected as Raspberry Pi are much more limited than classical "high performances" Dell servers. This experiment however illustrates that non-mining chains can be installed on low-power platforms like Raspberry Pi. This can be useful in the context of the development of blockchains in IoT / Edge computing. In the context of research on energy consumption, low-power platforms can be useful to illustrate subtle differences in the consumption.

We can, however, note that this conclusion may not be the same for mining systems such as Ethereum Ethash. Indeed, we could not install Ethash on our Raspberry Pi platform due to shortage in memory. The algorithm used by Ethereum Ethash for mining is memory intensive and therefore not suited for low-power platforms with not enough RAM. A solution for this issue could be to set-up both high-performance nodes dedicated to mining and low-power nodes that would only broadcast transactions to the miner's network.

B. Comparison of CPU usage of three blockchain systems

This experiment aims to illustrate the capabilities of BCTMark to deploy different blockchain systems. We deployed Hyperledger Fabric, Ethereum Ethash, and Ethereum Clique on the Ecotype cluster under four scenarios: IDLE (no-load generation) and load generation of 5, 50, and 200 transactions per second. The deployed network is composed of a network of

39 peers and three load generator workers. Figure 5 illustrates this experiment. The bar corresponds to the average CPU usage across all machines, whereas the error bar goes from the 10th quartile to the 90th quartile.

We can first notice that the CPU consumption of the Ethash system exceeds the CPU usage of the two others. Moreover, in this deployment, peers only mine blocks using one thread. It could be possible to dedicate more resources for mining, increasing the CPU consumption furthermore. The other two systems have non-mining consensus systems, decreasing the amount of computation needed to secure the network.

The CPU usage of non-mining systems are also more stable than the Ethash system. Figure 6 illustrates the evolution of the CPU usage for Ethash peers during the "200 transactions per second" scenario[7]. The spike at the beginning of the experiment, reaching almost 100% CPU, is due to the construction of the data structure needed by peers to start mining. We can also see that, after this spike, the CPU usage increases over time. This increase may be due to the evolution of the difficulty in mining resulting from the mining competition between peers.

On the other hand, in the first three scenarios, the CPU consumption of the two private blockchains is roughly the same. However, at 200 transactions per second, the CPU consumption of the Ethereum Clique network increases from 0.3% to 2.9%. This increase suggests that Hyperledger Fabric could have better performances in the context of a private blockchain. These results about private blockchains are consistent with those shown in the Blockbench paper [29].

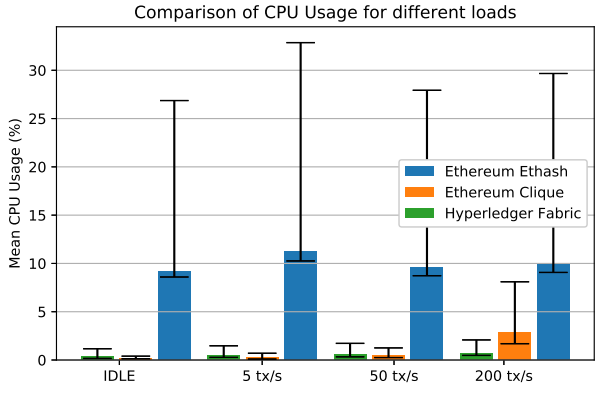

Fig. 5. Comparison of CPU Usage for different loads

C. Experiments Reproducibility

One of the goals behind BCTMark was to enforce reproducibility on blockchain experiments. Reproducibility means that running experiments several times (in similar conditions) should give coherent results. The goal of this section is to illustrate how experiments made with BCTMark can be reproduced.

[7]CPU usage values seem to differ from ones in Figure 5 but this visual effect is due to 1) high variance in data and 2) high density in data points that hides lowest values. We can notice the high variance on Figure 5.

