Diablo: A Benchmark Suite for Blockchains

Vincent Gramoli
University of Sydney
Sydney, Australia
EPFL
Lausanne, Switzerland
vincent.gramoli@sydney.edu.au

Rachid Guerraoui
EPFL
Lausanne, Switzerland
rachid.guerraoui@epfl.ch

Andrei Lebedev
University of Sydney
Sydney, Australia
EPFL
Lausanne, Switzerland
andrei.lebedev@sydney.edu.au

Chris Natoli
University of Sydney
Sydney, Australia
chrisnatoli.research@gmail.com

Gauthier Voron
EPFL
Lausanne, Switzerland
gauthier.voron@epfl.ch

Abstract

With the recent advent of blockchains, we have witnessed a plethora of blockchain proposals. These proposals range from using work to using time, storage or stake in order to select blocks to be appended to the chain. As a drawback it makes it difficult for the application developer to choose the right blockchain to support their applications. In particular, the scalability and performance one can obtain from a specific blockchain is typically unknown. The claimed results are often obtained in isolation by the developers of the blockchain themselves. The experimental conditions corresponding to these results are generally missing and the lack of details make these results irreproducible.

In this paper, we propose the most extensive evaluation of blockchain to date. First, we show how the experimental settings impact the performance of 6 state-of-the-art blockchains and argue for more detailed experiments. Second, and to cope with this limitation, we propose a unifying framework to evaluate blockchains on the same ground. The framework includes a suite of 5 realistic Decentralized Applications (DApps), helps deploy the blockchain nodes at different scales and evaluate their performance. Finally, we show that selecting a particular virtual machine or weakening guarantees can help handle computationally demanding workloads but that none of the tested blockchains can yet support the load of these realistic DApps.

CCS Concepts: • **Computer systems organization → Reliability**; • **Computing methodologies → Distributed algorithms**; • **Networks → Security protocols**.

Keywords: scalability, decentralization, security, performance, latency, throughput, byzantine

1 Introduction

With the growing adoption of blockchain technology, the number of readily-available solutions has multiplied dramatically. As of March 2021, approximately five thousand distinct cryptocurrencies have been reported on a single website [2]. Each of these consists of a separate protocol offering distinctive features like speed, a new financial service, scalability, etc. 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 blockchain implementation. A recent survey [28] highlights the breadth of the blockchain landscape through a classification 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 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 described in the form of white papers that present a 10-000-foot-view of their implementation details. As an example, the Ethereum yellow paper [41] 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 chain. In order to analyze 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 [16].

Another approach is for researchers to evaluate blockchains as black boxes by generating workloads and measuring their performance. The idea is to spawn a blockchain network of nodes, to send them transactions that will be propagated and executed at all blockchain nodes while measuring the performance of the blockchain network to store the results of these transactions in the blockchain. Following this approach, many announcements were made online about the performance of a specific blockchain. As an example, Avalanche was recently claimed to achieve 4500 transactions per second (TPS) with a 2 second latency on its official website [1], but we could not find the experimental conditions in which these results were obtained. This could be confusing, especially given that an earlier technical report presented a peak throughput at 1300 TPS [30].

Blockchain	Claimed results			Observed results		
	throughput	latency	setup	throughput	latency	setup
Algorand	1K–46K TPS [26]	2.5–4.5 s [26]	?	885 TPS	8.5 s	`testnet`
Avalanche	4.5K TPS [29]	2 s [8]	?	323 TPS	49 s	`datacenter`
Solana	200K TPS [34]	<1 s [43]	150 nodes	8845 TPS	12 s	`datacenter`

Table 1. Differences between the claimed performance and the actual performance of different blockchains. The results we observed for each blockchain are the best performances we obtained among all configurations we presented in §5.1. These were obtained in the `testnet` and `datacenter` configurations.

There have been some thorough scientific publications about new blockchains and their performance [7, 14, 17, 25]. These publications usually provide detailed environmental settings that allow the reader to reproduce the experiments. Except for too few occasions [14], the blockchains are evaluated in isolation of other blockchains [7, 17, 25] making it hard to compare them. Efforts were separately devoted to compare the performance of different blockchains [3, 15, 20, 33], however, these evaluations are typically done with synthetic workloads that are not representative of real workloads.

