

BlockCompass: A benchmarking platform for blockchain performance

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

Abstract—Blockchain technology has gained momentum due to its immutability and transparency. Several blockchain platforms, each with different consensus protocols, have been proposed. However, choosing and configuring such a platform is a non-trivial task. Numerous benchmarking tools have been introduced to test the performance of blockchain solutions. Yet, these tools are often limited to specific blockchain platforms or require complex configurations. Moreover, they tend to focus on one-off batch evaluation models, which may not be ideal 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 various 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 show how continuous transactional workloads may be more appropriate than batch workloads in capturing certain stressful events for the system. Finally, we present the results of a usability study about the convenience and effectiveness offered by *BlockCompass* in blockchain benchmarking.

Index Terms—Benchmark, Performance Testing, Blockchain, Decentralized Database, Consensus 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]. Blockchain, also known as decentralized ledger technology (DLT), consists of a set of peer-to-peer nodes that do not inherently trust each other. The core concept of blockchain is its consensus algorithm, which protects data against undesired alterations. To have a transaction confirmed, the majority of peers in the network should agree to add it to 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 its strengths and limitations in terms of performance, resource utilization, and data integrity. For adopters, this implies extra effort to evaluate and choose from a set of available solutions. As a result, an efficient and easy-to-use benchmark tool is needed to help decide the best platform based on the requirements. Several performance studies [7]–[10] have been conducted recently to investigate

the scalability, resource consumption, and energy consumption of traditional blockchain technologies such as Bitcoin and Ethereum with Proof of Work.

To evaluate such platforms, performance testing and benchmarking tools, such as Hyperledger Caliper¹, Blockbench [6], Diablo [11], Gromit [12], and BCTMark [13] have been proposed. However, an important limitation of these tools is that they are designed to perform “Batch Workloads”, meaning the entire workload runs until the experiment is completed or if it fails. While this may provide interesting insights, it can be considered largely incompatible with evaluating the system’s long-term scalability and endurance under continuous and variable workloads. Such scenarios may stress the platform differently in ways that are not covered by cross-sectional, where an experiment is observed at a given moment, as opposed to longitudinally, where an experiment is observed over a significant period. BlockCompass implements batch workloads, but also “Transactional Workloads” [14], where a variable number of virtual users send 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 a long-running experiment that tests the platform’s endurance. In addition, requests to a blockchain network usually consist of multiple steps, including submission, hash generation, validation, writing, and confirmation. Consequently, requests are sessionful and are better modeled as transactions. Transactional workloads facilitate this continuous, real-time operation, maintaining the blockchain’s consistency and reliability. Especially in applications like smart contracts, quick feedback, and response are paramount [15], [16].

The main contribution of this paper is BlockCompass itself, a benchmarking framework designed to assess

- Mohammadreza Rasolroveicy is affiliated with IBM Canada in Markham, Ontario, L3R 9Z7 (e-mail: roveicy@ibm.com). ORCID: 0000-0002-7231-7889.
- Wejdene Haouari and Marios Fokaefs are with the Department of Electrical Engineering & Computer Science at York University, Toronto, ON, Canada, M3J 1P3 (e-mail: wejdene@yorku.ca; fokaefs@yorku.ca). Wejdene’s ORCID: 0000-0003-2960-8629. Marios’ ORCID: 0000-0003-3623-5015. (Corresponding author: Mohammadreza Rasolroveicy.).

The manuscript published in the IEEE Transactions on Computers (Volume 73, Issue 8, August 2024)

1. <https://hyperledger.github.io/caliper/>

blockchain platforms under real-world scenarios, specifically those that require continuous operation amidst variable and increasing workloads. The main innovation of BlockCompass compared to other works is the use and experimentation under longitudinal transactional workloads, on top of batch workloads, to expand the capabilities of load testing and benchmarking tools to also allow for detecting saturation problems due to continuous workloads. Besides this, our work also contributes to the following.

- **Blockchain Adapters:** Development of extendable and configurable adapters for Ethereum, Hyperledger Fabric, and Hyperledger Sawtooth, aimed at processing and submitting user requests to the blockchain.
- **Workload Support:** Integration of two distinct workload types within BlockCompass, namely, Batch and Transactional, to accommodate a variety of operational scenarios.
- **Comparative Studies:**
 - An analysis of the performance overhead of different consensus algorithms, including Ethereum’s Clique-PoA (Proof of Authority) and PoW (Proof of Work), and those employed by Hyperledger Fabric and Hyperledger Sawtooth.
 - A comparison between batch and transactional workloads to highlight that batch processing may fail to identify errors resulting from system saturation.
 - An experimental study utilizing a state-of-the-art blockchain benchmarking tool that provides insights from the evaluation of a blockchain network using a batch workload.
- **Usability Study:** Execution of a study involving 33 participants to evaluate the user experience of BlockCompass in comparison with other blockchain benchmarking tools and manual processes, across various blockchain platforms.

2 BACKGROUND

2.1 Benchmarking of Blockchain

The current literature provides a comprehensive insight into the diverse tools available for blockchain benchmarking. Dinh et al.’s Blockbench [6] aims to illustrate several performance metrics such as latency, throughput, and scalability. Similarly, Nasrulin et al. [12] proposed “Gromit,” a benchmarking framework designed for blockchain networks. Focusing on system-level performance metrics, specifically throughput and latency, Gromit’s methodology is characterized by its use of batch transaction workloads. In the conducted experiments, transactions were continuously submitted for two minutes, followed by a one-minute pause to ensure all transactions were finalized. Dong et al. [17] introduced a cross-functional tool named DAGBENCH to measure key performance metrics, including latency, throughput, and resource consumption. The tool operates by dispatching batches of workload per second and then observing the peak values for CPU, memory, network, latency, and throughput.

