

BlockCompass: A benchmarking platform for blockchain performance

Mohammadreza Rasolroveicy, *Member, IEEE*, Wejdene Haouari, and Marios Fokaefs, *Member IEEE*

Abstract—Blockchain technology has gained momentum among researchers and other stakeholders due to its immutability and transparency. Several blockchain platforms with different kinds of consensus protocols have been proposed. However, this makes choosing and configuring such a platform, a non-trivial task. Several benchmarking tools have been presented to test the performance of blockchain solutions. However, these solutions are either limited to specific blockchain platforms or require complex configurations. Moreover, they tend to focus on transactional evaluation models, which may be counter-intuitive for longer-running instances under continuous workloads. In this work, we present *BlockCompass*, an all-inclusive blockchain benchmarking tool that can be easily configured and extended. We demonstrate how BlockCompass can evaluate the performance of a variety of blockchain platforms and configurations, including Ethereum Proof-of-Authority, Ethereum Proof-of-Work, Hyperledger Fabric Raft, Hyperledger Sawtooth with Proof-of-Elapsed-Time, Practical Byzantine Fault Tolerance and Raft consensus algorithms against workloads that continuously fluctuate over time. We also present the results of a usability study about the convenience and facility offered by BlockCompass in blockchain benchmarking.

Index Terms—Blockchain, decentralized database, protocols, software performance

1 INTRODUCTION

Blockchain technology is gaining ever-increasing popularity among researchers and practitioners because of its consistency, high availability, transparency and immutability [1]. The technology has begun to benefit various fields, including, but not limited to, decentralized finance (DeFi), retail, Internet of Things (IoT), and art [2], [3]. The core concept of blockchain is its consensus algorithm which protects the data against undesired alteration. Blockchain, which is also known as a decentralized ledger technology (DLT), is a set of peer-to-peer nodes which do not trust each other. In blockchain, in order to have a transaction confirmed, the majority of peers in the network should agree to add it in the block, according to the consensus protocol. Each node keeps replicas of the data and has to validate the transaction into the block [4]–[6].

Due to the promising potential of blockchain in different technologies and industries, several blockchain platforms with a variety of consensus algorithms and other properties have been proposed. Each consensus protocol and blockchain technology comes with strengths and limitations in terms of performance, resource utilization and data integrity. Due to this variability, practitioners need an efficient and easy-to-use benchmark tool than can help them decide the best platform based on their needs. Recently, a number of research works have been conducted on blockchain technologies and consensus algorithms. These projects [7]–[10] aim to improve the scalability, resource consumption, and energy consumption of the traditional blockchain technologies such as Bitcoin and Ethereum with Proof of Work.

To evaluate such platforms, benchmark tools, such as Hyperledger Caliper¹, Blockbench [6] and BCTMark [11] have been proposed. However, one important limitation of these tools is that they are designed to perform “Batch Workloads”, which means that the entire workload runs until the experiment is completed or if it fails. This is incompatible with evaluating the system’s long-term scalability and endurance under continuous and variable workloads. Such a scenario may stress the platform differently in ways that are not covered by cross-sectional experiments. Our tool is based on “Transactional Open Workloads” [12]: a variable number of virtual users sending small work units repeated over the execution of an experiment. This type of workload achieves a realistic variance in the platform’s incoming traffic and long-running experiment that tests the platform’s endurance.

The main contributions of this work are as follows.

- **BlockCompass**: a benchmarking tool to evaluate and analyze the blockchain platforms in real-world scenarios where they are required to run continuously under a fluctuating and increased workload.
- A set of extendable and configurable blockchain **drivers** for Ethereum, Hyperledger Fabric, Hyperledger Sawtooth to serve the user’s requests and submit them to the blockchain.
- A real-time and online **monitoring dashboard** to observe different metrics: Resource Consumption (CPU consumption, memory usage, network I/O), Performance metrics (Response Time, Throughput, Error Rate, Arrival Rate).
- A **comparative study** on the performance overhead of the available consensus algorithms in Ethereum PoA (Proof of Authority), Ethereum PoW (Proof of Work),

• The authors are with the Department of Computer and Software engineering, Polytechnique Montreal, Montreal, QC H3T 1J4 Canada (e-mail: mohammadreza.rasolroveicy@polymtl.ca; wejdene.haouari@polymtl.ca; marios.fokaefs@polymtl.ca).

Hyperledger Fabric and Hyperledger Sawtooth, as an example of how BlockCompass can be used.

- A **usability study** with 33 participants to evaluate the user's experience with BlockCompass when considering different blockchain platforms compared to other blockchain benchmarking tools and manual effort.

2 RELATED WORK

2.1 Benchmarking of Blockchain

Dinh et al. [6] presented a benchmarking tool called Blockbench, which is a cross-sectional benchmark framework. This tool can analyze blockchain technologies in terms of latency, throughput, scalability and fault tolerance. The authors have designed drivers for three blockchain platforms including Ethereum, Hyperledger Fabric and Parity. They have studied these platforms with various batch workloads to understand their performance and security, and detect bottlenecks. This tool has two main drawbacks: it supports only batch workloads and it does not support monitoring of resource consumption (CPU, memory, network). Although the tool is open-source, extending it for new blockchain systems is not easy as there is no proper documentation for extendability. It cannot support large-scale experiments in some systems (e.g., more than 16 nodes in Fabric) and some of its drivers are outdated.

Saingre et al. [11] proposed a benchmark framework for reproducible research on blockchain technology performance (latency, throughput, and energy consumption). The BCTMark (Blockchain Technologies Benchmarking) is used for deployment, comparison, and evolution of different blockchain platforms. The tool does not demonstrate high extensibility and reusability, especially when it concerns performance metrics.

Wang et al. [13] used Hyperledger Caliper, a tool to evaluate the latency and throughput of Hyperledger Fabric. This tool is developed by Hyperledger foundation and it can now evaluate several blockchain platforms, including Fabric, Besu and Ethereum. The authors in their batch experimental study have shown that when the number of users querying the system suddenly increases to 500 users, the response time increases 11 times. Caliper supports only batch workloads.

Birim et al. [14] designed "GoHammer" to evaluate the latency, throughput, resource utilization and failed requests in Quorum blockchain network. The tool is partially similar to Quorum Profiling [15] but the difference is that it currently supports only the Raft consensus algorithm and it is easier to start the workload and the network. The tool is a cross-sectional benchmarking tool. The tool still requires several improvements, such as the ability to monitor one blockchain node, more workload scenarios, and more blockchain platforms, extensions that can be provided by other researchers or practitioners, as the tool is open-source.

Gupta et al. [16] proposed the BFT-Bench tool to evaluate the performance and efficiency of different Byzantine Fault Tolerance State Machine Replications (BFT- SMRs). However, State Machine Replicas (SMRs) do not support transaction request verification, which is one of the main features of blockchain consensus protocols and has a significant impact on performance. Their experimental results demonstrate that

this tool could be helpful for practitioners and researchers to evaluate SMRs based on blockchain.

A review of related literature [11], [14], [16]–[19] and our experimentation with existing benchmark tools for blockchain showed that they are often designed and deployed for one or two platforms without a clear or explicit way to add more platforms. More importantly, the existing tools are cross-sectional which means that we need to configure a single batch transaction and we get the results for throughput and latency for the specified configuration. The problem with this kind of benchmark is that we can not simulate a real-time application and to see when and how the application fails when the workload continuously fluctuates over time. In some cases, the benchmark tools do not provide adequate or even live monitoring of the performance and resource consumption of the application, information critical to maintaining the proper operation of the system in the long term.