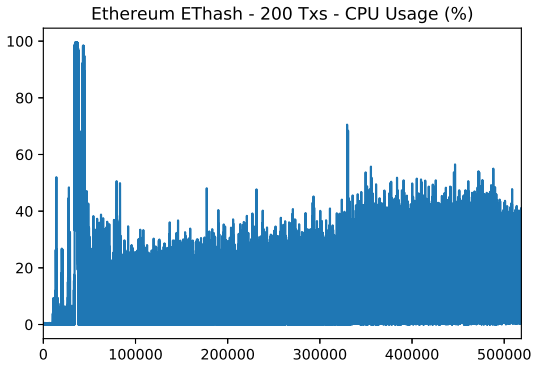

Fig. 6. Evolution of Ethash CPU usage for 200 Txs

Deployment	Min	Max	Mean	Std
Ethereum Clique IDLE	0.113	0.117	0.115	0.002
Ethereum Clique 5 Txs	0.138	0.155	0.145	0.007
Ethereum Clique 50 Txs	0.463	0.526	0.494	0.028
Ethereum Clique 200 Txs	1.682	3.686	2.185	0.843
Ethereum Ethash IDLE	8.751	9.574	9.158	0.304
Ethereum Ethash 5 Txs	9.137	10.169	9.542	0.410
Ethereum Ethash 50 Txs	9.192	11.012	9.945	0.821
Ethereum Ethash 200 Txs	8.934	10.621	9.584	0.630

TABLE I
REPRODUCIBILITY ACROSS SIX RUNS

We reproduced the experiments done in subsection IV-B on Ethereum Clique and Ethash to have data on both public and private blockchain systems (readers can refer to this section to read about the infrastructure used and the deployment topology). Each of the four scenarios has been run six times. For every run, we have recorded the average CPU usage across all machines. The data presented in Table I illustrate the differences in the results we obtained. For instance, the min column illustrates the min CPU average across the six runs. Experiments should show consistent results to be considered reproducible.

These results show that we obtained few differences between the six runs. The standard deviation (column 'Std') remains low across all scenarios. This small difference in results leads us to believe that experiments with BCTMark should produce consistent results. Having exactly the same deployment topology with the same configuration is, in our opinion, the main factor explaining these consistent results. BCTMark allows researchers to share experiments that can be run in the same way by other peers in their community.

D. Performance analysis of Smart contracts

The experiment, presented in Figure 7, is intended to illustrate the capabilities of BCTMark for performance analysis of software developed for blockchains. As explained in subsection II-B, blockchains like Ethereum enable developers to write applications through smart contracts. On Ethereum, each call to a smart contract requires a "fee" related to its cost in gas. Gas is a unit related to the computational cost of each instruction in a contract. The more computation there is in a

contract, the more expensive for end-users it will be. Moreover, in each mined block, there can be a limit to the sum of each transaction's cost in gas. As a result, there is an incentive for smart contract developers to control their contract's cost in gas.

To illustrate how implementation design and details can impact the cost of a smart contract, we implemented three classical sorting algorithms: Quicksort, Bubblesort, and Mergesort. We then have deployed those contracts on a four nodes Ethereum network and generated calls to those contracts. We have measured the cost in gas for each call. For each algorithm, we have generated calls to its sorting function with a random array of integers.

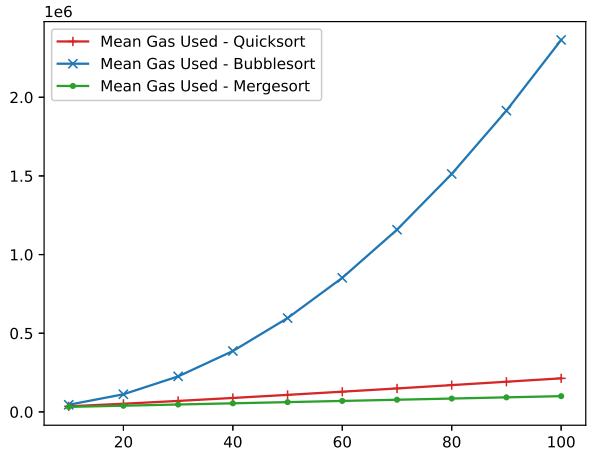

Fig. 7. Cost in gas of three smart contracts depending on provided input

Figure 7 illustrates the evolution of gas consumption depending on the size of the input array. We can see that the evolution of gas requirements are coherent with the complexity of those three algorithms. Quicksort and Mergesort have the same average complexity of $O(nlog(n))$, whereas Bubblesort has an average complexity of $O(n^2)$. This statement is reassuring as it tends to show that the EVM[8] has been correctly implemented. This simple example demonstrates that BCTMark can be used to study smart contracts' costs and that its implementation has an impact on smart contracts possibilities. For the same iso-functionality (here, sorting), the cost of calling the contract will differ depending on the underlying algorithm. For information, at the time this article was written, the Mergesort algorithm would have cost around $0.55 to sort 100 items (cost of $\sim$101659 gas). In contrast, the Bubblesort algorithm would have cost around $11.77 (cost of $\sim$2168761 gas)[9].