In a related study, Gramoli et al. [11] proposed “Diablo,” a comprehensive evaluation of the performance of several state-of-the-art blockchains. The core objective of their study

was to emphasize that the performance of these blockchains varies considerably based on the underlying experimental settings. By employing realistic DApps for evaluation, the researchers assessed the blockchains’ proficiency in handling load peaks and committing transactions expediently. Interestingly, the results indicated that many of these blockchains could not effectively manage the demands of the chosen decentralized applications, especially when deployed on contemporary commodity computers globally. Furthermore, the research illustrated that the nature of the workload profoundly influences blockchain performance. Detailed timing data, encompassing average latency values for each blockchain-DApp pairing, was also provided. One significant drawback of this study was the lack of a detailed analysis concerning resource utilization across these blockchain platforms. Such utilization is paramount, affecting associated costs and energy consumption within blockchain frameworks, especially for private blockchains or data centers.

Many tools discussed in the literature primarily focus on handling batch workloads and often extend support to only one or two platforms [11]–[13], [18], [18]–[24]. The major drawback here lies in the workloads’ intermittent nature; every new batch resets the experimental state, eliminating prior data and making it challenging to assess the system’s behavior and performance over cumulative saturation in an extended timeframe. In contrast, transactional workloads, which are designed for endurance and stress testing, maintain a consistent level of system saturation. Continuous and real-time monitoring of performance and resource consumption becomes vital in navigating the complexities of long-term system operations and managing resource and energy consumption effectively [14].

In our previous works, we compared Ethereum PoW, Ethereum PoA, and Hyperledger Fabric [25]. 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 continuous approach highlights the weaknesses batch processes overlook, such as the system’s ability to recover from errors or high loads without a reset. It also demonstrates the cumulative impact of transactional errors and performance degradations that can arise from sustained operations, which are typically masked in batch processing as the system starts afresh with each new batch. Furthermore, in the context of distributed systems, especially blockchains, where state and data integrity are paramount, batch processing fails to simulate real-time transactional challenges, such as network delays, state divergence among nodes, and consensus failures under load.

The Table 1 showcases the comparison between transactional and batch workloads.

In our previous works, we compared Ethereum PoW, Ethereum PoA, and Hyperledger Fabric [25]. The experiments were carried out manually, including blockchain instal-

TABLE 1: Comparison of Batch and Transactional Workloads

Feature	Batch Workload	Transactional Workload
Nature of Workload	Intermittent; starts afresh with each new batch, resetting experimental state.	Continuous; maintains consistent system saturation and stress testing over time.
Performance Monitoring	Limited by the reset of the system with each batch, making cumulative assessment challenging.	Continuous monitoring of performance and resource consumption.
Resource Utilization	Lack of detailed analysis on resource utilization, impacting associated costs and energy consumption.	More effective management of resources and energy due to continuous operation.
System Recovery	System resets mask the ability to recover from errors or handle high loads effectively.	Demonstrates the system's ability to recover from errors and handle high loads without resetting.
Cumulative Impact	Errors and performance degradations reset with each new batch, masking cumulative impact.	Exposes cumulative impact of errors and performance degradations, highlighting operational sustainability.
Realism in Simulation	Fails to simulate real-time challenges like network delays, state divergence, and consensus failures.	Better simulates real-time transactional challenges, enhancing realism in distributed systems like blockchains.

lation, 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.

3 THE BLOCKCOMPASS BENCHMARKING TOOL

In this section, we outline the architecture and design of the BlockCompass tool, including the metrics that need to be recorded, its components, and their implementation for specific platforms, and how the tool can be extended to support experiments on different blockchain platforms.

3.1 Architecture

BlockCompass is designed with three objectives in mind: a) serve longitudinal transactional workloads, b) allow for customizability and extensibility, and c) allow for ease of deployment and of use. The workload generator is specially design to produce such workloads, while the resource monitor can capture longitudinal measurements. These properties are especially critical when performing ensurance or stress tests. BlockCompass follows a highly modular architecture with well-defined interfaces. This allows almost all components to be easily replaceable with completely custom ones and enable different kinds of tests on different kinds of platforms. Finally, the entire deployment is containerized for easy and fast launch of an experiment.

The platform uses a multi-tier architecture illustrated in Figure 1. In its back end, it consists of a workload generator, an adapter for the tested blockchain, and a resource monitor connected to the deployment infrastructure. In the front end, a web interface serves as a dashboard that displays live metrics of the experiment. The dashboard, source code, and documentation are accessible on our GitHub repository for research. The repository can be accessed at

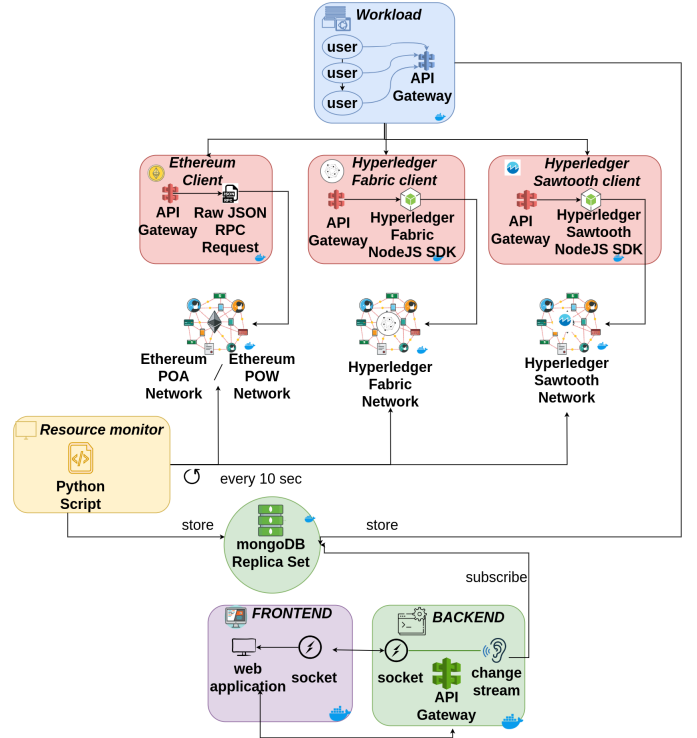


Fig. 1: The BlockCompass architecture

<https://github.com/yorku-ease/blockcompass>. Each component will be detailed next.

