DIABLO: A Distributed Analytical Blockchain Benchmark Framework Focusing on Real-World Workloads

Harold Benoit
EPFL

Vincent Gramoli
University of Sydney
EPFL

Rachid Guerraoui
EPFL

Christopher Natoli*
University of Sydney

Abstract

The plethora of blockchain proposals raises the question of selecting the ideal blockchain for a given decentralised application (DApp). Most blockchain performance evaluations are obtained through tailored environments under unknown experimental settings documented through whitepapers or blog posts. Unfortunately, the few blockchain benchmark efforts typically generate workloads either from a central location, or, purely based on synthetic request patterns.

We propose a benchmark framework, DIABLO, to reason about the performance of blockchains under realistic workloads. DIABLO is extensible, allowing DApp programmers to define new workloads, and blockchain designers to integrate and test their own blockchain. DIABLO currently supports 6 blockchain implementations and 4 built-in workload DApps. Our results indicate that blockchains with lower fault tolerance can achieve higher performance, confirming that speculative execution of transactions might drastically impact performance. These results also show that one implementation can offer significantly better performance than another implementation of the same blockchain protocol.

*The authors are listed in the alphabetical order of their lastname. Christopher Natoli is the primary author of this publication.

1 Introduction

With the growing adoption of blockchain technology, the number of readily-available solutions have multiplied dramatically. As of March 2021, approximately five thousand distinct cryptocurrencies have been reported on a single website [3]. Each of these implementations aim at offering improvements through distinctive features, focused on the performance and application to various use-cases. Although a number of these variants could, in theory, be running on multiple instances of the same blockchain, they are often packaged as their own standalone implementation with distinctive features. A recent survey [12] highlights the breadth of the blockchain landscape through classifications of blockchains, listing 8 different protocols to select nodes that are tasked with proposing blocks, 13 different consensus protocols and 9 data structures to store transaction information. This diversity illustrates a probably small subset of all possible blockchain implementations that exist today.

This plethora of blockchain proposals raises the question of which proposal is the ideal blockchain for a particular application. Unfortunately, most of these proposals are not reported in scientific publications. They are at best presented in the form of white papers that present a 10-000-foot-view of their implementation details. As an example, the Ethereum yellow paper [16] presents the technicalities of the Ethereum Virtual Machine but does not explain how Ethereum participants can reach consensus on a unique block at a given index of the

DApp	Trace	Feature
Microblogging	Twitter	load burst
Exchange	Nasdaq	herd behavior
Supply chain	Aviation	low demand
Constant	Synthetic	tunable

Table 1: The Decentralized application (DApps) of DIABLO and their features

Blockchain	Consensus	Fault tolerance
Go Ethereum	PoA Clique	Byzantine
Open Ethereum	PoA Aura	Byzantine
CollaChain	DBFT	Byzantine
Quorum	IBFT	Byzantine
Quorum	Raft	Crash
Hyperledger Fabric	Raft	Crash

Table 2: The blockchains evaluated with DIABLO

chain. In order to analyse the underlying protocols of such blockchains, researchers typically had to look at the available source code before being able to reason about the correctness of the protocols [6]. Another approach is for researchers to evaluate blockchain as black boxes by generating workloads and measuring their performance. In comparison with the variety of blockchain proposals, very little evaluations have been made on blockchain and most of these evaluations [5, 2, 11, 9, 13, 4] evaluate few blockchains on synthetic workloads that are not representative of realistic use-cases.

In this paper, we propose the *DIstributed Analytical BLOckchain benchmark framework* called DIABLO, to reason about the performance of blockchain under realistic workloads. DIABLO is extensible in that it allows users to defined their own *Decentralised Application (DApp)* and offers built-in workloads as depicted in Table 1, including (i) Microblogging, featuring bursts of simple requests, (ii) Exchange, featuring herd behaviors in financial markets and (iii) Supply chain, featuring a Poisson distribution of requests, and a (iii) Constant, featuring a tunable fixed rate of requests. DIABLO comes with various blockchains indicated in Table 2 including Go Ethereum or Geth, Open Ethereum (formerly known as parity), CollaChain a new blockchain combining Red Belly Blockchain optimisations [4] with Ethereum smart contracts, Quorum, the blockchain originally designed by J.P. Morgan and maintained by Consensys, and Hyperledger Fabric [1] hosted by the Linux Foundation.

We show empirically that (i) tolerating Byzantine faults is typically more costly than tolerating exclusively crash faults, (ii) OpenEthereum is consistently slower than Go Ethereum and (iii) Hyperledger Fabric (HLF) suffers more from contention than blockchains based on traditional (non-speculative) executions.

The rest of the paper is organised as follows. In Section 2 we present DIABLO. In Section 3 we present the experimental evaluation of the existing blockchains. In Section 4 we present the related work and we conclude in Section 5.

2 Diablo

This section introduces DIABLO, the *DIstributed Analytical BLOckchain benchmark framework*, that allows to compare different blockchains on the same ground under realistic and tunable workloads. DIABLO is extensible in that it allows users to define more workloads and plug new blockchains easily.