2.2 Performance Studies on Blockchain

Wang et al. [4] used Hyperledger Caliper to evaluate Ethereum PoW, Hyperledger Fabric, Hyperledger Sawtooth and Fisco-Bcos. They studied the architecture of each platform and what kind of consensus protocols are supported. They only studied the throughput and response time which are the only parameters that Hyperledger Caliper can collect. Their results show that the performance of the Hyperledger Fabric is better than Ethereum and throughput of both Sawtooth and Fisco-Bcos is far better than Hyperledger Fabric.

A limitation of this work is that while Ethereum, Hyperledger Fabric, Hyperledger Sawtooth and Fisco-Bcos support multiple consensus algorithms, it is not explicitly mentioned which consensus algorithms are used in the study of the blockchain platforms.

Kuzlu et al. [20] examined the effects of various batch workloads on the performance of the Hyperledger Fabric v1.4 regarding throughput, latency, and scalability. The authors used the Hyperledger Caliper tool to evaluate the blockchain platform. The findings of the experiments show that hardware capacity, smart contract complexity, and blockchain network architecture all impact on throughput and latency.

Dabbagh et al. [21] used Hyperledger Caliper to present a comparative performance evaluation of two blockchain platforms, Hyperledger Fabric V.1.1 and Ethereum, against four metrics by executing 100 transactions. With respect to average latency and the throughput, Hyperledger Fabric outperformed Ethereum. On the other hand, Ethereum performs better than Hyperledger Fabric in terms of the successful number of transactions when executing the Transfer function. For future work, the authors intend to conduct a study on the scalability of blockchain platforms and also see what is the highest number of transactions that each blockchain platform can handle.

In our previous work, we compared Ethereum PoW, Ethereum PoA, and Hyperledger Fabric [22]. The experiments were carried out manually, including blockchain installation, running the workload and monitoring the system. We observed that Ethereum PoW is the most expensive solution

in terms of CPU utilization, memory usage and response time due to the design of its consensus algorithm. Moreover, we showed that Hyperledger Fabric with Raft consensus algorithm is the most expensive in terms of network usage. Finally, we found that Ethereum PoW consumes much less memory in comparison with Hyperledger Fabric when there are more concurrent users. This study motivated the creation of “BlockCompass,” as presented here.

In summary these studies show that performance analysis of blockchain can be a complex task. On one hand, this is because most platforms have a large number of parameters to be configured that can affect performance, including consensus algorithm, transaction rate, batch size, number of nodes and others. On the other hand, we need to evaluate several performance dimensions to provide complete and holistic conclusions, including throughput, latency, resource consumption (CPU, memory, network) and others. To achieve meaningful and reliable results and reproducible studies, these concerns necessitate extensive and systematic configurations and instrumented monitoring. In addition, we have found that most of these studies are limited by the tools they use. Most notably, the tools employed by these studies use batch workloads, meaning that they are capable of stress testing the platform and identify their capacity. However, in longer running workloads, backlogs may appear which will make the platform’s performance degrade and fail even if there is no high stress at a given moment. For these reasons, we have designed BlockCompass to be configurable to monitor the systems and the experiments to be extensible and to allow for transactional workloads to evaluate the endurance of the platforms under continuous and fluctuating workloads.

3 THE BLOCKCOMPASS BENCHMARKING TOOL

In this section, we characterize the architecture and design of the BlockCompass tool, beginning with its requirements as identified in our literature review, and advancing to its components and details about their implementation for specific platforms, in addition to how the tool can be extended to support experiments on different blockchain platforms.

3.1 Functional and Non-Functional Requirements

The primary objective of BlockCompass is to enable developers (researchers or practitioners) to conduct experiments on the performance of blockchain systems and their different configurations and allow for their comparison. The basic workflow of using BlockCompass is as follows:

- 1) The end-user chooses the target blockchain platform and consensus protocol, and configures the workload settings, in terms of number of concurrent users and transactions per user.
- 2) The *workload generator* creates a continuous flow of data to and from the blockchain network through an adapter.
- 3) The *blockchain adapter* is responsible for the communication between the benchmark environment and the actual blockchain network. The adapter is specific to the network and exposes an API for communication with the workload generator.
- 4) The end-user can monitor the experiment’s progress through a real-time *dashboard* and get a report with all of the experiment’s performance information. The dashboard monitors the workload generator for input related metrics (e.g., transactions sent), the adapter for performance metrics (e.g., latency and error rate) and the network infrastructure for resource consumption.

Metrics are a key component in a benchmark platform. They provide the data to be analyzed during a benchmarking experiment. In its first version, the focus of BlockCompass is on performance and cost and so it currently supports the following metrics:

- **Emit rate:** The workload generator in BlockCompass generates concurrent users. Each user sends the next request only when they receive a response to the previous one. The emit rate indicator represents the average number of requests sent per second by all active users. This metric is mainly used as a baseline to assess the output performance of a blockchain network.
- **Throughput:** The average number of successful transactions that are committed to the blockchain per second. This metric is used to assess the network’s processing capability in terms of the emit rate.
- **Error rate:** The average number of failed transactions per second. This metric is used to assess the integrity of data and the robustness of the network. Failed transactions could depend on many reasons: scalability, bottlenecks, attacks and so on.
- **Latency:** This metric measures the average time between sending a transaction from the workload generator to the blockchain network and receiving a response. When a blockchain node receives a transaction, it first generates a transaction hash and then returns a response. However, generating a transaction hash does not imply that the transaction is added to a block. This is the main performance metric for the network in the context of the benchmark.
- **Resource Consumption:** This measures the consumption of CPU, memory, and network during the experiment. This metric can be used as an indicator of cost or energy consumption and it can be the cause for failures or errors.
- **Scalability:** Measures the variation of the latency, the throughput, the error and emit rates, and the resource consumption when the number of nodes and the number of concurrent users increase or decrease. This is a longitudinal measure that considers changes both in the workload and the network infrastructure. It is assessed as graphs over time presented by BlockCompass dashboard.

BlockCompass is also designed with a few “quality-of-life” properties both for developers and users. The platform guarantees the following quality properties on top of the functional requirements:

- **Extensibility:** BlockCompass is designed in a way that makes it easy to add new workloads, different monitoring tools or support for new blockchain platforms.
- **Usability:** BlockCompass requires minimum understanding of distributed technologies (e.g., cloud) and very little knowledge about blockchain technologies.

We have used container technology to automate most parts of the benchmark including the installation of the blockchain network, the monitoring and the generation of workload. Users can easily configure their test scenario and get the results in a user-friendly report.

- **Maintainability:** BlockCompass is designed to be used by active projects and industry practitioners and to allow for continuous evolution. The BlockCompass code is open-source and follows community standard processes with respect to bug reporting or resolution, and adding new features.

3.2 Architecture

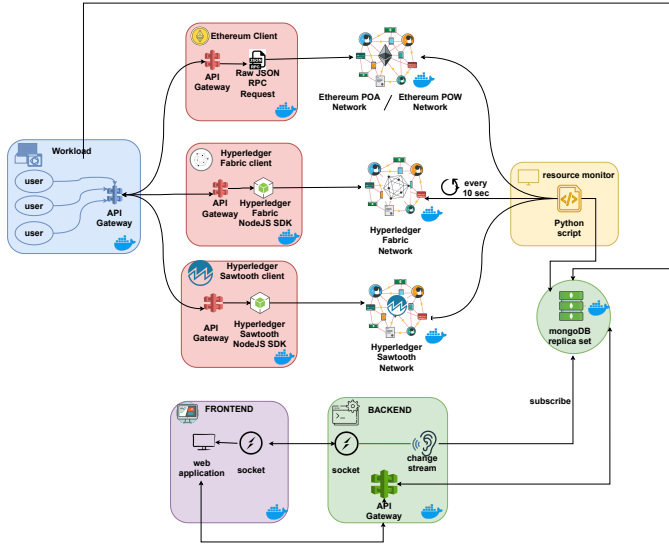


Fig. 1: The BlockCompass architecture

BlockCompass follows a distributed multi-tier architecture, as shown in Figure 1. The first tier is the workload generator, which is responsible for sending requests to the adapter. The two components can be deployed on the same server or even on the client side. The fact that we use containers makes deployment or redeployment of components a fairly trivial task. The adapter then accesses the blockchain network. BlockCompass offers the option to access an existing network, like a publicly available Ethereum network, or a private network, which can also be deployed automatically. Finally, the dashboard, like the adapter and the workload generator, can also be deployed anywhere. Next, we describe each component in detail.