Workload generator: To exercise the tested blockchain network, the workload generator simulates concurrent users that send requests of different sizes of data at 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 determines the duration of an experiment. Given that each interval may have a different number of users sending different requests, this creates a continuous and fluctuating workload, allowing us to test the capacity and

endurance of the blockchain network. Figure 2 shows how the workload generator works in conjunction with the rest of the system.

Our workload generator employs two distinct models: **Batch Workload**: In this model, the system is exercised using a workload of fixed size and the experiment lasts until the work is completed. Batch workload is better suited for *load and spike testing*. It is important to note that in batch workloads, the system starts anew with each batch, erasing the previous state, and thereby not offering insights into long-term or cumulative effects.

Transactional Workload: In this model, a number of concurrent and simulated users send requests and wait for responses, which may be either successful or failed. After a predefined “think time” (an artificial delay to mimic the processing of a network response), the user sends the next request. Transactional workloads are better for *endurance and stress testing*. Unlike the batch workload, the transactional model maintains the system’s previous state, increasing workload over time. As a result, long-term users apply more persistent pressure on resource usage, providing a thorough analysis of system performance under prolonged and escalating demands, and revealing potential points of failure or resource constraints.

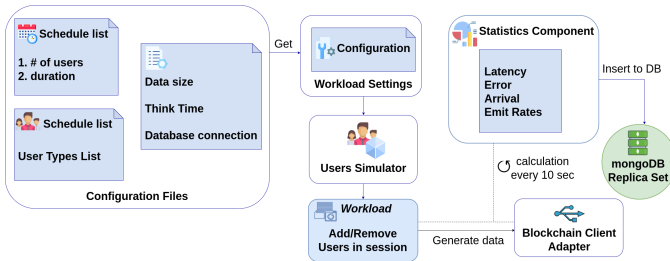


Fig. 2: The architecture of the workload generator

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 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. This allows to reuse components in different experiments and set them up in a modular way.

BlockCompass currently supports 4 different blockchain platforms, including Hyperledger Fabric, Hyperledger Sawtooth, Ethereum with PoA (also known as Clique), and Ethereum with PoW. Each of the platforms needs to have its own client adapter in order to interact with the main blockchain network. It is usually provided by the developers of the blockchain platform itself.

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 is inserted into a MongoDB database as described in Figure 3.

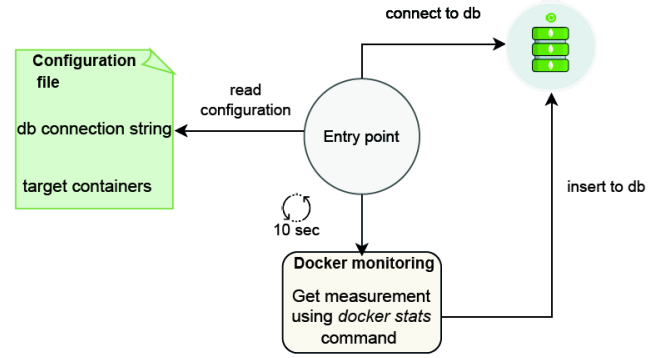


Fig. 3: The architecture of the resource monitor

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** is a Python script that automatically launches all the components needed to run an experiment, namely: the blockchain network, workload, monitor, back-end, and frontend components. All containers are run in the background.

3.2 Performance Metrics Requirements in BlockCompass

BlockCompass integrates both transactional and batch workload capabilities in its benchmarking suite to provide comprehensive evaluations of blockchain systems. However, the emphasis in our experiments is placed on transactional workloads. This focus stems from the pivotal role of transactional workload testing in blockchain evaluation, as it crucially exposes cumulative errors and performance bottlenecks that manifest due to system saturation—nuances that batch workloads can conceal through their state-resetting executions.

By simulating the continuous and rigorous demands of real-world transaction processing, BlockCompass aims to assess the true resilience and reliability of blockchain systems [26]–[28].

BlockCompass collects the following metrics:

- **Emit Rate:** In BlockCompass, the workload generator creates concurrent users in frequent intervals, with each user issuing a new request only after receiving a response to their previous one. The emit rate λ (also referred to *arrival rate* in other systems), measured in transactions per second (tx/s), is calculated as the total number of requests sent by every active user over the sum of all intervals, as follows:

$$\lambda = \frac{\sum_{i=1}^N \sum_{t=1}^T N_t \times R_{it}}{T} \quad (1)$$

where N is the total number of active users (N_t at time t), R_{it} is the number of requests for user i at time t , and T is the duration of the observed period in seconds.

- **Throughput (X):** Throughput is defined as the average rate at which successful transactions (C) are committed to the blockchain, measured as transactions per second (tx/s and calculated as:

$$X = \frac{\sum C}{T} \quad (2)$$

where T represents the total duration of the experiment in seconds, and $\sum C$ is the total number of successful transactions.