V. Related work

To the best of our knowledge, only Blockbench [29] and Hyperledger Caliper [30] aim to study the performances of blockchains systems. Blockbench is an academic tool that aims to analyze private blockchains. Hyperledger Caliper is a tool maintained by the Hyperledger Foundation. Main differences in term of functionalities between those two tools and BCTMark are presented in Table II[10].

Blockbench uses two workloads, YCSB [31] and Smallbank [32], to quantify the transaction rate, latency, scalability and failure resistance of three blockchain technologies (two implementations of Ethereum [33][34] and one of Hyperledger [35]). Deployment of blockchains to be tested is managed through bash scripts that do not offer abstractions over the targeted testbed. On the contrary, playbooks written in Ansible and deployed with BCTMark can be used to deploy an arbitrary number of peers on any testbed that supports SSH connections. BCTMark also provides the same abstraction over network, enabling scientists to express easily network constraints and topology. While Blockbench collects metrics about performances (latency and throughput), BCTMark can also collect both system metrics like CPU, memory or disk usage (important to consider the overall footprint of blockchain technologies) and functional metrics (e.g., number of connected peers). Finally, Blockbench only targets private blockchain whereas BCTMark also target public blockchains (as demonstrated in the experiments).

Hyperledger Caliper is well integrated with Hyperledger products and offers a complete lifecycle similar to the one we introduced in subsection III-A. Hyperledger Caliper can monitor blockchain performances but also server metrics through Prometheus [36]. Unfortunately, it does not seems to include any network emulation, crucial for studies on the impact of network failure or latency on a blockchain. Moreover, Hyperledger Caliper does not seem to include any functionality for resources reservation on scientific testbeds like Grid'5000.

VI. Future work

The goal of this paper is to illustrate BCTMark's capacities. To do so, we implemented experiments on two different testbeds and three different blockchains. The next step in the development of BCTMark would be the integration of new blockchain systems or layer-two blockchain solutions (e.g., the Lightning network [15]). We would also like to define and implement benchmarking scenarios that are specific to blockchains (like performance under network partition).

VII. Conclusion

In this paper, we introduce BCTMark, a framework for benchmarking blockchains. Existing tools, while promising, do not include important aspects of reproducible experiments on blockchain systems like network emulation or reproducible deployment. BCTMark aims to empower developers and researchers to create reproducible experiments on blockchain performances. For this purpose, BCTMark provide abstractions over testbeds and network. To facilitate the development of benchmarks, BCTMark includes functionalities like

[8]Ethereum Virtual Machine, the VM running smart contracts

[9]Calculated on https://ethgasstation.info with average gas price

[10]*SUT = System Under Test*

	Blockbench	Hyperledger Caliper	BCTMark
Targeted systems	Private blockchains	Mainly Hyperledger systems	Every blockchain
Deployment management	No	Yes	Yes
Network emulation abstraction	No	No	Yes
Portability to new testbeds	N/A (do not manage deployment)	Yes (But no management of ressources reservation)	Yes (as long as testbed has SSH)
Metrics collection	Yes (SUT performances)	Yes (SUT + Testbed)	Yes (SUT + Testbed)

TABLE II
COMPARISON OF FUNCTIONALITIES WITH THE STATE OF THE ART

load generation and metrics collection. To illustrate BCTMark's functionalities, we have run three experiments on three blockchains (Ethereum Ethash *vs* Clique and Hyperledger Fabric) and two testbeds (one Grid'5000 cluster and one Raspberry Pi cluster). BCTMark's code is open-source and accessible here: https://gitlab.inria.fr/dsaingre/bctmark.

ACKNOWLEDGMENT

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr).