Figure 1 provides a visual representation of the components of DIABLO. DIABLO includes two main elements, the *Primary* (Section 2.1), tasked at orchestrating the workload and performing benchmark flow, and the *Secondary* (Section 2.2), tasked at interacting with the blockchains and collecting benchmark statistics.

2.1 DIABLO Primary

The Primary is the conductor machine of the benchmark, orchestrating the timing and operations. Its main function is to generate and distribute the workload data to the secondary machines, sending commands to start benchmarking, and once complete,

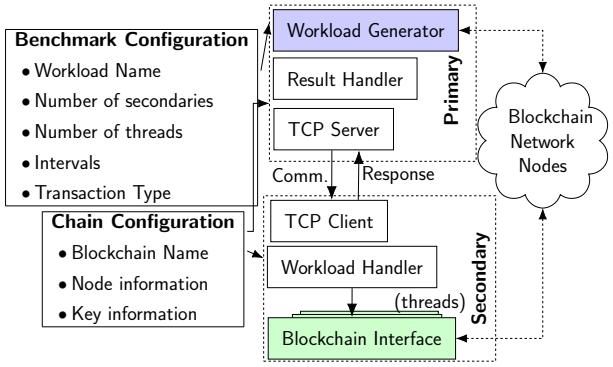

Figure 1: DIABLO Architecture.

retrieving the results. The Primary hosts the communication server for the secondaries to connect to, communicating over TCP, sending commands and retrieving the responses that indicate success or failure of the operation.

Workload Generator. Each blockchain integration requires the implementation of a workload generator. The sole responsibility of the generator is to parse information from the benchmark configuration, such as transaction information and number of total transactions to create, and generate data relating to the workload that is usable by the blockchain system. This generation phase also initialises related smart contracts by deploying them to the chain if required. Often, this generator interacts directly with the blockchain under test by requesting information that is used through for workload generation, such as account information, chain-related transaction information and any other key details that are required for workload generation.

Result Handler. Once all DIABLO Secondary machines have signalled completion of the benchmark, the Primary will request to retrieve all the results. At this point, all results are passed to the result handler, which aggregates and formats outputs to usable information and the related JSON files. The result handler not only provides a central point to collect, aggregate and perform calculations on all information, but also standardises the way information is represented across all chains, so as to facilitate the comparison between blockchain systems. The obtained results indicate both the performance of the whole system under consideration and the performance of each DIABLO Secondary. This fine-grained information about per-machine transaction latency and throughput is typically instrumental in identifying slow machines impacting the overall performance.

2.2 DIABLO Secondary

The Secondaries are responsible for communicating directly with the blockchain under test through a blockchain client interface. With the ability to have a number of worker threads running in parallel, it allows further concurrent sending of transactions from the source. Each Secondary connects to the DIABLO Primary, awaiting the appropriate commands and information to start the benchmark execution. Upon initialisation and connection, each secondary is given a unique identifier, denoting which blockchain node it connects to for information, and how it will be distinguished. Each of the Secondary's threads is a separate instance, connecting to the blockchain under test and interacting in its own work loop. It comprises of two major components, the *Workload Handler*, responsible for directing the flow of the benchmark, and the *client interface* used to connect to the blockchain.

Workload Handler. The workload handler manages the distribution of workload information to the worker threads running on the secondary machine. The handler is responsible for initialising the worker threads, setting up the workload schedule, and operating the worker loop that performs the benchmark. The workload handler also format the results.

Client Interface. For the sake of modularity and similar to the *workload generator* on the Primary, each blockchain integration requires the implementation of an interface, allowing DIABLO to support basic operations and interact with the blockchain under

test. The client interface is key to the modularity of DIABLO, it integrates the different functionalities abstracted from the benchmark flow, making it possible to perform various actions, like connecting to a node, sending a transaction, etc. Another task handled by the client interface is to obtain the results, throughput and latency, and return these results when asked upon by the workload handler, for these results to be then forwarded to the Primary.

```
bench:
  type: "simple"
  txs:
    0: 1
    10: 11
    20: 11
    28: 6
```

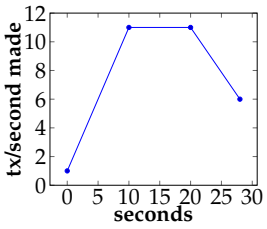

Figure 2: Benchmark Configuration Example

2.3 Configurations

Defining a benchmark consists of two things: (i) defining the workload as patterns of request intensity and timing and (ii) the experimental setup, denoting the configuration of machines to make the network of the blockchain. For simplicity, DIABLO provides two configuration files specifying each of these characteristics making it easy to record and measure precisely, the *benchmark configuration file* and the *chain configuration file*.