With the advent of Decentralized Applications (DApps) comes the possibility to run real workloads. A DApp is a decentralized application that executes smart contract functions and often exposes a web-based frontend to users. They are an inherent part of the Web3, a decentralized version of the web. The smart contracts are generally written in a Turing-complete programming language and their functions are deterministic. A user can request the blockchain network to execute a DApp by sending a request and a fee, expressed in units of *gas*, which fuels the execution of the corresponding request. There exist various DApps, some are decentralized exchanges to trade cryptocurrencies, others are transparent services to decentralize the sharing economy, and a large part of them are games.

In this paper, we make four main contributions:

1. We propose Diablo (DIstributed Analytical BLOckchain benchmark framework)[1], written in 10,083 lines of Go code, that allows developers to evaluate their blockchain with realistic applications. This framework features several DApps including (i) a multiplayer game, (ii) an exchange with the Nasdaq workload, (iii) a web service experiencing the Fifa requests during the soccer world cup, (iv) a mobility service with a Uber workload and (v) a video sharing service with a YouTube workload.
2. We thoroughly evaluate the performance of 6 state-of-the-art blockchains, including Algorand [17], Avalanche [30], Ethereum [41], Diem [9], Quorum [12] and Solana [43]. We demonstrate that their performance is heavily dependent on the underlying experimental settings in which they are evaluated. In particular, we observe important differences between claimed results and the results we obtain. We thus argue in favor of more detailed blockchain experiments.
3. Our development of DApps in the Solidity, Move and Teal languages revealed that executing generic programs on blockchains can be challenging. First, some of the supported programming languages are too low-level to be written easily without a higher-level programming language. Second, the programming languages can have limited support (like for floating point functions). Third, real DApps may not even execute successfully as some of their functions would consume more than the maximum allowed computational steps expressed in gas units. On the bright side, the blockchains based on the Go Ethereum (or `geth`) virtual machine seem to handle generic programs the best.
4. Finally, our main observation is that, despite being innovative in many regards, the blockchains we evaluated are not capable of handling the demand of the selected centralized applications when deployed on modern commodity computers across the world. We also note that the evaluated blockchains with a leader-based Byzantine fault tolerant consensus protocol are more impacted by constantly high workloads than blockchains with weaker (probabilistic or eventually consistent) guarantees. More research is thus necessary to reduce the overhead of secure blockchains before they can fully serve a highly demanding DApp at large scale.

The rest of the paper is organized as follows. We first describe the problem (§2) and then list the 5 decentralized applications we propose (§3). We present the Diablo framework (§4) and our experimental settings (§5) before demonstrating the performance we obtain with 6 state-of-the-art blockchains (§6). Finally, we present the related work (§7) and we conclude by arguing for a more thorough evaluation of upcoming blockchains (§8).

[1]Diablo is a follow-up of the original Diablo framework [10].

2 Problem Statement

It is common to see new blockchain protocols offering supposedly higher performance results than existing ones, however, these results are usually obtained in isolation and are often irreproducible, which makes them hard to compare. Table 1 illustrates the magnitude of the differences between the announced results of recent blockchains and our actual measurements.

Solana [43] claims 250,000 TPS and 200,000 TPS on a 50-node and a 150-node testnets, respectively [34]. However, the official recommendation is for participants to run Solana on machines equipped with at least 12 cores and special AVX2 Intel instructions [35]. It also claims sub-second finality [43] meaning that transactions can supposedly be committed in less than 1 second. The problem is that the Solana blockchain can fork into multiple branches, hence leading to potential inconsistencies, and it is recommended to wait for additional appended blocks (or "confirmations") to ensure a transaction will not abort [5]. We experimented Solana on machines with up to 36 vCPUs and 72 GiB memory and set the number of confirmations to 30 [24] but could only observe an average throughput of up to 8,845 TPS and an average latency of at least 12 seconds (§6).