This paper has been financed by the *HYDDA FSN* project.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," *Journal for General Philosophy of Science*, vol. 39, no. 1, pp. 53–67, 2008.
[2] J. Zawistowski, P. Janiuk, A. Regulski, A. Skrzypczak, A. Leverington, P. Bylica, M. Franciszkiewicz, P. Peregud, A. Banasiak, M. Stasiewicz, R. Zagórowicz, and W. Davis, "The Golem Project," https://golem.network/crowdfunding/Golemwhitepaper.pdf, 2016, whitepaper.
[3] Steem, "Steem - An incentivized, blockchain-based, public content platform," https://steem.com/steem-whitepaper.pdf, 2016, whitepaper.
[4] "E-Estonia," https://e-estonia.com/.
[5] P. Labs, "Filecoin: A Decentralized Storage Network," https://filecoin.io/filecoin.pdf, 2017, whitepaper.
[6] I. Storj Labs, "Storj: A Decentralized cloud storage network framework," https://storj.io/storjv3.pdf, 2018, whitepaper.
[7] M. Mylrea and S. N. G. Gourisetti, "Blockchain for smart grid resilience: Exchanging distributed energy at speed, scale and security," in *2017 Resilience Week (RWS)*. IEEE, 2017, pp. 18–23.
[8] E. Mengelkamp, J. Gärttner, K. Rock, S. Kessler, L. Orsini, and C. Weinhardt, "Designing microgrid energy markets: A case study: The brooklyn microgrid," *Applied Energy*, vol. 210, pp. 870–880, 2018.
[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. Van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 45–59.
[10] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.
[11] H. Vranken, "Sustainability of bitcoin and blockchains," *Current opinion in environmental sustainability*, vol. 28, pp. 1–9, 2017.
[12] K. J. O'Dwyer and D. Malone. (2014) Bitcoin mining and its energy footprint.
[13] F. Saleh, "Blockchain without waste: Proof-of-stake," *Available at SSRN 3183935*, 2019.
[14] T. Rocket, "Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies," 2018.
[15] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.
[16] D. Balouek, A. Carpen Amarie, G. Charrier, F. Desprez, E. Jeannot, E. Jeanvoine, A. Lèbre, D. Margery, N. Niclausse, L. Nussbaum, O. Richard, C. Pérez, F. Quesnel, C. Rohr, and L. Sarzyniec, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science, I. I. Ivanov, M. van Sinderen, F. Leymann, and T. Shan, Eds. Springer International Publishing, 2013, vol. 367, pp. 3–20.
[17] B. Smaalders, "Performance anti-patterns," *ACM Queue*, vol. 4, no. 1, pp. 44–50, 2006.
[18] "Grafana - the open observable platform," https://grafana.com/, accessed: 2019-12-6.
[19] "PoET 1.0 Specification," https://sawtooth.hyperledger.org/docs.
[20] E. Foundation, "A Next-Generation Smart Contract and Decentralized Application Platform," https://github.com/ethereum/wiki/wiki/White-Paper, 2013, whitepaper.
[21] V. Saraph and M. Herlihy, "An empirical study of speculative concurrency in ethereum smart contracts," *arXiv preprint arXiv:1901.01376*, 2019.
[22] "Ethereum Wiki - Ethash," https://github.com/ethereum/wiki/wiki/Ethash.
[23] Enoslib, "Enoslib : A framework to build experimental frameworks on various platforms," https://github.com/BeyondTheClouds/enoslib, github repository.
[24] Ansible, "Ansible is Simple IT Automation," https://www.ansible.com/.
[25] "Telegraf - the plugin-driven server agent for collecting and reporting metrics." https://github.com/influxdata/influxdb, accessed: 2019-12-6.
[26] "Influxdb - scalable datastore for metrics, events, and real-time analytics," https://github.com/influxdata/telegraf, accessed: 2019-12-6.
[27] "Locust - a modern load testing framework," https://locust.io/, accessed: 2019-12-6.
[28] J. Pastor and J. M. Menaud, "Seduce: a testbed for research on thermal and power management in datacenters," in *2018 26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE, 2018, pp. 1–6.
[29] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan, "Blockbench: A framework for analyzing private blockchains," in *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 2017, pp. 1085–1100.
[30] "Hyperledger caliper," https://www.hyperledger.org/projects/caliper, accessed: 2019-12-6.
[31] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, pp. 143–154.
[32] M. J. Cahill, U. Röhm, and A. D. Fekete, "Serializable isolation for snapshot databases," *ACM Transactions on Database Systems (TODS)*, vol. 34, no. 4, p. 20, 2009.
[33] "Go-ethereum - official go implementation of the ethereum protocol," https://github.com/ethereum/go-ethereum, accessed: 2019-12-6.
[34] "Parity - fast and feature-rich multi-network ethereum client." https://github.com/paritytech/parity-ethereum, accessed: 2019-12-6.
[35] "Hyperledger fabric," https://github.com/hyperledger/fabric, accessed: 2019-12-6.
[36] "Prometheus - from metrics to insight," https://prometheus.io/, accessed: 2019-12-6.