2.3.1 Benchmark

The benchmark configuration file is fundamental in the operation of DIABLO, defining the workload and all key aspects of how the benchmark runs. It contains general information about the benchmark, defining the distribution of DIABLO Secondaries and the number of worker threads, but also the pattern of transactions. To define a workload curve, the benchmark has a time-series representation of the transaction request intensity, showing the change of request intensity over time. As depicted in Figure 2, the transaction curve is defined in per-second granularity and allows a dynamic intensity of requests. This gives rise to importing real-world request curves obtained through time-series datasets observed and captured in other systems, or programmatically designed.

The benchmark configuration also details transaction information, containing the function calls, data types and variants of what data is passed into the transaction. If further complexity is required, the benchmark configuration also denotes a *premade workload*, where all transaction information and distribution of requests is preconfigured, allowing the work and intensity per secondary to be heterogeneous.

2.3.2 Chain

The chain configuration denotes the location of the blockchain nodes to contact from the benchmark. This configuration is the primary means of understanding how the blockchain is able to connect, and defines the account information for the transactions to be generated for. The unique aspect of this configuration, is that the modular design allows *extra data* to be defined in a way that can be interpreted directly by each blockchain integration. If a specific blockchain requires additional key material, or specific configuration requirements, then they can be defined as part of the chain configuration.

3 Experiments

This section presents the results from running DIABLO against various blockchain systems as depicted in Table 2. We intentionally selected blockchain offering different guarantees, we classified them in three groups: the CFT ones that tolerate only crash failures, the BFT ones that can also tolerate $f < n/3$ byzantine failures and the PoA (or proof-of-authority) ones that can tolerate some byzantine failures as long as all messages delays take less than a known and bounded amount of time [6] (i.e., these latter blockchains assume *synchrony*).

```
modifier checklen(string memory data) {
  require(bytes(data).length <= maxlen,
      "tweet too large");
  _;
}
function tweet(string memory data)
  public checklen(data) {
    tweets[msg.sender] = data;
    emit NewTweet(msg.sender, data);
}
```

Twitter code snippet in solidity for EVM-based blockchains

```
func (s *SmartContract) Tweet(
  ctx contractapi.TransactionContextInterface,
  message string, owner string) error {
    if len(message) > MaxLen {
     return fmt.Errorf("tweet too large, %v",
       len(message))
  }
  b := []byte(message)
  return ctx.GetStub().PutState(owner, b)
}
```

Twitter code snippet in golang for Hyperledger Fabric

Figure 3: The DIABLO Twitter DApp is written as a smart contract and a chain code

3.1 Setup

We perform all experiments on a local OpenStack cluster, where all virtual machines are connected through a virtual network over 4 host machines. We utilise "m1.xlarge" instances, having 8 vCPUs and 16 GB Memory for distributed experiments, and "c5.2XLarge" having 8 vCPUS, 16GB RAM, and a 700GB disk for experiments running local experiments.

- **PoA blockchains**: A set of predefined validators seal each block one at a time in a round robin fashion. We set the period between blocks to 1 second to match the parameters of other blockchains. DIABLO interacts with these blockchains through Websocket JSON RPC, subscribes to new block events and confirms the existence of a transaction when receiving the block header.

- **Quorum**: Both for Quorum-IBFT and Quorum-Raft, the block period is set to 1 second. DIABLO interacts with these blockchains through Websocket JSON RPC, subscribes to new block events and confirms the existence of a transaction when receiving the block header.

- **CollaChain**: Similar to Red Belly Blockchain [4], CollaChain appends new blocks only when needed (based on a timeout and the number of transactions in the mempool) so the period between blocks varies based on the load. DIABLO interacts with CollaChain through Websocket JSON RPC, subscribes to new block events and confirms the existence of a transaction when receiving the block header.

- **Hyperledger Fabric**: DIABLO interacts with Hyperledger Fabric v2.2 LTS through http requests. New blocks are created after a default timetout of 2 seconds or a threshold of 500 transactions per block. Hyperledger's Golang API supports the functionality for transaction confirmation, so all new commits are observed through a channel linked to the transaction event.

The Microblogging DApp consists of a DIABLO configuration that follows the request pattern of Twitter[1], except that we lowered the average throughput of 5700 transactions per second to 3% or 171 transactions per second with a one second peak of 4296. Its code is written as a smart contract for EVM-based blockchain and a chain code for Hyperledger Fabric whose code snippets are given in Figure 3. The Exchange DApp consists of a DIABLO configuration (Section 2.3) that follows the request pattern of the trace of observed trades on the Nasdaq stock exchange. Similarly to the MicroBlogging Dapp, its code is also written as a smart contract and chain code. The Synthetic application consists of a simple fixed sending rate of 250 basic transactions per second for 150 seconds, also written in smart contract and chain code. Each plotted value of the curves results from the average over at least 5 runs, the shade

[1]This experiment mimics the phenomemon experienced during the release of Castle In the Sky: https://blog.twitter.com/engineering/en_us/a/2013/new-tweets-per-second-record-and-how.html

around curves indicate the minimum and maximum values.