- **Error rate (ϵ):** Defined as the average number of error (failed transactions) per total transactions received, 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. Given an emit rate λ and the total number of successful transactions C per second, the error rate ϵ is simply given by:

$$\epsilon = \frac{\lambda - C}{\lambda} \quad (3)$$

- **Latency:** The time it takes for a transaction to be validated and recorded (latency or response time) is a critical metric. Transactional workloads are more suitable for capturing the latency variations macroscopically throughout an extended period of usage. The average latency per transaction is measured as the time in seconds between sending a transaction from the workload generator to the blockchain network and receiving a confirmation response, which is done by an additional worker. 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. For a closed system, the relationship between response time (Rt), throughput (X), and the emit rate (λ) is [29]:

$$Rt = \frac{\lambda T}{X} - Z \quad (4)$$

where Z is the think time of a user when they receive a response and before they send the next request, and T is the duration of the experiment.

- **Resource Utilization:** In traditional batch workloads, data or operations are sent and processed in fixed batches. This method, however, isn't reflective of the intrinsic behavior of blockchain systems where users dispatch a dynamic and often fluctuating number of transactions [26]–[28]. BlockCompass's transactional workload design recognizes and addresses this disparity. Moreover, this granularity allows for a precise measurement of system saturation — pinpointing the exact moment the system struggles to efficiently handle additional transactions. The platform collects utilization metrics for CPU, memory and network directly off the monitoring component of the deployment.
- **Scalability:** The scalability measures the variation in latency, throughput, error and emit rates, and resource consumption as the number of nodes and concurrent users fluctuate. It is a longitudinal metric, which considers changes in both the workload and network infrastructure, where transactional workloads are pivotal, providing a more accurate representation than batch workloads, which might not faithfully depict scalability due to their snapshot-centric approach.
- **Gas Consumption:** In BlockCompass, the design of transactional workloads is strategically optimized to assess gas usage in blockchain systems. As gas represents the dynamic transaction fees, capturing its consumption in real-time is crucial. This feature ensures that users can navigate and strategize with the most accurate representation of operational costs at their fingertips. We employed the BlockCompass workload generator to assess the gas fees as detailed in [30] and [25].

3.3 Extensibility

BlockCompass's primary goal is to facilitate the comparison and analysis of blockchain platforms for the benefit of researchers and practitioners. It allows its users to customize the tool's composition to fit their needs by replacing or extending its modules. To achieve this, BlockCompass is built as a framework following the Open-Closed Principle (OCP), meaning it can easily allow for extensions but 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. This approach makes replacing them with different implementations possible as long as the new implementations adhere to these interfaces and do not affect the flow of data between components.

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:** Deploy the target blockchain. This can be a local private network, a locally emulated public network, or a connection to an existing public blockchain. Given BlockCompass's design around Docker containers [25], [30], deploying in containers is advised for effective monitoring. For public blockchain evaluations, simply modify the target to the public blockchain's receiver node.

- (ii) **Implement the client adapter:** The client adapter acts as an intermediary between the workload generator and the target blockchain. Since blockchains have distinct characteristics, each adapter is tailored. However, the incoming interface, consistent across all, is standardized for the workload generator, ensuring uniformity in data handling.
- (iii) **Implement the Smart Contract:** The smart contract mediates data submissions to the blockchain. For instance, Ethereum's 'kvstore' contract is accessible on our GitHub at `networks/contracts/ethereum`, serving as a key-value storage. Similarly, Hyperledger Fabric's functionality is delivered through 'chaincode', available at `networks/contracts/fabric-v2.2/kvstore`.
- (iv) **Adjust monitor script:** Refine the monitor configuration for the new blockchain. By default, it tracks containers with specific naming patterns from the environment file, which is housed at `monitor/.env`. This pattern should be aligned with the new blockchain's convention.

3.3.2 Extend the Workload Generator

The workload generator is designed to maintain a connection with the client adapter, as previously discussed, utilizing a specific port and API endpoint. Furthermore, it allows for significant flexibility in modifying its internal implementation through a comprehensive set of configuration files. BlockCompass inherently supports different types of users to implement workload mixes and explore even more realistic scenarios.

3.3.3 Extend the Resource Monitor

The current version of BlockCompass is based on Docker containers and as such, it uses a custom monitoring script that scrapes the results of `docker stats` to retrieve metrics for resource consumption. The script then provides 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 class named *Monitor*. This class contains methods to connect to and write to a database, or to 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* 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.

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, performance, and 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, 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. We take the role of a decision-maker that wants to choose between the four in terms of their performance (latency and throughput) and potentially cost. We show how easy it is to set up BlockCompass and execute multiple experiments in a streamlined fashion. All experiments are run using transactional workloads to show the cumulative effect of the workload on the performance of the tested platform.

4.2 Experimental setup

BlockCompass was deployed on an Amazon `t2.large` virtual machine running Ubuntu 18.04 with 8GB of RAM, 2 vCPUs, and 25GB of hard disk space. This VM acted as the host for Docker containers containing the blockchain network's nodes, workload, monitoring, frontend, and back-end components. During testing, each Docker container was allocated only 20% of the VM's total CPU and memory to ensure sufficient resources for all infrastructure elements. All components, including miners, were run as Docker containers within this single virtual machine.

For locally deployed blockchain networks, we utilized specific versions: `geth v1.10.3` for Ethereum, Hyperledger Fabric `v2.3.2` [8], and Hyperledger Sawtooth `v1.2` (Chime) [31]. Each blockchain network comprised five nodes deployed within containers. In Ethereum networks (both PoW and PoA), five miners and one bootnode were deployed. For Hyperledger Fabric, configurations included two organizations, one ordering service, and one channel. The first organization hosted two peers, while the second organization hosted three peers, with a block period set to two seconds. Hyperledger Sawtooth was configured with separate networks for PBFT, Raft, and PoET consensus, each running five validator nodes.

Regarding the workload, workload mixes with different types of users were employed, generating queries ranging from 70 to 80 KB. All experiments were conducted for a fixed duration of 16 minutes, with the number of users ranging variably from 0 to 100.

Given that the adapters were already developed, the only steps necessary to set up the experiments were to deploy and start the blockchain networks and configure the workload generator. Then, using the python launch script, BlockCompass started each experiment.