Avalanche [30] was initially presented in a technical report in 2018 where it could achieve 1300 TPS on 2000 machines with 2 vCPUs and 4 GiB memory each [30]. Avalanche now supports smart contracts and claims to achieve more than 4500 TPS [29] with a 2 second latency [8]. While the settings where these latest results were obtained are not detailed, our experiments showed that Avalanche reaches a peak throughput of 323 TPS and an average latency of 49 seconds. Algorand [17] was shown to achieve 1000 TPS with native transactions, but it recently featured TEAL smart contracts. It was expected to reach 46,000 TPS in 2021 thanks to block pipelining [26] but recent optimizations led to 6000 TPS in 2022 [22]. Unfortunately, it remains unclear whether Algorand would deliver one of these throughputs when deployed in a large network across different countries. In our experiments, we noticed that the highest average throughput across all our experiments is 885 TPS, and after communicating with the development team we got confirmation that only the peak throughput could reach more than 1000 TPS.

Overall, the results observed are surprisingly far from the announced results. The announced results could be better for many reasons, like batching many transactions at a single client, sending dummy requests with an empty payload, running short benchmarks whose requests do not clog any memory pool, however, we believe good blockchain benchmarking practice should inject real workloads sending useful transactions mimicking a distributed set of blockchain clients. This paper is precisely about offering a benchmark to evaluate blockchains on the same ground when running realistic applications.

3 The Decentralized Applications Suite

In this section, we present the five default DIABLO decentralized applications (DApps) used to measure the performance of blockchains in a realistic setting. As summarized in Table 2, each of these DApps illustrates a distinct behavior and runs a workload trace taken from a real centralized application. For the sake of compatibility with all blockchains, we developed DApps in three programming languages: (i) Solidity v0.7.5, the language supported originally by Ethereum but also by Avalanche, Quorum and Solana, (ii) PyTeal v5, the Python language binding for Algorand smart contracts, and (iii) Move v3, the language for Diem smart contracts.

Exchange DApp / NASDAQ. We implemented an exchange DApp as a decentralized exchange (DEX) with a workload trace taken from the National Association of Securities Dealers Automated Quotations Stock Market (NASDAQ). The NASDAQ experiences a boom of trades at its opening at 9 AM Eastern Time Zone. We extracted the number of trades for Google (GOOGL), Apple (AAPL), Facebook (FB), Amazon (AMZN) and Microsoft (MSFT) from the official website [4]. These workloads proceed in burst by experiencing an initial demand of about 800 TPS for Google, 1300 TPS for Amazon, 3000 TPS for Facebook, 4000 TPS for Microsoft and 10,000 TPS for Apple before dropping to 10–60 TPS. The accumulated workload, denoted GAFAM, runs for 3 minutes and experiences a peak of 19,800 TPS before dropping between 25–140 TPS.

The exchange DApp is implemented as an `ExchangeContractGafam` smart contract with functions `checkStock`, `buyGoogle`, `buyApple`, `buyFacebook`, `buyAmazon`, `buyMicrosoft`. Each order consists of invoking the corresponding `buy*` function that, in turn, checks the availability of the stocks before updating the number of available stocks and emitting a corresponding event. More specifically, the process consists of a fungible token available in limited supply implemented by a single integer counter. Each transaction buys 1 token by decrementing the counter after checking that this counter is greater than 0.

Gaming DApp / Dota 2. At the time of writing, gaming is the most popular type of DApps.[2] We thus implemented a gaming DApp executing the trace of the most popular game on Steam, which is a Multiplayer Online Battle Arena video game called Dota 2 [39]. The number of Steam users peaked at 26.85 million in March 2021 [36]. Each match of Dota 2 is played between two teams of five players, with each team occupying and defending their own separate base on the map. A team wins as soon as they destroy the "Ancient" structure located within the base of the opponent team. Our DApp comprises players who interact with each other and with the environment.