3.2 Overall Performance Comparison

In this section, we present the throughput and latency of all the built-in blockchains. First, we depict their throughput as a time series on different applications. Then, we describe their latency as a Cumulative Distribution Function (CDF) on the same workloads. As mentioned before, we compare blockchains depending on their guarantees: CFT for the ones that tolerate crash failures, BFT for the ones that tolerate Byzantine failures and PoA for the ones that tolerate Byzantine failure only in synchronous networks.

Throughput. Figure 4 depicts the throughput of all blockchains—computed over a 1-second sliding window—as time elapses when running the Exchange, the Microblogging, the Synthetic applications. Each column includes the results obtained with a different application while each row includes the results for each group of blockchains (CFT, BFT, PoA).

The first observation is that all blockchains offer a varying throughput under the exchange application, a burst throughput under the Microblogging application and a constant throughput under the Synthetic application. This follows from the nature of the rate at which these applications send requests to the blockchain systems: varying rate for Nasdaq, in burst for Twitter and constant for Tunable. Note that the request rate of each application is represented as a purple dashed line on each of these graphs, which is followed closely by the throughput of most blockchain systems.

We observe that in all applications, the throughput of PoA-geth (Go Ethereum) is higher than the throughput of PoA-openethereum (Open Ethereum), this indicates that Go Ethereum performs generally better. Note that we will confirm this observation by investigating their respective latency later. Another interesting observation is that some blockchains start losing requests when under high load, this is the case of Quorum-IBFT when running the Microblogging application: we can see that the throughput drops to zero and we confirmed that some of the requests were lost as we will discuss later. This is seemingly due to the contention induced by the burst of requests generated by the Twitter workload, which exceeds the capacity of Quorum-IBFT. Finally, Quorum-Raft and Hyperledger Fabric seems to offer comparable results, even if Quorum-Raft is faster at times. We will see later that this depends on the experimental settings and cannot be generalized to all settings.

Latency. Figure 5 depicts the CDF latencies of all blockchains when running the Exchange, the Microblogging, the Synthetic DApps. As before, each column includes the results obtained with a different application while each row includes the results for each group of blockchains (CFT, BFT, PoA).

We observe, at first glance, a large variation of transaction latencies across most settings as it varies from seconds to minutes. In particular, we can see that the PoA blockchains have in general a larger latency than the CFT and BFT blockchains. We also note that PoA-openethereum and Quorum-IBFT loses some requests which explains why their CDF never reaches 100%. In particular, the graphs in the Twitter column shows that PoA-openethereum, PoA-geth and Quorum-IBFT loses requests due to contention bursts, which confirms the reason why the throughput of Quorum-IBFT drops rapidly as well in Figure 4. By contrast, CollaChain does not experience any losses even under bursts, however, its latency increases due to the Twitter load burst.

3.3 The Impact of Contention

Figure 6 depicts the throughput and latency of Hyperledger Fabric and Quorum-Raft. The workload curve is a low constant send rate of 20 transactions per second. The duration is 30 seconds. The workload is purposefully very simple to showcase that transaction conflicts may arise even in low arrival rate situations. The contention percentage denotes the percentage of transactions which modify