Workload generator: To exercise the subject of blockchain networks, the workload generator simulates users that send requests of different sizes and types of data in different frequencies. The workload is split into time intervals, for which the user of BlockCompass can configure the number of concurrent users, the size of data they can send, and how often they make requests. Finally, the sum of all the intervals determine the duration of an experiment. Given that each interval may have different number of users sending different requests, this creates a continuous and fluctuating workload, giving us the opportunity to test the capacity and endurance of the blockchain network.

Our workload generator is based on a closed transactional model [12, Chapter 8]. In this model, simulated system users send requests and wait for the response, either failed or successful. After some “think time” (as an artificial delay that simulates the processing of a response from the network), the user sends another request. While in an open workload we provide a fixed arrival rate for requests, a closed workload resembles a more realistic scenario with independent users who need to process the system’s response before sending another request.

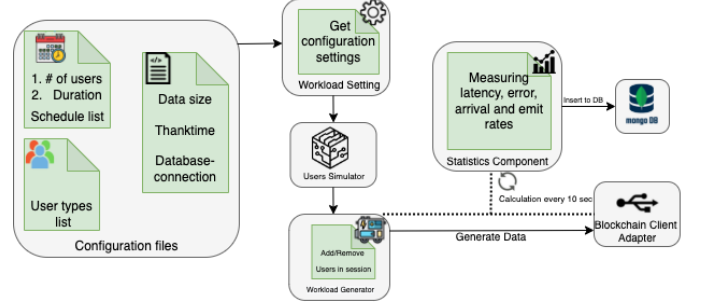


Fig. 2: The architecture of the workload generator

Figure 2 shows how the workload generator works in practice. The configuration contains the number of users at each interval and their type, which signifies what size of data they send every 10 seconds which is also customizable. A scheduler reads one line of the configuration file for every interval (approximately every 10-12 seconds) and it adds or removes active users between intervals accordingly. The data is then sent to the adapter. At the same time, we monitor the emit rate, latency and error rate (in case of failed requests) at the workload level. A monitoring thread calculates averages for these values every 10 seconds and stores them in a MongoDB database.

Client Adapter: Each blockchain platform provides its own unique methods to allow applications to interact with the network. The workload generator is agnostic of the underlying network and it uses a single predetermined format to send the data. The client adapter is then responsible for reformatting the data according to its corresponding network and calling its methods to send the data to the blockchain. All client adapters have the same API to receive data from the workload. This way the workload generator does not need to be changed for every new blockchain network, but an adapter needs to be created instead.

BlockCompass currently supports 4 different blockchain platforms, including: Hyperledger Fabric, Hyperledger Sawtooth, Ethereum with PoA and Ethereum with PoW. Each of the platforms need to have their own client adaptor in order to interact with the main blockchain network. It is usually provided by the developers of the blockchain platform itself.

Ethereum is a well-known public blockchain platform that is primarily used to mine an Ether (ETH) cryptocurrency. However, Ethereum can also be set up as a permissioned/private blockchain platform. Ethereum’s public version depends on the network peers’ computing capacity to validate blocks.

The **Proof of Work (PoW)** or **Proof of Authority (PoA)** consensus system may be set up on an Ethereum network

(see <https://geth.ethereum.org/docs/interface/private-network>). In PoW, network peers are referred to as miners, and they compete to solve a cryptographic challenge. The first one to solve the cryptographic challenge will be rewarded an Ether as an incentive. Next, the miner broadcasts the block to the other nodes in the network.

Proof of Authority (PoA) is another consensus protocol in Ethereum, and it utilizes less computational resources because a block is created by authorized and pre-approved signers based on their reputation, which is specified in the first block, also known as the “genesis block” when the network is created. After the creation, signers can be added or removed using a voting mechanism [23]. The processing of adding blocks in Ethereum PoA is called “sealing” [24], [25]. In BlockCompass configurations, we followed the default parameter and set all the nodes as valid signers.

Ethereum also imposes limitations on gas consumption. The blocks in Ethereum also have a block gas limit which defines the maximum amount of gas a block can include. The initial block gas limit is defined in the genesis block, but this value can continuously fluctuate. As miners use the network, the gas limit can increase or decrease by about 0.1% for each new block.

Hyperledger Fabric is a private blockchain platform. It employs ChainCode² (specialized smart contracts) a type of smart contract programming language with specific functions and states. The purpose in which Fabric and Ethereum were designed varies.

In Fabric, peers are divided into three categories: orderer peers, committing peers, and endorser peers. The peer who owns the Chaincode and will be in charge of endorsing (approving) a proposed transaction is known as the endorser peer. Currently, Hyperledger Fabric is based on Raft Consensus algorithm.

Hyperledger Sawtooth³ is an open-source blockchain platform which is designed especially for supply-chain managements. It is a member of the Intel-developed Linux Foundation umbrella project. Sawtooth’s key advantage over other blockchain systems is its ability to process transactions and data in parallel rather than serially, improving system speed. Sawtooth supports both PoET [26], Raft [4] and Practical Fault Tolerance (PBFT) [16].

The **Proof of Elapsed Time (PoET)** consensus algorithm is based on the Trusted Execution Environment (TEE in Intel® [27]). For the PoET to work correctly, we would need to have at least three nodes. The PoET consensus relies on a “fair lottery” mechanism and Intel SGX CPU is responsible to provide a random sleeping time to all the network participants [27], [28].

The **Practical Byzantine Fault Tolerance (PBFT)** protocol is a voting-based algorithm designed on the Byzantine Fault Tolerance principle, which guarantees the system’s integrity if up to one-third of the network’s total nodes are malicious or corrupted ($n = 3f + 1$, where f is the number of malicious nodes). We need to have $2f + 1$ honest nodes to agree on processing a transaction [29], [30].

The **Raft** consensus protocol [31] is a leader-follower model based on Crash Fault Tolerance (CFT) for small and

restricted platforms. The Raft consensus algorithm requires a significant amount of exchanges among the peers of the network as the majorities should agree on selecting the leader and all network progress. Subsequently, the more nodes we have in the network, the more exchanges we will have, making the extensive network impractical [32].

The **resource monitor** component monitors the CPU, memory, network input and output consumption. BlockCompass is currently implemented using Docker containers, and this also includes the deployment of the supported blockchain networks. Therefore, Docker stats is used as a tool to gather the resource consumption metrics. The collected data will be inserted into a MongoDB collection named “performance” as described in Figure 3.

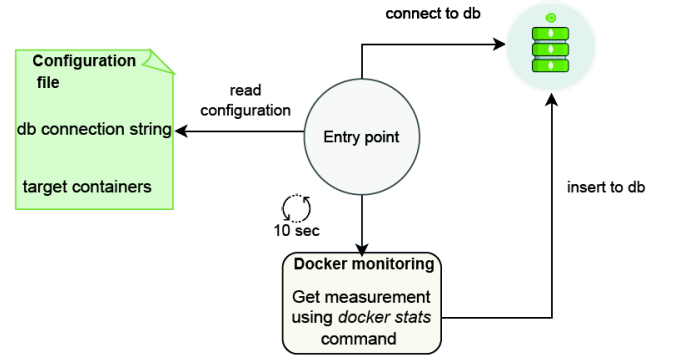


Fig. 3: The architecture of the resource monitor

The data collected by both the Workload and Resource Monitor components are stored in different collections within a **MongoDB Replica Set**. Change Streams is a feature of the MongoDB Replica Set that allows applications to access real-time data updates. We used this feature to create a data stream used by the dashboard to generate real-time data visualizations as the experiment progresses.