[2]https://dappradar.com/rankings

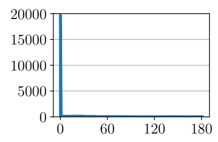
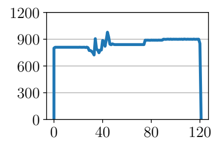
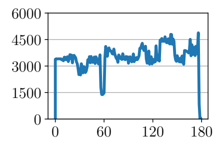
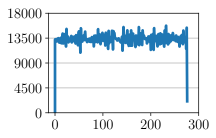
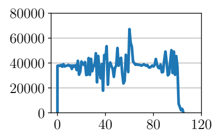
DApp	Exchange	Mobility service	Web service	Gaming	Video sharing
Workload					
Source trace	Nasdaq	Uber	Fifa	Dota 2	YouTube
Characteristics	Burst	Compute intensive	Contended	High sending rate	Very high sending rate

Table 2. Decentralized applications (DApps) used as Diablo benchmarks and their associated workload based on real traces. Each graph shows the number of submitted transactions (y-axis) per second (x-axis).

The gaming DApp is implemented as a smart contract `DecentralizedDota` whose `update` function moves the positions of 10 players along the x-axis and y-axis of a 250-by-250 map so that they turn back whenever they reach the limit of the map. The trace lasts for 276 seconds invoking at an almost constant update rate of about 13,000 TPS, which is particularly demanding.

Web service DApp / Fifa. Web3 promises to decentralize the web by offering DApps that are not controlled by a single institution. We thus naturally implemented a decentralized web service DApp combined with an existing demanding website workload. We selected the Fifa website workload during the 1998 soccer world cup. More than 1.35 billion requests to the Fifa website were recorded over the course of the 84 days of the world cup with an average request length of 3689 bytes. In particular, during the final match on June 30^{th}, 1998, between 11:30 PM and 11:45 PM, the total number of requests reached 3,135,993 for an average request per minute of 209,066. During the most demanded minute of this period, 215,241 requests were sent, translating into an average of 3587 TPS.

In order to measure the number of visits hitting the Fifa website on the day of the final of the 1998 football world cup, we implemented the web service DApp as a simple `Counter` smart contract, with an `add` function, that gets incremented at each request, hence its workload is highly contended. The duration of the workload is 176 seconds, sending an overall 3500 transactions at a rate varying from 1416 to 5305 requests per second.

Mobility service DApp / Uber. Blockchain is often mentioned as a way to bring fairness to the "gig" economy. In this economy, centralized institutions are often criticized to offer services or information to consumers using an opaque algorithm. As an example, Uber has been criticized for manipulating drivers.[3] By contrast, DApps are inherently transparent algorithms because their code is publicly available on the blockchain, which could incentivize developers to design fair algorithms.

[3]https://www.nytimes.com/interactive/2017/04/02/technology/uber-drivers-psychological-tricks.html

We thus implemented a mobility service DApp based on a study of Uber requests in New York City (NYC) from 2018 [11]. The study reports a peak of 16,496 requests per hour between January 2015 and March 2015. As the demand grew since 2015, this peak throughput does no longer reflect the Uber demand. The average number of Uber trips between January and March 2015 was 70,348, reaching 556,387 in March 2019, resulting in a 7.91-fold increase.[4] We thus approximate the current Uber demand in NYC to $16,496 \times 7.91 = 130,483$ requests per hour, which translates into 36 TPS. To extrapolate this demand to Uber world wide, we observe that in the first quarter of 2019 nearly 1,55 billion Uber trips were booked around the world while 63,48 million Uber trips were booked in NYC alone [38]. As the NYC demand represents 1/24 of the world demand, we derive the Uber demand globally to $24 \times 36 = 864$ TPS. Note that this is an approximation as the Uber demand varies between cities.

The mobility service DApp consists of a `ContractUber` smart contract whose function `checkDistance` computes the distance between the customer (the requester) and 10,000 drivers in an area (a 2-dimension grid) of $10,000 \times 10,000$ in order to match the closest driver to the customer. As neither the PyTeal nor the Move languages support floating points or define the square root function $\sqrt{}$ to compute Euclidean distances, we implemented the Newton's integer square root function in Solidity, PyTeal and Move languages, and used it to compute the Euclidean distance. As Algorand DApps state is limited to key-value pairs, the PyTeal implementation of `ContractUber` only stores the position of one driver and computes the Euclidean distance to this unique driver 10,000 times. As the function executes a loop with 10,000 iterations computing the distance, the mobility service DApp is computation intensive.