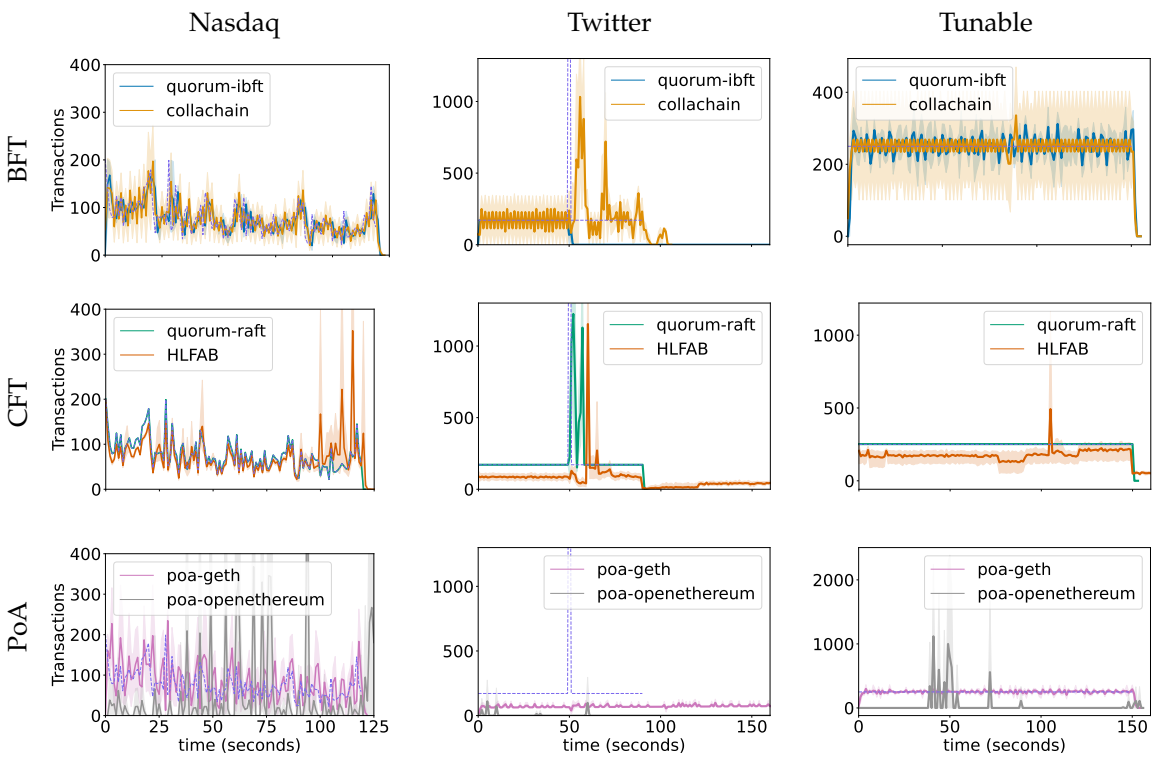

Figure 4: Throughput time series

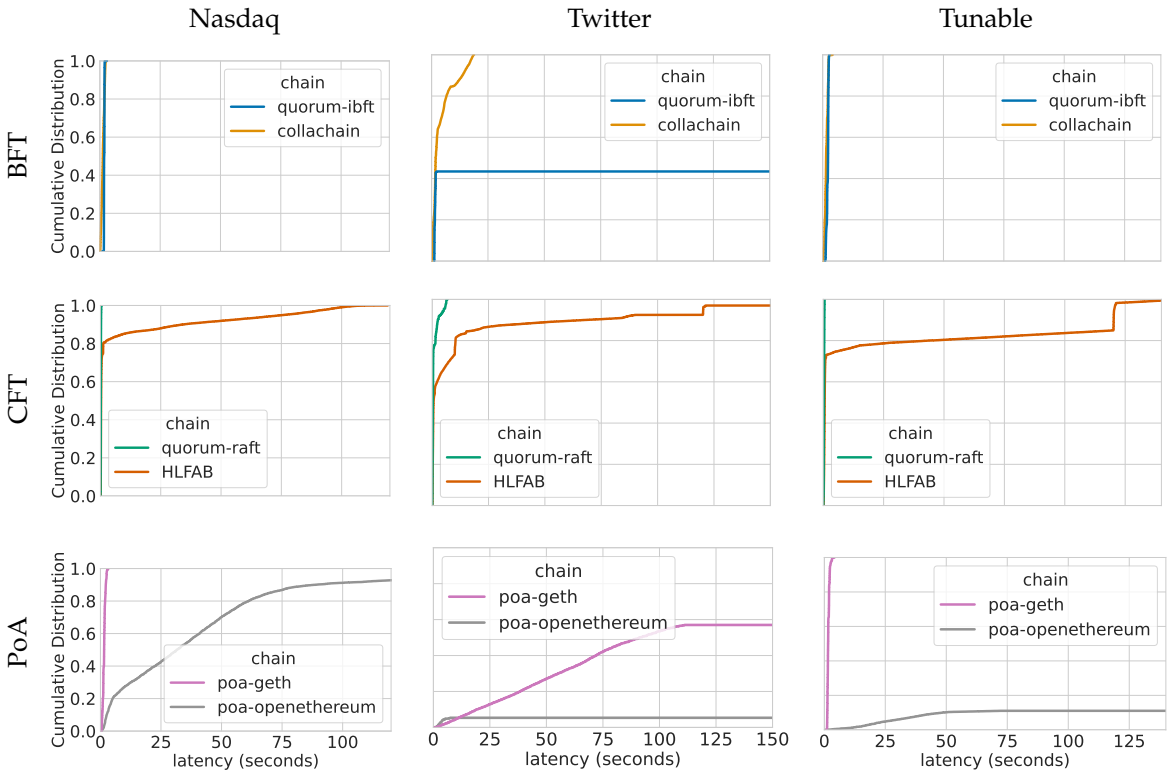

Figure 5: Cumulative distribution functions (CDFs) of the latency

the same asset.

3.3.1 Drawbacks of speculative executions

As a permissioned blockchain, Hyperledger Fabric can have parallel pre-order execution on a subset of network nodes (subset defined by the endorsing policy). However, a model that executes each transaction in parallel is inherently unable to detect transaction conflicts (e.g., concurrent modifications of a same asset) during execution.

As an example, suppose the system supports a throughput of 1000 transactions per second. Additionally, suppose 20 transactions try to modify the same asset in each block of 100 transactions. Then, only one of the 20 transactions will be valid and the rest invalid. The "valid" conflicting transaction out of the 20 is the first one to appear in the block of 100 transactions. Subsequently, suppose all 19 aborted clients attempt to re-execute their transactions. These add up to the 20 new conflicting transactions in the next block. This leads to 38 invalid transactions in the next round, and so on.