The **Back-end Server** is a simple ExpressJS server that provides API endpoints to:

- 1) Get configuration details,
- 2) Get resource consumption data from the database,
- 3) Get performance metrics from the database.

The back-end server, in addition to the API endpoints, can subscribe to the MongoDB Change Streams and, upon receiving any updates, can push them to the front-end client through socket IO events.

The **Front-end Client** is an Angular 10 Web Application that displays real-time charts of all the collected metrics. The charts show the average of each metric over time as well as statistics per node. Aside from the charts, the front-end client displays statistical metrics such as average, standard deviation, variance, minimum, and maximum values in a summary table. It also provides the option of exporting the results as a PDF report.

The **Launcher script**⁴ is a python script that automatically launch all the components needed to run an experiment, namely: the blockchain network, workload, monitor, back-

2. <https://hyperledger-fabric.readthedocs.io/en/release-2.3/>

3. <https://sawtooth.hyperledger.org/faq/consensus/>

4. <https://github.com/polytechnique-ease/blockcompass/blob/master/launcher.py>

end, and frontend components. All containers are run in background.

3.3 Extensibility

BlockCompass' primary goal is to facilitate comparisons and analyses of blockchain platforms' performance for the benefit of researchers and other practitioners. As such, it is important to allow BlockCompass users to customize the composition of the tool to fit their needs, by replacing or extending its modules. Therefore, extensibility is an important BlockCompass characteristic. To achieve this, BlockCompass is built as a framework following the OCP (Open-Closed Principle), meaning it can easily allow for extensions but it is closed for modifications of each concrete components. Furthermore, we follow a "design by contract" approach, meaning that the main components of the framework, namely the workload generator, the client adapter and the resource monitor, expose predefined interfaces, which makes replacing them with different implementations possible as long as the new implementations adhere to these interface and do not affect flow of data between the components.

In this section, we explain how to extend BlockCompass. First, you can add support for a new blockchain platform. Then, you can change the workload. Finally, you can add a new way to track resources.

3.3.1 Extend support for blockchain

The steps to add support for a new blockchain network are as follows.

(i) **Deploy the blockchain network:** To add a new target blockchain, a deployment of this network is required. There are options to deploy private networks locally or to simulate public networks also locally. It is also possible to connect to a public blockchain network. As BlockCompass is currently built around Docker containers, it is recommended to deploy the target network in containers to facilitate monitoring. If a different deployment is available or desired, the Resource Monitor module must be customized in the manner shown below.

(ii) **Implement the corresponding client adapter:** The client adapter is an intermediate component with two interfaces: one incoming from the workload generator and another outgoing towards the tested blockchain. Each blockchain platform has its own client implementation and supported languages. As a result, it is impossible to create a single common outgoing interface for all client adapters. Instead, each client adapter is treated as part of the tested platform, and is the responsibility of the developer. On the contrary the incoming interface is concretely defined to work with our workload generator as follows. The workload will send data to the client adapter, which will be responsible for submitting the transaction to the blockchain network in the correct format. The payload sent by the workload is in JSON format as follows: `{"function": "Write", "args": args}`. Where `args` is a table containing the key-value pairs to be stored in the blockchain network. By default the client adapter runs on port 3000 under the `invoke` API

endpoint, so that it can be called by the workload generator. After receiving the data, the client adapter needs to call the corresponding smart contract (see next step) to submit the data to the blockchain and return a response, either success or failure, to the user (or in practice to the workload generator).

(iii) **Implement the corresponding Smart Contract:** The smart contract contains the mechanism to submit the data from the workload generator to the blockchain network. In the case of the current version of blockcompass, the smart contract function as a simple key-value storage. Listing 1 shows the `kvstore` smart contract in Solidity for Ethereum. The smart contract implements a key-value map and two accessor functions: a function `get` that retrieves a value from the map based on a key and a function `set` that takes a key and value pair as input and stores them in the map.

```

1
2 contract KVStore {
3     function get(string key) constant returns(string)
4         ↪ {
5         ↪     return store[key];
6     }
7     function set(string key, string value) {
8         ↪     store[key] = value;
9     }
10 }
```

Listing 1: Ethereum kvstore Smart Contract

The same functionality for Hyperledger Fabric is shown in Listing 2. Hyperledger Fabric uses `chaincode`, which is the equivalent of a smart contract, implemented in the Go language. Because Hyperledger Fabric interprets the ledger as a key-value map by default, it is not required to specify a new structure as in the case of Ethereum.

```

1
2 type KVStore struct {
3     contractapi.Contract }
4 func (t *KVStore) Write(ctx
5     ↪ contractapi.TransactionContextInterface, key
6     ↪ string, value string) error{
7     ↪ var err error
8     ↪ err = ctx.GetStub().PutState(key,
9     ↪ []byte(value))
10    ↪ if err != nil {return err }
11    ↪ return nil }
```

Listing 2: Hyperledger Fabric kvstore chaincode

(iv) **Adjust monitor script:** The last part of adding a new blockchain platform is adjusting the monitor configuration file by adding the new target blockchain and specifying the containers name pattern to monitor in the environment file⁵. By default, the monitor script will monitor containers whose names match the patterns given in the environment file: `(.*name.*)`. Listing 3 shows an example of the environment file. For each blockchain target, we specify the container pattern names to monitor. For an Ethereum clique network with three miners,

5. <https://github.com/polytechnique-ease/BlockCompass/blob/master/monitor/.env>

the containers names are miner1, miner2, miner3, and bootnode. Using the following environment file, we will monitor all the containers that have the word *miner* in their names. It is also possible to provide more than one name pattern by separating the names with a comma.

```

1  ethereum-clique=miner
2  ethereum-ethereum-pow=miner
3  sawtooth-pbft=validator
4  sawtooth-raft=validator
5  sawtooth-poet=validator
6  Fabric=peer
7

```

Listing 3: Environment file for monitoring

3.3.2 Extend the Workload Generator

The workload generator needs to maintain a connection with the client adapter, as explained earlier, through a given port and a given API endpoint. Besides that, it is possible to change most of the internal implementation details of the workload generator, through a set of configuration files⁶ ⁷. The `users.list` file allows to specify the number of users, type of users and emit rate per user type. By adding new user types or changing the existing ones, the workload generator can be extended for different experiments⁸.

3.3.3 Extend the Resource Monitor

The current version of BlockCompass is based on Docker containers and as such it is using a custom monitoring script⁹ that scrapes the results of `docker stats` to retrieve the metrics for resource consumption. The script then gives the option to save the data in a database or in a file (for reporting purposes). A developer has the option to change the monitoring tool and/or the database to store the metrics. This can be easily done by extending an abstract `Monitoring`¹⁰ class. This class contains methods to connect and write to a database or write to a file. The code to retrieve and organize data is specific to the underlying monitoring tool and needs to be provided by the developer. Moreover, the developer has the option to change the database by providing a concrete extension to the abstract `Database`¹¹ class, which provides an interface for connecting to and interacting with a database. In its current implementation, BlockCompass is using an instance of MongoDB to store monitoring data.