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 2 shows that Ethereum PoA has the greatest overall throughput. This is due to the fact that Ethereum

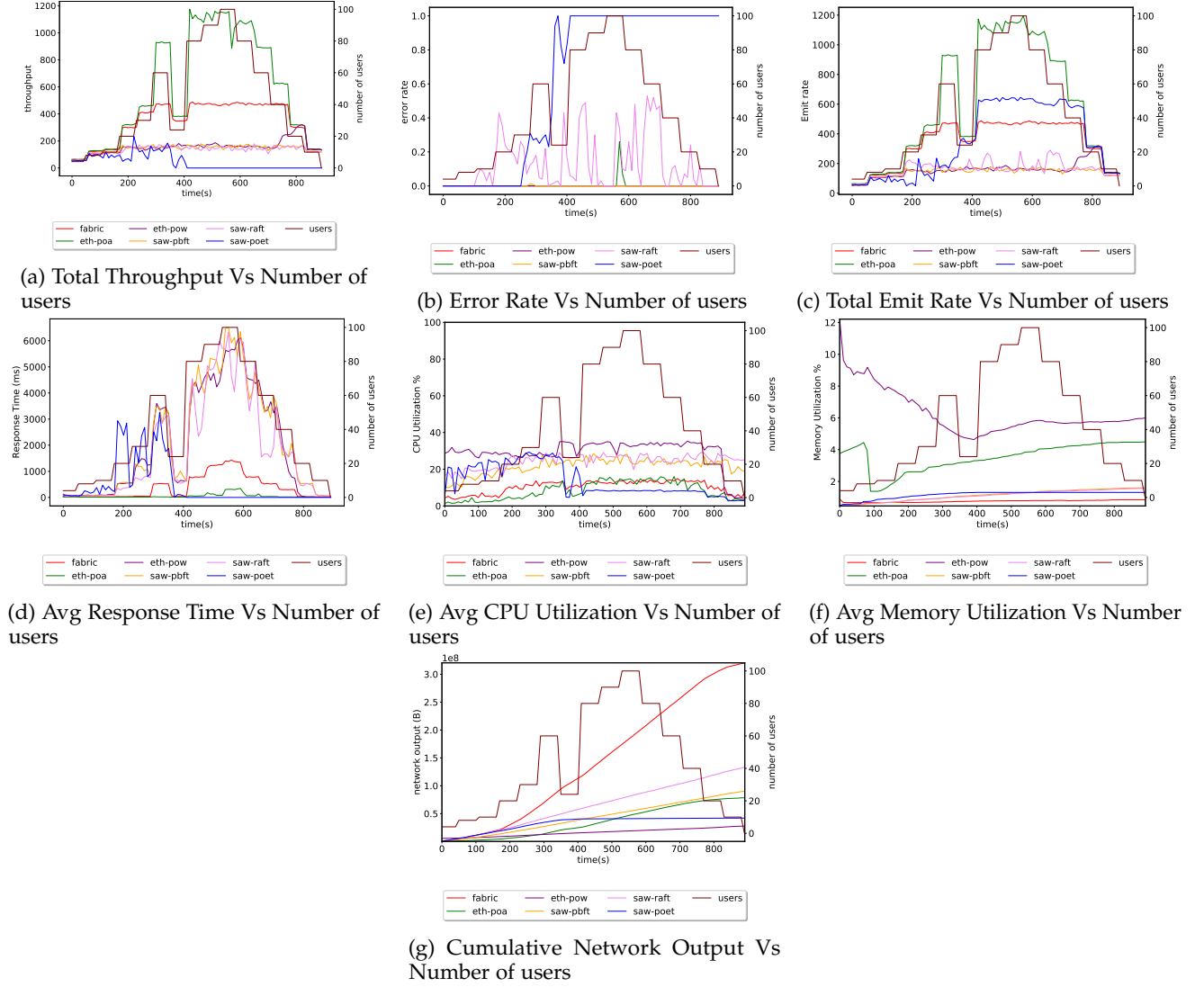


Fig. 4: Number of send requests, Average Response Time, CPU Utilization, Memory Utilisation, and Network throughput using 5 nodes and 1 receiver node

TABLE 2: 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 requests sent	30770	53135	13986	12865	14671	31156
Total requests successfully sent	30770	52743	13986	12865	12175	3844
Avg. Emit (Tx/S)	344.23	590.38	155.4	142.94	162.03	346.17
Avg. Throughput (Tx/S)	344.23	586.03	155.4	142.91	135.27	42.71
Avg Error (%)	0	0.4	0	0.02	13	62
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	50.6	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	67.5	32.65
St.Dev. Network Output	108.14	27.85	6.63	27.49	41.87	12.9

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 with Raft

consensus protocol has a lower throughput, despite being implemented in both Hyperledger Fabric with Raft consensus algorithm. Additionally, 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 a correlation between workload and response time in terms of the number of users in all the blockchain networks, as expected. Additionally, based on Figure 4d and Table 2, we note that Ethereum PoW and Sawtooth PBFT have the highest response times among all platforms, which are 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 the 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. We assume that the architecture and implementation of the blockchain platform for both platforms are different. For example, Hyperledger Fabric consists of different network peer components including Endorser, Anchor, and Orderer peers, with only the Orderer peer responsible for creating a block and reaching consensus based on the Raft consensus protocol. Therefore, for communications, only Orderer peers are involved. Hyperledger Sawtooth consists of multiple validators, all of which are responsible for adding a new block, necessitating communication among them to reach a consensus. Moreover, in Sawtooth, the Raft algorithm has been extended 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 shown in Figure 4e and Table 2, 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 balanced CPU usage throughout the experiment. Ethereum PoA is the most efficient algorithm in terms of CPU consumption, as it is only based on the 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 require significant work during the validation of blocks, while the third, Sawtooth in general, is the one that requires a higher workload, which motivated the use of PoET consensus.