Video sharing DApp / YouTube. Blockchains promise to decentralize the sharing economy by rewarding prosumers instead of large corporations, a tendency illustrated

[4]https://toddwschneider.com/dashboards/nyc-taxi-ridehailing-uber-lyft-data/.

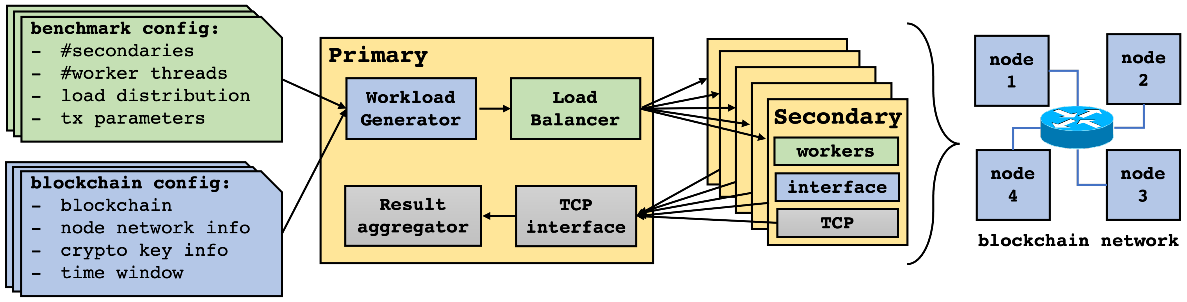


Figure 1. The architecture of Diablo comprises configuration files for the Primary to send the right workload for the right blockchain to a set of Secondaries that then send requests to blockchain nodes and collect performance results from these blockchain nodes.

by DTube, an alternative to YouTube that rewards the creation and consumption of video content.[5] We thus implemented a video sharing DApp based on the number of videos uploaded to YouTube [18]. More precisely, from the YouTube traffic observed during 3 months of 2007, we extracted the day with the peak request rate and the hour within this day with the peak request rate of 1,680,274 transactions per hour. We normalized this result to obtain a request rate of 467 transactions per second. Between 2007 and 2021, the number of videos uploaded to YouTube has been multiplied by 83 [37], hence we approximate the average throughput to $467 \times 83 = 38,761$ TPS, which makes this DApp very demanding. The video sharing DApp corresponds to a smart contract called `DecentralizedYoutube` with an `upload` function that gets some video data as a parameter and assigns the requester's address to the data before emitting a corresponding event.

4 Diablo Overview

Diablo, whose architecture is depicted in Figure 1, is a benchmark suite to evaluate blockchains with realistic applications. To facilitate distributed workload generation, Diablo comprises two main components, a single *Primary* and multiple *Secondaries*. For the sake of extensibility, Diablo offers a blockchain abstraction with 4 functions that a developer can implement to compare their new blockchain protocol to existing blockchains and Diablo also offers a workload specification language for a developer to add new DApps.

Primary. The purpose of the Primary machine is to coordinate the experiment: it generates the workload and dispatches it between Secondaries, launches the benchmark, aggregates the results and reports them back.

Prior to starting the benchmark, its workload generator parses the benchmark and blockchain configuration files. A benchmark configuration file indicates the requests type, whether requests are native transfers or DApp invocations, and their distribution between Secondaries and blockchain nodes over time (§4). For each workload invoking DApps, the Primary also deploys the smart contracts listed in the benchmark configuration file. The blockchain configuration file is necessary to generate the workload appropriately because the transaction distribution depends on the number and locations of the deployed blockchain nodes. Then, the Primary transmits a description of the transactions to the Secondaries, waits for all Secondaries to be ready and informs all Secondaries when to start the benchmark.