6. <https://github.com/polytechnique-ease/BlockCompass/blob/master/workload/IoT/schedule.list>

7. <https://github.com/polytechnique-ease/BlockCompass/blob/master/workload/IoT/sensors.list>

8. To extend the workload generator to support different type of users and generate specific data or changing the way it stores the data in database, you can visit: <https://github.com/polytechnique-ease/BlockCompass/blob/master/workload/documentation.md>

9. <https://github.com/polytechnique-ease/BlockCompass/blob/master/monitor/dockerMonitoring.py>

10. <https://github.com/polytechnique-ease/BlockCompass/blob/master/monitor/monitoring.py>

11. <https://github.com/polytechnique-ease/BlockCompass/blob/master/monitor/database.py>

4 CASE STUDY

We demonstrate the functionality and the steps to configure and use BlockCompass in the context of a large-scale comparative study. The objective of this study is to compare the scalability, the performance, and the cost, in terms of resource consumption, between three blockchain networks, namely Ethereum, Hyperledger Fabric, and Hyperledger Sawtooth. In this section, we first outline how we implemented the three networks and how we set up BlockCompass for each one of them. The results of the study are then presented, as well as how they can be retrieved efficiently and in a user-friendly manner using BlockCompass.

4.1 Study Motivation

To evaluate the capability and ease of use of BlockCompass, we deployed four different blockchain platforms, including Ethereum with PoW, and Ethereum with PoA consensus algorithms, Hyperledger Fabric with Raft, and Hyperledger with PoET, PBFT, and Raft consensus protocols. All these platforms are commonly used among practitioners, industries, and researchers. In terms of blockchain design and codebase maturity, they hold various positions. Our collaboration with several industrial partners inspired us to design BlockCompass tool. We were tasked to design and prototype a decentralized application on both public and private blockchain platforms with a large number of users and requests in real-time. In addition, we had to investigate and compare several prominent blockchain platforms, consensus protocols, and configurations to find the most scalable platform. We've done various private blockchain comparative tests before the development BlockCompass. In these tests, we had to install platforms, plan workloads, build monitoring scripts, and incorporate blockchain adapters separately and manually.

In this research, we show how much simpler it is to utilize BlockCompass to analyze the performance and resource consumption of multiple blockchain networks, specifically if the performance and resource consumption of the same consensus protocol in two distinct blockchain networks may be influenced. BlockCompass differs from other cross-sectional tools on the market, such as Blockbench [6] or Caliper, in that it can simulate concurrent users who fluctuate over time. In practice and in collaboration with our industrial partners, BlockCompass assisted us in determining bottlenecks of a blockchain network like Hyperledger Sawtooth. More specifically, our experiments showed that when the network gets overloaded with too many requests and users, the whole platform fails and must be reset and restarted. It would have taken days to install all the necessary dependencies to do a side-by-side performance evaluation of various blockchain systems without BlockCompass. Thanks to BlockCompass, as an all-inclusive tool, it is possible to set up the environment and install all dependencies in less than 15 minutes and then repeat the same experiments for different blockchain platforms under similar, controlled conditions. In this section, we demonstrate how BlockCompass can be set up, installed and configured in the context of a comparative analysis between the four blockchain platforms.

4.2 Experimental setup

BlockCompass was installed on a single Amazon `t2.large` virtual machine running Ubuntu 18.04 with 8GB of RAM, 2 vCPUs, and 25GB of hard disc space. This VM served as the host for the Docker containers of: the blockchain network's nodes, workload, monitoring, frontend and backend components. In our testing, each Docker container is given only 20% of the VM's total CPU and memory to ensure that we have enough resources for all elements of the infrastructure. We have used one virtual machine for all components and each component such as miners is a docker container.

For the blockchain networks, which were deployed locally, we used `geth` (v1.10.3) [33] a Go implementation of Ethereum, for Hyperledger Fabric [8], we used v2.3.2, and for Hyperledger Sawtooth, we used v1.2 (Chime) [34]. Each blockchain network is composed of five nodes, deployed in containers. For Ethereum PoW and PoA, we used five miners and one bootnode. For Hyperledger Fabric, we used two organisations, one ordering service and one channel. The first organisation has two peers, and the second one has three peers. We also set the block period to two seconds. Finally, for Hyperledger Sawtooth, we have set up a network for each supported consensus (PBFT, Raft, and PoET), each running five validator nodes.

As far as the workload is concerned, we used different types of users that send queries ranging from 70 to 80 KB. The length of each experiment was fixed at 16 minutes, and the number of users ranged from 0 to 100. The number of users, emit rate and the duration of their existence can be easily customized in `workload/IoT/schedule.list`.

4.3 Results

As shown in Figure 4a, the number of queries handled within 16 minutes varies due to the variation in response time across platforms. Table 1 shows that Ethereum PoA has the greatest overall throughput. This is due to the fact that Ethereum PoA has the shortest response time, while Ethereum PoW is on the other end of the spectrum. As we said before, this divergence is to be expected and is based on how the consensus algorithm works. In contrast, Sawtooth has a lower throughput, despite being implemented in both Hyperledger Fabric and Hyperledger Sawtooth algorithms. This is because, as seen in Figure 4b, Sawtooth-Raft began to experience errors and requests were failing. As a result, the throughput is dependent on the consensus and the platform implementation. In contrast, Sawtooth has a lower throughput, despite being implemented in both Hyperledger Fabric and Hyperledger Sawtooth algorithms. And the error rate in Sawtooth Raft varies with the number of users; as the number of users grows, so does the error rate, indicating that the system cannot manage a large number of users at the same time. In our experiments, as shown in Figure 4b, Sawtooth PoET can only handle concurrent users until 250 seconds before the network falls down and must be restored. This coincides with the first large increase in the number of users and results in the total collapse of the platform after the second large increase. As we can see, the Sawtooth Raft error rate varies with the number of users. Only the PBFT consensus protocol in Sawtooth is error-free throughout the experiment.

Figure 4d illustrates the average response time for 6 platforms. We can observe that there is a correlation between workload and response time in terms of the number of users in all the blockchain networks, as is expected. We can also observe based on Figure 4d and Table 1 that Ethereum PoW and Sawtooth PBFT have the highest response times among all platforms which is 2034.67ms for Ethereum PoW and 2116.62ms for Sawtooth PBFT. This is because the generation of a transaction hash requires more resources for these two algorithms in particular. When comparing response time of Raft in Fabric and Sawtooth, we see a noticeable difference between the two, indicating that response time depends on the consensus as well as the platform itself. Our assumption is that the architecture and implementation of the blockchain platform for both platforms are different, for example, Hyperledger Fabric consists different network peers components including Endorser, Anchor, and Orderer peers, and only Orderer peer is responsible for creating a block and reaching the consensus based on Raft consensus protocol; therefore, for communications, only Orderer peers are involved. Hyperledger Sawtooth only consist of multiple validators, and all of them are responsible for adding a new block; therefore, communication is required among them to reach a consensus. Moreover, in Sawtooth they have extended the Raft algorithm to use a stable storage mechanism as the original implementation of Raft is based on in-memory storage and is faster for communication. The reason for this extension is to enable Sawtooth to restart in the event of a crash or arbitrary shutdown.

In terms of CPU utilization as it is shown on figure 4e and Table 1. Ethereum PoW has the highest CPU consumption of about 29.19%, mainly explained by how the consensus algorithm works. We also observed that the CPU consumption for Ethereum PoW remains relatively constant regardless of the fluctuation in the workload. This is the result of the difficulty parameter in the PoW consensus algorithm. When the number of transactions and load increase in the network, the difficulty reduces, and when we have either nothing or a few transactions in the system, the difficulty parameter increases, resulting in a balanced CPU usage throughout the experiment. Ethereum PoA is the most efficient algorithm in terms of CPU consumption, as it is only based on authority of each node of the network. In other words, we trust the miner to reach a consensus, and there is no mathematical puzzle to solve, so fewer resources are required. We can see that high CPU correlates with higher response time, especially for the three slowest networks (Ethereum PoW, Sawtooth PBFT and Sawtooth Raft). This is natural as the first two requiring significant work during the validation of blocks, while in the third Sawtooth in general is the ones that requires the higher workload, which motivated the use of PoET consensus.