More generally, with a cumulative re-execution, the number of aborted transactions grows linearly until it surpasses the throughput of the system. Thus, if clients re-execute invalid transactions as their default behaviour, this effectively becomes an unintentional denial of service attack on the blockchain [7]. Although the idea of re-executing transaction was proposed to bypass this limitation, it appears that such solutions are not ready for production as they are not durable (cf. Section 4).

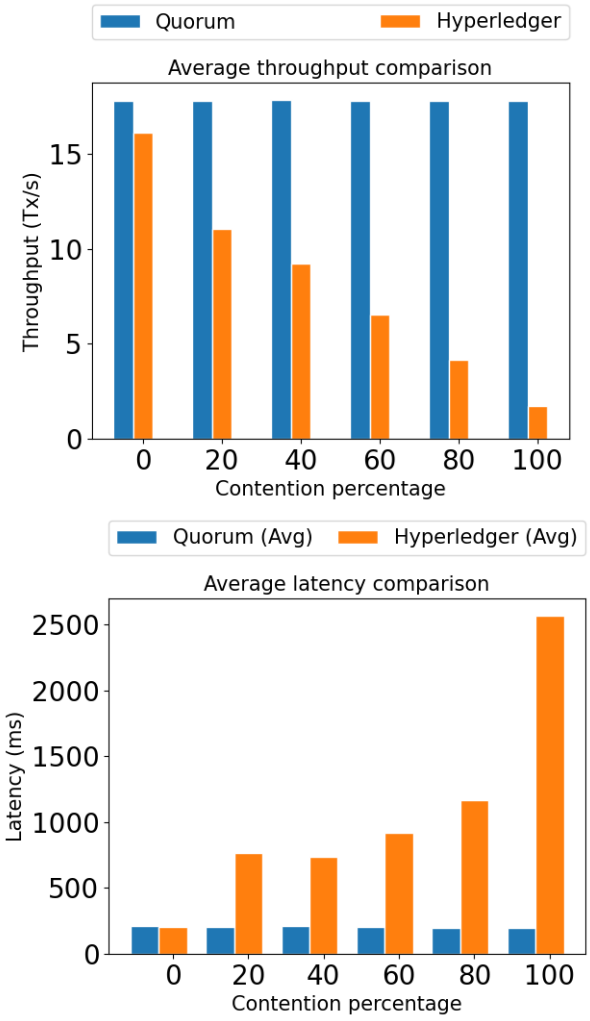

Figure 6: Throughput and latency comparison of Hyperledger Fabric and Quorum over contention percentage

3.3.2 Benefits of pessimistic executions

Most blockchains, including CollaChain, Ethereum, Quorum, execute transactions pessimistically, ordering transactions before executing them. We focus here on Quorum-Raft that is crash fault tolerant and can thus be compared to Hyperledger Fabric. Quorum-Raft, being a fork of geth, also implements this order-execute pattern in its transaction flow. Therefore, even under fully contentious workloads, it is not susceptible to transaction conflicts as every node in the network will execute the transactions in the same order. Every node should thus end with the same results (as long as the smart contract is deterministic). As a result, the performance of these blockchains should not be affected by the example mentioned above. In particular, Figure 6 shows that the performance of Quorum-Raft is stable in this workload, no matter the contention per-

centage.

3.4 The Impact of Fault Tolerance

In order to assess the inherent cost of tolerating Byzantine faults we compared the performance of the two versions of Quorum: Quorum-Raft that only tolerates crash failures but cannot tolerate Byzantine (arbitrary) failures and Quorum-IBFT that also tolerates Byzantine failures. Figure 7 shows the difference in performance between the BFT and CFT versions of Quorum (Quorum-IBFT and Quorum-Raft) and presents the performance of PoA-geth and PoA-OpenEthereum as baselines for comparison. In particular, the graph indicates the evolution of latency as we reach higher throughput. We can clearly observe that Quorum-IBFT is slower than Quorum-Raft. In particular, the latency of Quorum-IBFT starts booming at a lower throughput than Quorum-Raft. This tends to indicate that the BFT consensus induces a higher overhead than the CFT consensus.