Once the benchmark is complete, each Secondary sends its results to the Primary through a TCP interface and an aggregator collects them to output a JSON file, indicating the start time and end time of each transaction (as recorded by the Secondaries). These timestamps can then be used postmortem to generate time series and analyze the distribution of latencies (e.g., Fig. 6) or more simply to output aggregated values like the average (e.g., Fig. 3).

Secondary. Secondaries are responsible for the presigning of the transactions and the execution of the workload, interacting directly with blockchain nodes. The number and specification of the Secondaries are typically chosen to match the resources allocated to the blockchain (cf. Table 3) to be able to stress test the blockchain. Note that each Secondary can send requests to multiple blockchain nodes.

Each Secondary spawns a number of explicit worker threads as told by the Primary and indicated in the benchmark configuration file. (Note that the level of concurrency can be higher due to implicit go routines.) These worker threads mimic individual clients issuing requests concurrently. The Secondary schedules the transactions following the workload distribution instructed by the Primary.

The Secondary interacts with the blockchain through a client interface specific to each blockchain. The current clock is recorded as the submission time right before a transaction is sent. The Secondaries constantly check if the submission time is not too late compared to the time demanded by the Primary and emit a warning otherwise. Each worker thread constantly polls the blockchain nodes to obtain the last block and checks whether it contains sent transactions. When a

[5]https://d.tube.

sent transaction is detected within a block, the current clock time is recorded as the decision time for this transaction.

Blockchain abstraction. To make Diablo compatible with various blockchain implementations, we abstract away the main components of a blockchain. The Diablo benchmark specification interacts with the resulting blockchain abstraction. A blockchain is modeled as a tuple $\langle E, R, I \rangle$ where E is the finite set of *endpoints* that act as blockchain nodes, R is a finite set of resources (e.g., account balance, smart contract state) maintained in the blockchain state, and I is a potentially infinite set of interactions types (e.g., asset transfer, smart contract function invocation) between a client and the blockchain. Let C be the set of clients. An interaction event is denoted as a tuple $\{(c, i, r, t)\}$ with $c \in C$, $i \in I$, $r \in R$ and the time $t \in \mathbb{R}$.

The benchmark specification contains a function M mapping the Secondaries to the blockchain endpoints, the sets φ^C, φ^R and φ^I that specify the clients, resources and interactions types needed for the test and the interactions $\{(\varphi^c, \varphi^i, \varphi^r, t)\}$ where $\varphi^c \in \varphi^C$, $\varphi^i \in \varphi^I$, $\varphi^r \in \varphi^R$, $t \in \mathbb{R}$. More precisely, $M : S \times E \Rightarrow \varphi^C$ where S is the set of Secondaries, E is the set of endpoints, both available only at runtime, and φ^C is the set of specified clients, each specified client is implemented by an explicit worker thread. The two types of interactions are `transfer_X` to transfer `X` coins from one account to another one and `invoke_D_Xs` to invoke a DApp `D` with the parameters `Xs`.

To add a new blockchain, one has to implement at least one of these interaction types as well as 4 functions that convert the benchmark specification to an executable test program: (i) s`.create_client(`E`)` where $s \in S$, (ii) `create_resource(`φ^r`)` and $\varphi^r \in \varphi^R$, (iii) `encode(`φ^i, r, t`)` where $t \in \mathbb{R}$ and $r \in R$ to produce an opaque encoded interaction e, and (iv) c`.trigger(`e`)` to send the encoded interaction from client c to the blockchain. Since these functions are relatively fine grained, the implementations for the blockchains we test are small sized: between 1,000 and 1,200 LOC of Go.