As shown in Figure 4f memory is not a significant resources for any of the studied networks. Therefore, it is difficult to draw any specific arguments. On the other hand, Figure 4g shows the network traffic between the nodes. Here we can see that Raft (either in Fabric or in Sawtooth) and PoA in Ethereum have the highest network activity. For the two of them (Fabric and Ethereum), this compensates for the low CPU usage, as in these cases, validation occurs through communication between the peers rather than any particular computation load on each peer. In Sawtooth Raft the two

TABLE 1: Summary of results of comparison between Hyperledger Fabric (Raft), Ethereum PoA , Ethereum PoW, Sawtooth PBFT, Sawtooth Raft and Sawtooth PoET

Algorithms	Fabric Raft	E-PoA	E-PoW	S-PBFT	S-Raft	S-PoET
total number of requests	30770	53135	13986	12865	14671	31156
Total throughput	30770	52743	13986	12865	12175	3844
Avg Error (%)	0	0.40	0	0.02	13.00	62.00
St.Dev. Error	0	0.028	0	0.0014	0.16	0.46
Avg CPU (%)	10.36	8.24	29.19	21.62	24.51	13.71
St.Dev. CPU	3.43	4.84	7.5	4.37	3.59	8.65
Avg. Response Time (ms)	385.89	50.60	2034.67	2116.62	1849.53	447.92
St.Dev. Response Time	456.35	79.77	2032.28	2064.89	1937.93	868.03
Avg Memory (%)	0.75	3.53	6.29	1.12	1.1	1.16
St.Dev. Memory	0.07	0.87	1.39	0.35	0.33	0.23
Avg. Network Input (MB)	39.24	58.21	17.45	36.81	61.24	23.73
St.Dev. Network Input	154.15	31.15	7.25	23.39	38.08	10.87
Avg. Network Output (MB)	144.3	35.35	16.27	43.45	67.50	32.65
St.Dev. Network Output	108.14	27.85	6.63	27.49	41.87	12.9

resources (CPU and network) are used to a similar degree for presumably better balance between performance and security. In fact, Fabric requires fewer nodes to validate a transaction (the endorsed peers), while Sawtooth will use all of its validators.

4.4 Discussion

In previous works, we had performed similar studies to compare different blockchain platforms [35] or different consensus algorithms [36]. Without having BlockCompass, we had to firstly, install all the dependencies for each platform individually, setting up environments, checking the latest packages and see whether there are conflicts, installing the blockchain platform individually, designing a custom closed-workload for continuous and fluctuating users with different workload sizes, writing a custom monitoring script to capture the performance and resource usage, writing a script to export the results and finally generate figure and statistical results. Only the installation, configuration and execution of the experiments (without the final analysis) could take a couple of days by a single researcher. With BlockCompass it is possible to fully automate or at least systematize some of these steps. This results in a set up of less than 15 minutes, again by a single researcher or tester. Thanks to its container technology, BlockCompass allows the execution of an experiment with a single command, and eventually produces a publication ready report with summary results and visualizations. Workload and other configurations can be changed easily and in a rapid manner. Live monitoring helps to decide faster which configuration is suitable compared to waiting for the experiment to finish, and exporting and analyzing results.

5 USABILITY STUDY

In order to evaluate the usability and efficiency of BlockCompass in an unbiased way, especially in comparison to other similar blockchain benchmarking tools, we conducted a usability study involving a group of senior undergraduate software engineering students enrolled in software architecture and advanced design course at Polytechnique Montreal. The students were asked to conduct a small-scale comparative study with either BlockCompass or with

another blockchain benchmarking tool named Blockbench [6]. Blockbench is designed to perform “Batch Workloads”, which means that the workload runs until the experiment is completed or if it fails. On the other hand, BlockCompass is designed to create transactional workloads, which means that it simulates concurrent and fluctuating users with different data size to stress the network and simultaneously monitor resource usage and performance in real-time. After the completion of the study, we asked the students to answer a set of questions regarding their experience with the tool in an anonymous survey¹². The students were given a live demo of each tool in 2 different live sessions. Also, a user manual and offline tutorial were provided for the students to follow the steps required for both Blockbench and BlockCompass.

Figure 5 shows a flowchart on the participation in the usability study. 33 students were initially enrolled in the course and all were contacted to participate in the study. Eventually, 26 (or 79%) accepted to participate. Out of these, 15 (or 58%) used BlockCompass and 11 (or 42%) used Blockbench. Also, 14 (or 54%) used Fabric (8 with BlockCompass and 6 with Blockbench) and 12 (or 46%) used Ethereum (7 with BlockCompass and 5 with Blockbench). Overall, we had an acceptable balance between benchmarking and blockchain platforms.

The questions focus on four different measurable quantities that are spread across all four stages of the experimentation process: the installation phase, the configuration phase, the experimentation phase, and the reporting phase. Installation refers to installing prerequisites and setting up the platforms, including both the blockchain and the benchmark platforms. Configuration mainly refers to setting up the experiment, including setting up the environment and the virtual machines, configuring the workload and other tasks. Experimentation is simply running the experiment and reporting is getting the output, that may include data files or visualizations. For each of these phases, we enquired about the time it took on average to complete the phase, the number of attempts it required to successfully complete the phase, the number of errors that occurred during these attempts and, finally, about what percentage of these errors

¹². The usability study received ethics approval from Polytechnique Montreal Research Ethics Committee, Certificate #CER-2122-29-D-14 December 2021