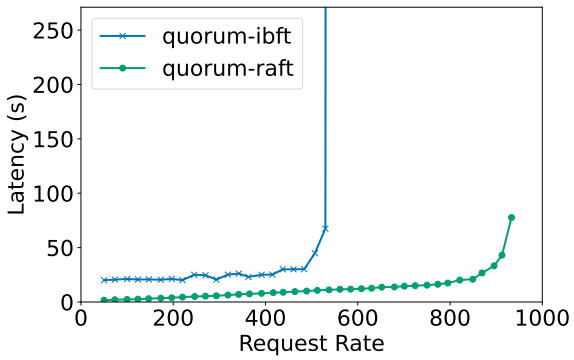

Figure 7: The BFT and CFT versions of Quorum

3.5 Aviation Supply Chain

We now present performance result of Hyperledger Fabric when running an aviation supply chain application inspired by the Honeywell Aerospace use-case [10]. Since the aviation industry is heavily regulated, sales in the U.S. require certification from the U.S. Federal Aviation Administration and other agencies. Each part must be documented with a complete history of its ownership, use, and repairs. Online buying of aircraft pieces in the style of Amazon thus requires trust and this is done by having a trusted ledger with data integrity which enables: tracking of the entire lifecycle of a part and all its previous owners and anti-counterfeit measures via certification and persistence. Because aircraft pieces are expensive and large, deals tend to be made using purchase orders and not card payments. For users to be confident in their purchases, every purchase order is stored indefinitely on the ledger to provide traceability. The corresponding chain code has 3 main functions:

- CreatePart() which creates an aviation part and adds it to the ledger.
- QueryByOwner() which queries all parts owned by a certain owner. Depending on the smart contract language, this may not be easily implemented.
- TransferPart() which transfers an aviation part from owner to a new one by modifying the owner field of the part and also adds a purchase order in the ledger to provide traceability of the aviation part.

As the trace is not publicly available, we defined the curve following a non-homogeneous Poisson process (NHPP). The assumption made here is that the intensity (mean arrival per second) of the Poisson process changes with a cubic rate as a function of time. The justification behind this assumption is that it mimics the very quick change in traffic between peak hour (e.g. day time just after work) and quiet hour (e.g. night time). Here we scaled it down to minutes for benchmarking purposes but the justification remains the same. It can be noted that the curve stays at its peak for 6 seconds before going down. The experiment includes 9 different workloads obtained by changing two parameters: the length (1, 2, or 3 minutes) and the peak intensity of the Poisson Process (100, 200, or 250). The transactions invocations are made up of 50% CreatePart(), 25% QueryByOwner() and 25% TransferPart().

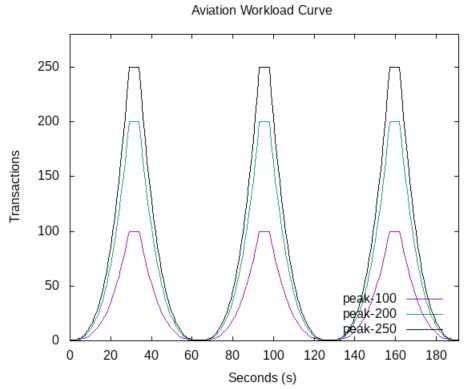

Figure 8: Aviation supply chain workload

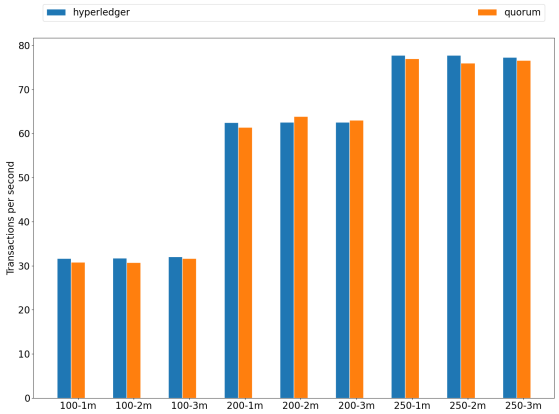

Figure 9: Average throughput in the aviation workloads. "100", "200" and "300" represent the peaks, "1m", "2m" and "3m" denote the duration and number of peaks.

Each set of experiments were run 3 times, ensuring averages were taken during the calculation of results. Figure 8 depicts the sending rate for each experiment with differing peaks. In these experiments, we modified (i) the number of peaks and duration of the tests, and (ii) the intensity of the peaks. Figure 9 presents the average throughput observed throughout these experiments. We can observe that the duration of the workload has little impact on the average throughput.

4 Related Work

It is no surprise that benchmarking and performance evaluation has caught the interest of many, assisting in the understanding of the many blockchain solutions currently available. Correct performance evaluation is critical in the identification of advantages and challenges faced by different systems, and their applicability to numerous use cases. Unfortunately, a majority of the results available to the public are released through non peer-reviewed channels, often through blogs or whitepapers where the evaluations were tailored for optimal environments. This gave rise to a number of benchmark frameworks that target multiple blockchains.

Existing blockchains benchmarks. Hyperledger Caliper [2] is a blockchain benchmark framework enabling users to evaluate the performance of blockchains developed within the Hyperledger project, such as Fabric, Sawtooth, Iroha, Burrow and Besu. It also supports Ethereum and has plans to extend to other blockchains in future. Caliper provides pre-defined workloads, specifying the calling contract, functions and the rate of transaction sending over time. Caliper is not designed to add DApps easily.