Workload specification. The benchmark configuration file specifies the function M, the set φ^R and the interactions $\{(\varphi^c, \varphi^i, \varphi^r, t)\}$ described above. For example, the gaming DApp configuration file below defines 4 variables: `acc` (line 4) is a set of 2,000 user accounts, `dapp` (line 5) is a set containing one instance of the `dota` DApp. Those 2 variables form the φ^R set. The variable `loc` (line 2) is the set of Secondaries tagged with the string `us-east-2` (an AWS availability zone) and `end` (line 3) is the set of all endpoints. These two variables are used in the definition of the M function (lines 7-10), which defines 3 clients invoking the DApp `dapp` (line 14) from accounts in `acc` (line 13) with the parameters parsed from `update(1, 1)` (line 15) at the rate specified in the `load` section (lines 16-19): each client sends 4432 TPS for the first 50 seconds then 4438 TPS for the next 70 seconds, after which the benchmark ends.

```
 1  let:
 2    - &loc { sample: !location [ "us-east-2" ] }
 3    - &end { sample: !endpoint [ ".*" ] }
 4    - &acc { sample: !account { number: 2000 } }
 5    - &dapp { sample: !contract { name: "dota" } }
 6  workloads:
 7    - number: 3
 8      client:
 9        location: *loc
10        view: *end
11        behavior:
12          - interaction: !invoke
13              from: *acc
14              contract: *dapp
15              function: "update(1, 1)"
16            load:
17              0: 4432
18              50: 4438
19              120: 0
```

5 Experimental Settings

In this section, we detail the experimental settings to evaluate 6 different blockchains when deployed in 5 configurations, called `datacenter`, `testnet`, `devnet`, `community` and `consortium`, on up to 200 machines distributed in 10 countries around the world.

5.1 Deployment configurations

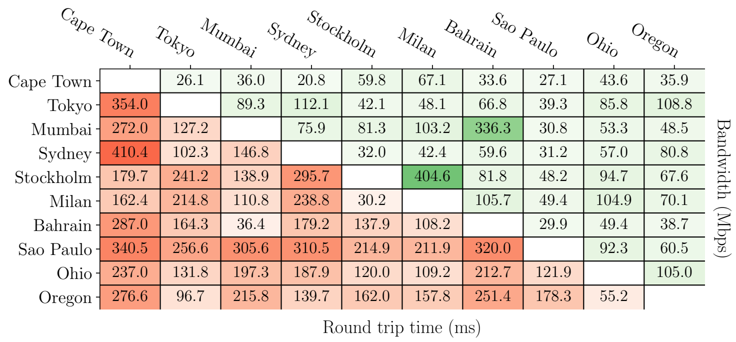
We deployed Diablo and the blockchains in different configurations with up to 200 virtual machines ranging from AWS c5.xlarge instances (with 2 vCPUs and 4 GiB memory each) to c5.9xlarge instances (with 36 vCPUs and 72 GiB memory each) and spread equally among different geo-distributed regions in five continents: Cape Town, Tokyo, Mumbai, Sydney, Stockholm, Milan, Bahrain, São Paulo, Ohio, Oregon. Table 3 (left) lists these different configurations.

Datacenter. The `datacenter` configuration aims at showcasing the blockchain peak performance in an idealized setting. Such a configuration features powerful c5.9xlarge machines located in the closed network of a single datacenter, the Ohio AWS availability zone. These machines are not commodity hardware as each machine features 36 vCPUs and 72 GiB memory, the bandwidth and latency between machines are 10 Gbps and 1 ms[6], respectively, which is not representative of an open network. Instead, this configuration allows us to evaluate blockchains when a lot of resources are available.

Testnet. The `testnet` configuration features small c5.xlarge machines located in a single datacenter, the Ohio

[6]https://aws.amazon.com/ec2/instance-types/c5/.

Configuration	Blockchain nodes			Regions
	number	#vCPUs	memory	
`datacenter`	10	36	72 GiB	Ohio
`testnet`	10	4	8 GiB	Ohio
`devnet`	10	4	8 GiB	all
`community`	200	4	8 GiB	all
`consortium`	200	8	16 GiB	all