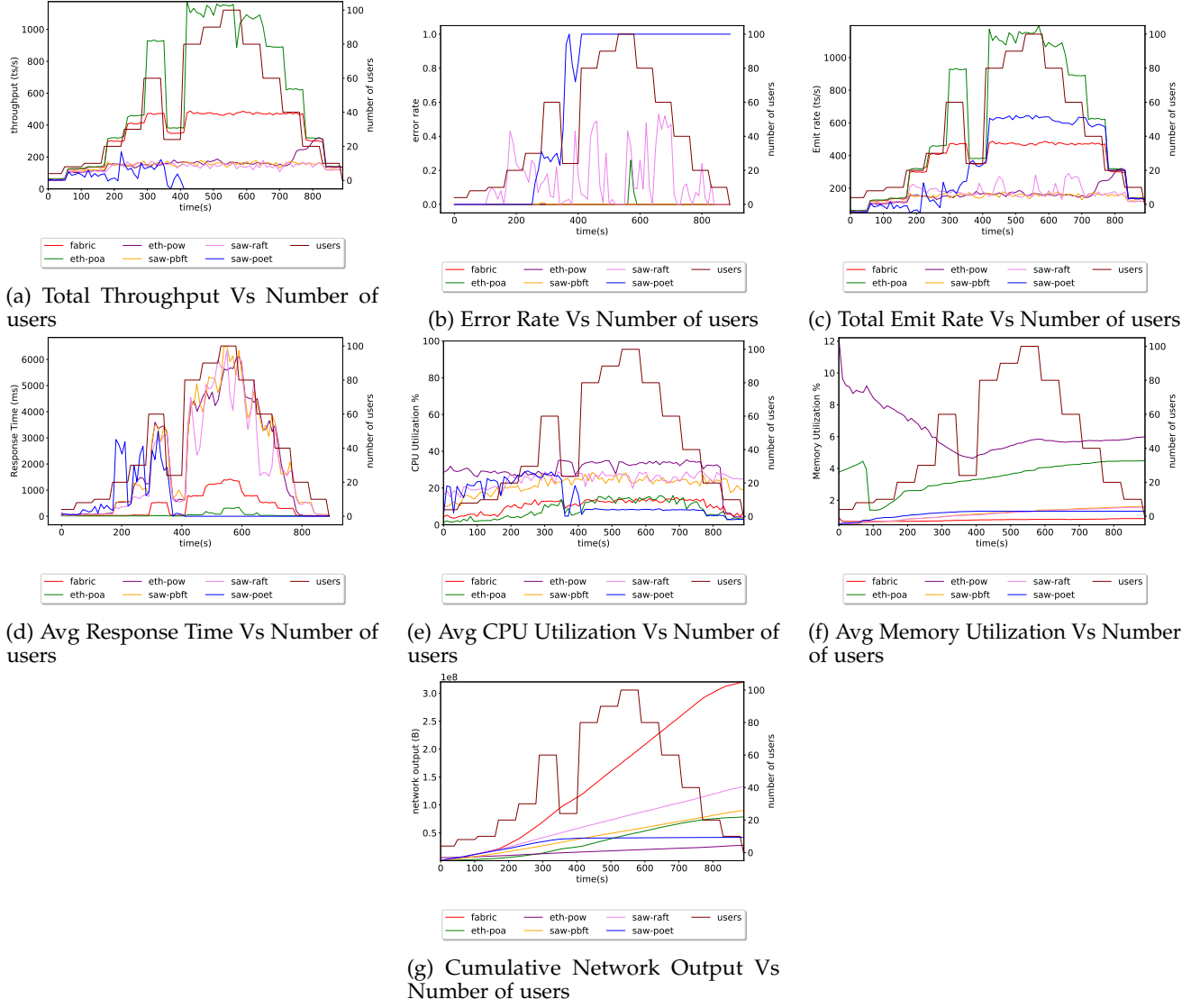


Fig. 4: Number of send requests, Average Response Time, CPU Utilization, Memory Utilisation and Network throughput using 5 nodes and 1 receiver node

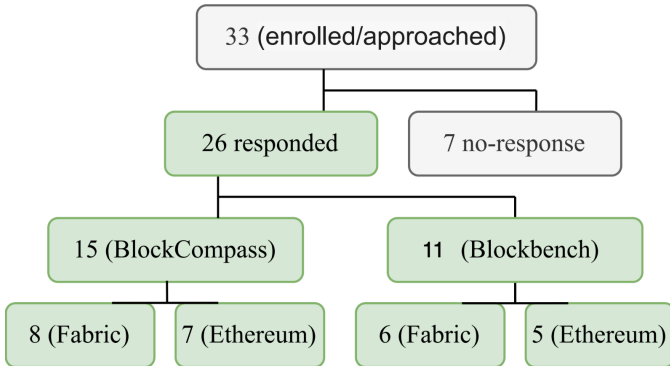


Fig. 5: Flowchart on the participation in the usability study

the participant believed was due to problems in the documentation or in the instructions provided. We also asked the participants to submit a complete list of their attempts with the associated time and errors, but we only received

4 responses, which we eventually did not analyze further. Finally, we included 5 questions on a Likert scale (1-10) about the participant's general satisfaction of the benchmark platform, its documentation, the reporting of results and whether it is generally easier to use a benchmark tool rather than perform the experiments manually.

To test the statistical significance of a difference between the averages for BlockCompass and Blockbench, we applied the Student's *t* [37] and the Wilcoxon Mann-Whitney *U*-tests [38] and for each question, depending on whether the responses were normally distributed or not, respectively. Normality was checked for each question with the Shapiro-Wilk test [39]. Moreover, as we are doing multiple comparisons (i.e., for multiple questions), we applied a Bonferroni correction [40], so that $p = 0.05/m$, where m is the number of questions, in our case 20. Finally, if $p < 0.0025$ in the original *U* or *t*-tests, the null hypothesis (i.e., the two averages do not differ) was rejected.

Table 2 shows the average responses for all questions between BlockCompass and Blockbench. As it can be seen

with respect to time, the two platforms are comparable, with BlockCompass being slightly faster when it comes to configuration and reporting. Similarly, using BlockCompass resulted in fewer attempts for each phase, with the exception of running experiments, however these differences were not found to be significantly different. Users consistently encountered more errors when working with Blockbench compared to BlockCompass, again with little to no statistically significant difference. Users reported that these errors mainly stemmed from lack of clarity in the documentation and instructions on Blockbench especially when it concerned installation errors (87.37% of errors). This difference was found to be statistically significant. Overall, participants were much happier with their experience on BlockCompass (scoring it between 7-8) than they were with Blockbench (scoring it between 4-5). This made the BlockCompass participants more favorable towards automatic benchmark solutions than manual efforts.

TABLE 2: Survey results. Each row reports the average of all responses for the two tools and the p -value for the corresponding statistical test (t for Student t-test and normal distribution, U for Mann-Whitney U-test and non-normal distribution). Significant differences ($p < 0.0025$) are noted in grey.

Question	BlockCompass	Blockbench	p -value
Installation-Time	27.5 min	40.27 min	0.01 ^t
Installation-Attempts	4.13	10.36	0.01 ^t
Installation-Errors	2.73	3.55	0.19 ^t
Installation-DocErrors	27.57%	87.37%	0.001 ^U
Configuration-Time	34.87 min	38.50 min	0.59 ^t
Configuration-Attempts	4.13	5.18	0.17 ^U
Configuration-Errors	2.73	4.55	0.028 ^t
Configuration-DocErrors	34.87%	52.73%	0.006 ^U
Experiment-Time	53.13 min	53.10 min	0.47 ^U
Experiment-Attempts	5.07	3.90	0.96 ^U
Experiment-Errors	3.13	4.60	0.31 ^U
Experiment-DocErrors	36.67%	41.90%	0.23 ^t
Report-Time	31.87 min	44.90 min	0.05 ^U
Report-Attempts	4.20	4.30	0.94 ^t
Report-Errors	2.21	3.30	0.11 ^U
Report-DocErrors	19.33%	37.82%	0.09 ^t
Overall experience	7.07	5.27	0.001 ^t
Documentation quality	7.20	4.82	0.003 ^t
Presentation quality	7.40	4.82	0.00004 ^t
Ease/Speed vs Manual	7.93	5.18	0.0007 ^U
Ease/Speed vs Other tools	7.05	4.27	0.00001 ^t

6 CONCLUSION

In this paper, we presented, BlockCompass, a novel benchmarking tool for blockchain platforms and their configurations. BlockCompass comes with a configurable workload generator that simulates concurrent users overtime, three adapters/clients for Ethereum, Hyperledger Fabric and Hyperledger Sawtooth, a monitoring framework based on Docker containers and a reporting tool with a live dashboard

or exportable results in PDF or CSV format. The benchmarking platform is configurable and extensible and can be used in a variety of studies or stress and capacity tests to evaluate the performance and the capabilities of blockchain solutions.

We demonstrate the use of BlockCompass and its capabilities in the context of an empirical study to compare three distinct blockchain platforms Ethereum, Hyperledger Fabric, and Hyperledger Sawtooth that employ distinct consensus protocols. During the study, we discuss how the experiments can be set up, what tools need to be configured and how to prepare the blockchain platforms to be compared. Eventually, we discuss the results of the study as they were obtained with the help of BlockCompass.

To evaluate the quality and usability of BlockCompass, we have conducted a survey study and compared it with Blockbench, an existing benchmarking tool for blockchain. Overall, the participants of the study were significantly more satisfied with BlockCompass regarding their experience with running benchmark tools, clarity of the documentation, presentation of final results, ease of performing experiments compared to manual effort or to other tools. In general, BlockCompass performed slightly better than Blockbench in terms of the amount of time it took to complete the experiments, as well as configuration, reporting, and documentation.

ACKNOWLEDGMENTS

We acknowledge the support of the NSERC CREATE DITA program, and Mitacs Accelerate project # IT25754.