As shown in Figure 4f, memory is not a significant resource 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 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 have conducted similar studies comparing different blockchain platforms [32] or different consensus algorithms [33]. Without BlockCompass, we had to go through several steps: installing all the dependencies for each platform individually, setting up environments, checking for the latest packages and potential conflicts, installing each blockchain platform individually, designing a custom closed workload for continuous and fluctuating users with varying workload sizes, writing a custom monitoring script to capture performance and resource usage, scripting the export of results, and finally generating figures and statistical analyses. The installation, configuration, and execution of experiments alone (excluding the final analysis) could take longer due to inadequate documentation and information, especially when conducted by a single researcher. With BlockCompass, it is possible to fully automate or at least systematize some of these steps. The BlockCompass dashboard provides visualizations of the blockchain platform's performance metrics over time, allowing researchers to observe any live anomalies or performance bottlenecks. Additionally, researchers can easily and rapidly change workload and other configurations. Live monitoring facilitates faster decision-making regarding which configuration is suitable compared to waiting for the experiment to finish and exporting and analyzing results.

5 BLOCKCOMPASS VALIDATION: COMPARISON OF BATCH VS TRANSACTIONAL WORKLOADS

In this section, we present a validation experiment to evaluate certain aspects and design choices around the proposed platform. In this experiment, we show an important difference between batch and transactional workloads, which is the need to capture the cumulative effect of a longitudinal workload, to support our claim that longitudinal workloads are important for blockchain benchmarking platforms.

We show the differences between batch and transactional workloads by running two versions of a similar experiment using the Ethereum Proof-of-Authority (PoA). According to

the experiment, the blockchain under test is exercised using a fluctuating number of users (from 100 to 150). In the first version of the experiment, we used a transactional workload and exercised the blockchain platform for 20 continuous minutes (results shown in Table 3). In the second version of the experiment, we ran 20 batches, 5 for each number of user (results shown in Table 4). The experiments are designed identically with the key difference that in batch workloads, the system resets for every new batch, which means that memory and all other resources are wiped and the infrastructure restarts.

As shown in Table 3 for the transaction workload, in the initial stage with 100 users, the system performed smoothly. However, as the system underwent stress, errors and CPU usage noticeably increased during a 3-minute interval. With 120 users, a continued rise in errors and a reduction in throughput were noted. Additionally, when the user number reached 150, the emission rate fell. While 1500 was expected at the 12-minute mark, the actual rate was 1050. This suggests that the server, set up with 8 GB RAM and 2 vCPUs, was at its full capacity, unable to support more users because we used a closed workload, where we waited for responses to calculate.

During the experiment, the maximum emission rate was recorded for 1200 users. This indicates that the current server setup could only maintain this number of users during a longer stress period where user requests were continually sent. At the 13-minute mark, where the system was overloaded with requests and users, it crashed, recovering after 2 minutes, and resumed at the 16-minute mark with a lighter load of 110 users.

In contrast, Table 4 highlights the spike test scenario, demonstrating experimental results for between 100 and 150 users. Unlike the continuous stress test, each batch in this experiment was transmitted, and the system was allowed to process it; only after retrieving the results was the subsequent batch sent. A crucial attribute of this batch spike test is the system's resetting and restarting between batches; for instance, data for 150 users would be transmitted, results obtained, and the system then shut down before initiating the next batch. Throughout the batch workload trials, we generally observed better results. However, in the 150-user scenario, errors were significantly elevated, ranging from 346 to 287, indicating the system's inability to efficiently manage this user volume in a singular batch, particularly as the average CPU usage also escalated to 24%.

6 USABILITY STUDY

To evaluate the usability of BlockCompass, our newly developed tool, we compared it against Blockbench, a popular blockchain benchmarking tool from 2021. Capturing genuine user experiences in the world of blockchain is vital, and our study aims to provide these insights.

After securing ethical clearance from Polytechnique Montreal², we reached out to both graduate and undergraduate software engineering students at Polytechnique Montreal. This group included students enrolled in the "Software

TABLE 3: Experimental results for transactional workload in Ethereum PoA representing Time (min), Response Time, Throughput, Emit Rate, Errors and CPU (%)

Time	Users	RT (ms)	Throughput	Emit Rate	Error	CPU
1	100	50	950	1000	0	18
2	100	77	1000	1000	0	21
3	100	109	1000	1000	0	22
4	100	146	928	1000	72	20
5	100	200	884	1000	116	23
6	120	218	1200	1200	0	22
7	120	250	1124	1200	76	21
8	120	221	1091	1200	109	25
9	120	219	830	1100	270	25
10	120	248	511	900	389	26
11	150	380	791	1050	259	25
12	150	317	421	1000	570	24
13	150	314	459	680	211	23
14	150	0	0	0	0	0
15	150	0	0	0	0	0
16	110	89	1100	1100	0	19
17	110	101	1100	1100	0	21
18	110	123	1090	1100	10	22
19	110	146	1067	1100	33	21
20	110	142	1039	1100	61	24

TABLE 4: Experimental results for batch workload in Ethereum PoA representing Time (min), Average Response Time(RT), Throughput (Tps), Emit Rate (Tps), Errors and Average CPU (%)

Batch #	Users	RT	Throughput	Emit Rate	Error	CPU
Batch 1	100	41	992	1000	8	18
Batch 2	100	56	1000	1000	0	20
Batch 3	100	69	1000	1000	0	21
Batch 4	100	51	1000	1000	0	19
Batch 5	100	44	1000	1000	0	20
Batch 1	120	210	1165	1200	35	22
Batch 2	120	209	1200	1200	0	22
Batch 3	120	189	1200	1200	0	23
Batch 4	120	275	1183	1200	17	21
Batch 5	120	188	1200	1200	0	20
Batch 1	150	321	1154	1500	346	23
Batch 2	150	367	1243	1500	257	24
Batch 3	150	301	1207	1500	293	25
Batch 4	150	315	1177	1500	323	23
Batch 5	150	346	1213	1500	287	24
Batch 1	110	103	1100	1100	0	20
Batch 2	110	107	1100	1100	0	21
Batch 3	110	98	1100	1100	0	19
Batch 4	110	112	1100	1100	0	20
Batch 5	110	103	1100	1100	0	20

Architecture" course, available to both undergraduate and graduate students. Participation was entirely voluntary, leading to the formation of distinct groups based on their tool preference.