Blockbench [5] is one of the most notable benchmarking frameworks for blockchains, as it supports a number of micro and macro benchmarks. The most significant aspect of Blockbench is that it ports over the notable YCSB workload from the database systems community. Aimed at private blockchains, Blockbench benchmarks the different layers of the blockchain, such as consensus or data storage, with tailored workloads, allowing fine-grained testing and measurement of the effectiveness of each of these layers. The evaluation metrics available show throughput and latency, but also the tolerance of faults through injected delays, crashes and message

corruption. Blockbench's complexity introduces difficulty when extending to other blockchains or introducing new workloads as well as moving to distributing the workload across multiple machines.

The variety of Ethereum adapted blockchains motivated the development of Chainhammer [11], a benchmark tool focused on the performance of Ethereum-based blockchains under extremely high loads. Chainhammer, unlike others, does not follow a workload curve but provides continuous high load generation, aiming to measure the throughput in extreme situations. The design is extremely specialised, meaning that there is little flexibility in modifications to support other workloads. Conversely, as Chainhammer is exclusively for Ethereum, it can perform post-benchmark analysis and obtain metrics on information critical to the Ethereum infrastructure, such as transaction cost analysis and the structure of blocks.

Ad-hoc blockchain evaluations. Most evaluations that were made on blockchains are ad hoc and do not aim at comparing very different blockchain designs on the same ground. Previous works have, for example, focused on permissioned blockchains as one can find in the context of Internet of Things [9], others have evaluated exclusively Byzantine fault tolerant blockchains [13].

A comprehensive performance study [15] led to the identification of bottlenecks in Hyperledger Fabric v1.0 and provided guidelines to design applications and operate the network to attain a higher throughput. As a result, few optimizations on the peer process have already been included in Fabric v1.4, and hence our evaluation includes these optimisations.

FastFabric [8] improves over Hyperledger Fabric in order to reach 20,000 transactions per second. It features optimisations that replace the state DB with a hash table, storing blocks in a separate server, separating the committing and endorsing peers into different servers, parallelly validating the transactions headers, and caching the unmarshaled blocks. XOX Fabric [7] builds upon FastFabric and goes a step further and introduces an additional execution-step to enable correct performance even in the presence of contentious workloads. In a later paper [14], Fast Fabric was mentioned to feature "optimizations [that] are not practical for a production environment [...] there is no disk IO which is not suitable for a production setup." XOX seems to be a complicated solution to a specific use-case that may not be frequently encountered.

5 Conclusion

We presented DIABLO an extensible benchmarking tool for blockchain systems. It includes build-in workloads that are realistic and synthetic but allows DApps designers to add new workloads easily. We illustrated DIABLO by comparing the performance of six different blockchain implementations offering crash fault tolerance, byzantine fault tolerance with and without assuming synchrony.

Future work includes adding more blockchains, like Algorand and Facebook's Diem, and evaluating the scalability and security of blockchain systems.

References

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, pages 30:1–30:15, 2018.

[2] Hyperledger caliper. https://hyperledger.github.io/caliper/, Accessed:2021-05-06.

[3] Coinmarketcap. https://coinmarketcap.com/, Accessed:2021-05-06.

[4] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red belly: a secure, fair and scalable open blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*, May 2021.

[5] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. Blockbench: A framework for analyzing private blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*, page 1085–1100, 2017.

[6] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. The Attack of the Clones against Proof-of-Authority. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS'20)*, Feb 2020.

[7] Christian Gorenflo, Lukasz Golab, and Srinivasan Keshav. Xox fabric: A hybrid approach to blockchain transaction execution. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9, 2020. doi:10.1109/ICBC48266.2020.9169478.

[8] Christian Gorenflo, Stephen Lee, Lukasz Golab, and Srinivasan Keshav. Fastfabric: Scaling hyperledger fabric to 20,000 transactions per second. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 455–463, 2019. doi:10.1109/BLOC.2019.8751452.

[9] Runchao Han, Gary Shapiro, Vincent Gramoli, and Xiwei Xu. On the performance of distributed ledgers for internet of things. *Internet of Things*, 10, Aug 2019.

[10] Honeywell aerospace creates online parts marketplace with hyperledger fabric, 2020. Accessed: 2021-05-19 - https://www.hyperledger.org/learn/publications/honeywell-case-study.

[11] A. Krüuger. Chainhammer: Ethereum benchmarking, 2017. https://github.com/drandreaskrueger/chainhammer, Accessed:2021-05-06.

[12] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Jorge Esteves Veríssimo. Deconstructing blockchains: A comprehensive survey on consensus, membership and structure. Technical Report 1908.08316, arXiv, 2019. URL: http://arxiv.org/abs/1908.08316.

[13] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. The performance of byzantine fault tolerant blockchains. In *Proceedings of the 19th IEEE International Symposium on Network Computing and Applications (NCA'20)*, pages 1–8, Nov 2020.

[14] Parth Thakkar and Senthilnathan Natarajan. Scaling hyperledger fabric using pipelined execution andsparse peers. Technical Report 2003.05113, arxiv, 2020.

[15] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. Technical Report 1805.11390, arXiv, 2018. URL: http://arxiv.org/abs/1805.11390.

[16] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.