REFERENCES

- [1] A. A. Monrat, O. Schelén, and K. Andersson, "A survey of blockchain from the perspectives of applications, challenges, and opportunities," *IEEE Access*, vol. 7, pp. 117134–117151, 2019.
- [2] M. Li, S. Shao, Q. Ye, G. Xu, and G. Q. Huang, "Blockchain-enabled logistics finance execution platform for capital-constrained e-commerce retail," *Robotics and Computer-Integrated Manufacturing*, vol. 65, p. 101962, 2020.
- [3] J. Chen, T. Cai, W. He, L. Chen, G. Zhao, W. Zou, and L. Guo, "A blockchain-driven supply chain finance application for auto retail industry," *Entropy*, vol. 22, no. 1, p. 95, 2020.
- [4] R. Wang, K. Ye, T. Meng, and C.-Z. Xu, "Performance evaluation on blockchain systems: A case study on ethereum, fabric, sawtooth and fisco-bcos," in *International Conference on Services Computing*, pp. 120–134, Springer, 2020.
- [5] 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*, pp. 1085–1100, 2017.
- [6] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE transactions on knowledge and data engineering*, vol. 30, no. 7, pp. 1366–1385, 2018.
- [7] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16440–16455, 2020.
- [8] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, pp. 1–15, 2018.
- [9] M. Schäffer, M. Di Angelo, and G. Salzer, "Performance and scalability of private ethereum blockchains," in *International Conference on Business Process Management*, pp. 103–118, Springer, 2019.
- [10] B. Ampel, M. Patton, and H. Chen, "Performance modeling of hyperledger sawtooth blockchain," in *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pp. 59–61, IEEE, 2019.

- [11] D. Saingre, T. Ledoux, and J.-M. Menaud, "Bctmark: a framework for benchmarking blockchain technologies," in *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–8, IEEE, 2020.
- [12] S. Kounev, K.-D. Lange, and J. von Kistowski, *Systems Benchmarking*. Springer, 2020.
- [13] K. Wang, M. Liu, X. Jiang, C. Yang, and H. Zhang, "A novel vehicle blockchain model based on hyperledger fabric for vehicle supply chain management," in *International Conference on Blockchain and Trustworthy Systems*, pp. 732–739, Springer, 2019.
- [14] M. Birim, H. E. Ari, and E. Karaarslan, "Gohammer blockchain performance test tool," *Journal of Emerging Computer Technologies*, vol. 1, no. 2, pp. 31–33, 2021.
- [15] Quorum, "Quorum Profiling, 2021." <https://github.com/ConsenSys/quorum-profiling>, 2021. Accessed: 2021-11-02.
- [16] D. Gupta, L. Perronne, and S. Bouchenak, "Bft-bench: Towards a practical evaluation of robustness and effectiveness of bft protocols," in *IFIP International Conference on Distributed Applications and Interoperable Systems*, pp. 115–128, Springer, 2016.
- [17] M. Di Pierro, "What is the blockchain?," *Computing in Science & Engineering*, vol. 19, no. 5, pp. 92–95, 2017.
- [18] A. Baliga, N. Solanki, S. Verekar, A. Pednekar, P. Kamat, and S. Chatterjee, "Performance characterization of hyperledger fabric," in *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pp. 65–74, IEEE, 2018.
- [19] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-enabled smart contracts: architecture, applications, and future trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, 2019.
- [20] M. Kuzlu, M. Pipattanasomporn, L. Gurses, and S. Rahman, "Performance analysis of a hyperledger fabric blockchain framework: throughput, latency and scalability," in *2019 IEEE international conference on blockchain (Blockchain)*, pp. 536–540, IEEE, 2019.
- [21] M. Dabbagh, M. Kakavand, M. Tahir, and A. Amphawan, "Performance analysis of blockchain platforms: Empirical evaluation of hyperledger fabric and ethereum," in *2020 IEEE 2nd International Conference on Artificial Intelligence in Engineering and Technology (IICAIET)*, pp. 1–6, IEEE, 2020.
- [22] M. Rasolroveicy, W. Haouari, and M. Fokaefs, "Public or private? a techno-economic analysis of blockchain," in *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*, pp. 83–92, 2021.
- [23] geth, "Geth private network, 2019." <https://geth.ethereum.org/docs/interface/private-network>, 2019. Accessed: 2021-06-10.
- [24] P. Ekparinya, V. Gramoli, and G. Jourjon, "The attack of the clones against proof-of-authority," *arXiv preprint arXiv:1902.10244*, 2019.
- [25] B. academy, "poaexplained 2022," 2022.
- [26] D. Ongaro and J. Ousterhout, "The raft consensus algorithm," *Lecture Notes CS*, vol. 190, 2015.
- [27] A. Baliga, "Understanding blockchain consensus models," in *Persistent*, scholar.archive.org, 2017.
- [28] K. Olson, M. Bowman, J. Mitchell, S. Amundson, D. Middleton, and C. Montgomery, "Sawtooth: An introduction," *The Linux Foundation, Jan*, 2018.
- [29] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of pbft consensus process for permissioned blockchain network (hyperledger fabric)," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, pp. 253–255, IEEE, 2017.
- [30] M. Castro, B. Liskov, et al., "Practical byzantine fault tolerance," in *OsDI*, no. 1999 in 99, pp. 173–186, 1999.
- [31] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pp. 305–319, 2014.
- [32] D. Huang, X. Ma, and S. Zhang, "Performance analysis of the raft consensus algorithm for private blockchains," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.
- [33] E. Go, "Ethereum Go 2021," 2021.
- [34] H. Sawtooth, "Hyperledger sawtooth 1.2 (chime)." <https://sawtooth.hyperledger.org/release/chime/>, 2019. Accessed: 2022-01-10.
- [35] M. Rasolroveicy and M. Fokaefs, "Performance evaluation of distributed ledger technologies for iot data registry: A comparative study," in *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)*, pp. 137–144, IEEE, 2020.
- [36] M. Rasolroveicy and M. Fokaefs, "Dynamic reconfiguration of consensus protocol for iot data registry on blockchain," in *Proceedings of the 30th Annual International Conference on Computer Science and Software Engineering*, pp. 227–236, 2020.
- [37] G. D. Ruxton, "The unequal variance t-test is an underused alternative to student's t-test and the mann-whitney u test," *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, 2006.
- [38] P. E. McKnight and J. Najab, "Mann-whitney u test," *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.
- [39] P. Royston, "Approximating the shapiro-wilk w-test for non-normality," *Statistics and computing*, vol. 2, no. 3, pp. 117–119, 1992.
- [40] E. W. Weisstein, "Bonferroni correction," <https://mathworld.wolfram.com/>, 2004.



Mohammadreza Rasolroveicy is a PhD candidate and teacher assistant at Polytechnique University of Montreal. His research interests include Internet of Things, software quality, blockchain scalability and self adaptive systems. He is part of DITA NSERC CREATE program and his works are funded by Mitacs Accelerate Awards. His project focuses on how to leverage self-adaptive system in balance cost, performance and energy consumption for NFT marketplaces.



Wejdene Haouari is a MSc student in the Department of Computer Engineering and Software Engineering at Polytechnique Montreal University. She completed her BEng in Computer Engineering at the National Institute of Applied Science and Technology in Tunisia majoring in software security. Her research interests include Blockchain, software security and cloud computing. Her Project focuses on blockchain smart contract security.



Marios Fokaefs is an Assistant Professor in the Department of Electrical Engineering and Computer Science at the Lassonde School of Engineering, where he directs the EASE (Economics and Administration of Software Engineering) lab. His expertise revolves around Software Engineering and more specifically on software evolution and DevOps. His research focuses on Software Engineering Economics, Software Performance Engineering, Cloud Computing and Self-Adaptive Systems among others.