To provide students with an unbiased introduction to both tools, the teaching assistant (TA) prepared and recorded detailed demos covering the installation processes for both BlockCompass and Blockbench. These instructional materials, along with related documents, were shared with the students. To ensure fairness, these resources were distributed by a third party, who was not part of the survey process.

It's important to note that one of our co-authors, who also served as a teaching assistant (TA), is associated with BlockCompass. To avoid potential biases, this affiliation was not disclosed to participants. Since Blockbench lacked

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

official documentation, the TA provided additional guidance, ensuring both tools had equivalent instructional support.

In terms of research transparency, our collected data is available for review here: ³. The assignment, which was part of the students' academic course in autumn 2021, was managed by their professor, ensuring the TA did not influence grading or feedback.

Feedback was gathered in January 2022 after the students received their final grades in December 2021. To ensure the confidentiality and integrity of the feedback, students were required to send consent forms separately to a co-author who was neither the TA nor involved in the course. This ensured their participation was anonymous and free from any potential biases.

We followed a between-subjects study approach [34] as it was particularly suitable given that the study was embedded within an assignment. This approach allowed participants to delve deeply into one platform, enhancing their domain knowledge. Additionally, configuring just one tool streamlined the process, offering greater convenience to the participants. Given the set deadline, focusing on a single platform also made the study more concise and manageable.

Figure 5 presents a flowchart detailing the participation in the usability study. Of the 33 students originally enrolled in the course, all were invited to join the study, with 26 (79%) agreeing to participate. Importantly, no enforcement or persuasion was applied to guide their choice of platform. Within this group, 15 (58%) opted for BlockCompass while 11 (42%) chose Blockbench. In terms of blockchain platforms, 14 (54%) utilized Fabric, with 8 of those using BlockCompass and 6 using Blockbench. Conversely, 12 participants (46%) employed Ethereum, 7 of whom were on BlockCompass and 5 on Blockbench.

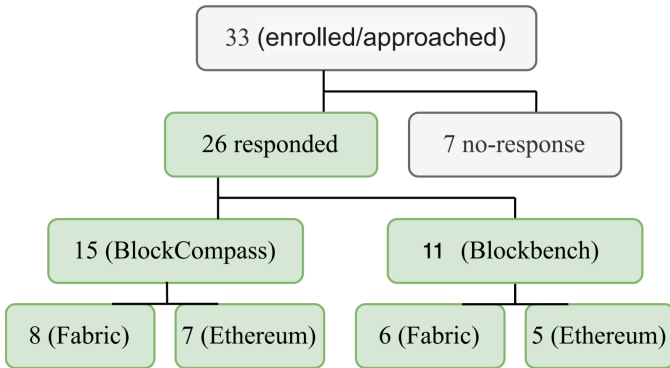


Fig. 5: Flowchart on the participation in the usability study

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 obtaining the output, which may include data files or visualizations. For each of these phases, we inquired 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 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 with 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* [35] and the Wilcoxon Mann-Whitney *U*-tests [36] 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 [37]. Moreover, as we are conducting multiple comparisons (i.e., for multiple questions), we applied a Bonferroni correction [38], 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 5 shows the average responses for all questions between BlockCompass and Blockbench. As 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 a 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.

7 THREATS TO VALIDITY AND LIMITATIONS

With regard to the applicability of BlockCompass on various blockchain platforms, one potential threat to the external validity of our work is the generalizability of the benchmarking platform to other networks. However, this concern is alleviated by the extensibility of BlockCompass, which allows for the definition of other network adapters. Additionally, while the results of the case study may face a similar threat, its aim is to compare the studied networks and consensus protocols rather than identify the best option. Finally, our user study might lack generalizability, particularly since

3. Result of the survey: <https://tinyurl.com/yc5nxd3>

TABLE 5: 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 min 30 s	40 min 16 s	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 min 52 s	38 min 30 s	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 min 8 s	53 min 6 s	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 min 52 s	44 min 54 s	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

it involved a small group of the intended user base. This limitation could be addressed in the future with a broader study involving professionals and researchers with diverse backgrounds and levels of experience.

Concerning internal validity, a potential threat is selection bias. In the case study, the selection of blockchain networks and consensus protocols may have been biased, but it encompassed several popular systems, including both public and private networks, and a variety of consensus algorithms. Additionally, some networks were chosen to showcase specific behaviors relevant from a performance perspective. In the user study, selection bias was mitigated by inviting a large number of participants and then selecting those who met predefined exclusion and inclusion criteria applicable to the study.

The final threat relates to the construct validity of our study. Firstly, confounding variables in the case study, such as cloud variability or network instability, were mitigated through repetitive experimentation to average out their effects. In the user study, confounding variables like age or sex were not examined as they were deemed unrelated to the study objectives. However, additional analysis may be necessary to investigate these factors. An important consideration is the potential for hypothesis guessing or researcher expectations influencing the results. Participants may have speculated which of the two tools was developed by the research team. To minimize this threat, equal documentation and support were provided for both platforms to all

participants, aiming to ensure a fair comparison. Ultimately, the impact of any such biases may have been minimal based on our findings, particularly concerning the quantitative metrics of comparison.

8 CONCLUSION

In this paper, we presented BlockCompass, a novel benchmarking tool for blockchain platforms and their configurations. BlockCompass is highly configurable and extensible, allowing for extensive performance testing, including stress and endurance testing for blockchains. We demonstrated the ease of use of BlockCompass and its capabilities in the context of an empirical study comparing three distinct blockchain platforms: Ethereum, Hyperledger Fabric, and Hyperledger Sawtooth, each employing distinct consensus protocols. To evaluate the usability of BlockCompass, we conducted a user study comparing our platform with Blockbench, an existing benchmarking tool for blockchain. Overall, participants in the study expressed significantly greater satisfaction with BlockCompass regarding their experience with running benchmark tools, the clarity of the documentation, the presentation of final results, and the ease of performing experiments compared to manual efforts or other tools. In general, BlockCompass outperformed Blockbench in terms of the time taken to complete experiments, 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 *Services Computing-SCC 2020: 17th International Conference, Held as Part of the Services Conference Federation, SCF 2020, Honolulu, HI, USA, September 18–20, 2020, Proceedings*, vol. 17, pp. 120–134, Springer International Publishing, 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] V. Gramoli, R. Guerraoui, A. Lebedev, C. Natoli, and G. Voron, "Diablo: A benchmark suite for blockchains," in *Proceedings of the Eighteenth European Conference on Computer Systems*, pp. 540–556, 2023.
- [12] B. Nasrulin, M. De Vos, G. Ishmaev, and J. Pouwelse, "Gromit: Benchmarking the performance and scalability of blockchain systems," in *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pp. 56–63, IEEE, 2022.
- [13] 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.
- [14] S. Kounev, K.-D. Lange, and J. von Kistowski, *Systems Benchmarking: For Scientists and Engineers*, vol. 1 of *Computer Science*. Springer International Publishing, 2020. ISBN: 978-3-030-41707-9.
- [15] R. K. Abbott and H. Garcia-Molina, "Scheduling real-time transactions: A performance evaluation," *ACM Transactions on Database Systems (TODS)*, vol. 17, no. 3, pp. 513–560, 1992.
- [16] P.-Å. Larson, A. Birka, E. N. Hanson, W. Huang, M. Nowakiewicz, and V. Papadimos, "Real-time analytical processing with sql server," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1740–1751, 2015.
- [17] Z. Dong, E. Zheng, Y. Choon, and A. Y. Zomaya, "Dagbench: A performance evaluation framework for dag distributed ledgers," in *2019 IEEE 12th international conference on cloud computing (CLOUD)*, pp. 264–271, IEEE, 2019.
- [18] D. Gupta, L. Perronne, and S. Bouchenak, "Bft-bench: Towards a practical evaluation of robustness and effectiveness of bft protocols," in *Distributed Applications and Interoperable Systems: 16th IFIP WG 6.1 International Conference, DAIS 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, vol. 16, pp. 115–128, Springer International Publishing, 2016.
- [19] M. Di Pierro, "What is the blockchain?," *Computing in Science & Engineering*, vol. 19, no. 5, pp. 92–95, 2017.
- [20] 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.
- [21] 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.
- [22] 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.
- [23] G. Shapiro, C. Natoli, and V. Gramoli, "The performance of byzantine fault tolerant blockchains," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8, IEEE, 2020.
- [24] TAPE, "TAPE, 2023." <https://github.com/Hyperledger-TWGC/tape>, 2023. Accessed: 2023-08-01.
- [25] 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.
- [26] M. Andreas and W. Gavin, "Mastering ethereum," *O'Reilly*, 2018.
- [27] M. Pacheco, G. Oliva, G. K. Rajbahadur, and A. Hassan, "Is my transaction done yet? an empirical study of transaction processing times in the ethereum blockchain platform," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, pp. 1–46, 2023.
- [28] S. Gupta and M. Sadoghi, "Blockchain transaction processing," *arXiv preprint arXiv:2107.11592*, 2021.
- [29] J. D. Little and S. C. Graves, "Little's law," *Building intuition: insights from basic operations management models and principles*, pp. 81–100, 2008.
- [30] M. Rasolroveicy and M. Fokaefs, "Performance and cost evaluation of public blockchain: An nft marketplace case study," in *2022 4th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pp. 79–86, IEEE, 2022.
- [31] H. Sawtooth, "Hyperledger sawtooth 1.2 (chime)." <https://sawtooth.hyperledger.org/release/chime/>, 2019. Accessed: 2022-01-10.
- [32] 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.
- [33] 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.
- [34] I. S. MacKenzie, *Human-Computer Interaction: An Empirical Research Perspective*. Cambridge, MA, USA: Elsevier Science, 2012.
- [35] 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, pp. 688–690, 05 2006.
- [36] P. E. McKnight and J. Najab, "Mann-whitney u test," *The Corsini Encyclopedia of Psychology*, pp. 1–1, January 30 2010.
- [37] P. Royston, "Approximating the shapiro-wilk w-test for non-normality," *Statistics and computing*, vol. 2, no. 3, pp. 117–119, 1992.
- [38] E. W. Weisstein, "Bonferroni correction," *MathWorld—A Wolfram Web Resource*, 2004. Accessed: 28 May 2024.



Mohammadreza Rasolroveicy works in DevOps team at IBM Data Security, applying his expertise to enhance system efficiency and integrity within the tech industry. Previously, he was a postdoctoral fellow at York University in Toronto, Canada, and holds a Ph.D. in Computer Engineering from Polytechnique Montreal, with research interests spanning Software Quality, Clouds Blockchain Performance, and Self-adaptive Systems.



Wejdene Haouari is a MSc student in the Department of Electrical Engineering & Computer Science at York 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, machine learning, 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.