	Cape Town	Tokyo	Mumbai	Sydney	Stockholm	Milan	Bahrain	Sao Paulo	Ohio	Oregon
Cape Town		26.1	36.0	20.8	59.8	67.1	33.6	27.1	43.6	35.9
Tokyo	354.0		89.3	112.1	42.1	48.1	66.8	39.3	85.8	108.8
Mumbai	272.0	127.2		75.9	81.3	103.2	336.3	30.8	53.3	48.5
Sydney	410.4	102.3	146.8		32.0	42.4	59.6	31.2	57.0	80.8
Stockholm	179.7	241.2	138.9	295.7		404.6	81.8	48.2	94.7	67.6
Milan	162.4	214.8	110.8	238.8	30.2		105.7	49.4	104.9	70.1
Bahrain	287.0	164.3	36.4	179.2	137.9	108.2		29.9	49.4	38.7
Sao Paulo	340.5	256.6	305.6	310.5	214.9	211.9	320.0		92.3	60.5
Ohio	237.0	131.8	197.3	187.9	120.0	109.2	212.7	121.9		105.0
Oregon	276.6	96.7	215.8	139.7	162.0	157.8	251.4	178.3	55.2	

Table 3. The experimental settings range from a `datacenter` scenario with extensive resources to a `testnet` of collocated machines, to a geo-distributed `devnet`, to a large-scale `community` of machines, to a large-scale `consortium` of modern machines. The left side indicates the number of blockchain nodes deployed, the hardware they use and on how many regions they are spread. The right side indicates the bandwidth (top right corner, in green) and round trip time (bottom left corner, in red) between each region measured with `iperf3` on machines from the devnet configuration.

AWS availability zone. This typically corresponds to a testnet setting were blockchain developers typically run their blockchains in order to assess performance and stability during development phases. As the machines are cheaper to rent than c5.9xlarge, they allow the testnet to run for long period of time, allowing for continuous deployment.

Devnet. The `devnet` configuration geo-distributes the machines in an open network to assess the performance in a setting involving the network latencies over long distances. This intends to mimic the performance one could expect from a blockchain devnet, where selected beta testers or preliminary validators from different regions could participate in the evaluation of the blockchain. Once the evaluation with external beta testers is successful, then the devnet is ready to be opened to all internet users as a mainnet.

Community. The `community` configuration increases the number of machines to about the number of countries around the world. This configuration aims at mimicking the behavior of a geo-distributed mainnet involving as many blockchain participants as there are jurisdictions (there are currently 195 universally recognized self-sovereign states in the world). Such a highly distributed setting is often considered to be particularly censorship resistant because it would not be strongly affected by political decisions in only one of the jurisdictions where it operates.

Consortium. The `consortium` configuration geo-distributes 200 blockchain nodes similarly to the `community` configuration, however, it features more powerful c5.2xlarge machines that better represent modern computers featuring 8 vCPUs and 16 GiB of memory. This aims at mimicking a consortium of individuals or institutions, like the R3 consortium [27], who have resources to devote modern machines without specialized hardware to participate in the blockchain service.

5.2 Blockchains

In this section we describe the six blockchains with smart contract support that we compare using Diablo. The reason why we chose these blockchains is because they form a diverse set of representative blockchains: Algorand is a blockchain that appeared in a peer-reviewed scientific publication, Avalanche partially orders transactions in a directed acyclic graph, Ethereum is the largest smart contract blockchain in market capitalization, Quorum is a blockchain originally developed by the finance industry and Solana is one of the most recent smart contract blockchains. Their characteristics are listed in Table 4.

Blockchain	Prop.	Consensus	VM	DApp lang.
Algorand [17]	prob.	BA* [17]	AVM	PyTeal
Avalanche [30]	prob.	Avalanche [30]	`geth`	Solidity
Diem [9]	det.	HotStuff [44]	MoveVM	Move
Quorum [12]	det.	IBFT [32]	`geth`	Solidity
Ethereum [41]	◇	Clique [21]	`geth`	Solidity
Solana [43]	◇	TowerBFT [42]	eBPF	Solidity

Table 4. Blockchains evaluated in Diablo. They differ by their language virtual machine (VM), the language in which their DApps are written, their consensus protocols and the properties (Prop.) they offer, which are either probabilistic (prob.), deterministic (det.) or eventual (◇).

Algorand. Algorand [17] is a proof-of-stake blockchain that elects a subset of nodes, through sortition, that can append the next block. It does not fork with high probability, so the transaction is considered final as soon as it is included in a block.[7] Algorand features a blocking API that waits for the

[7]https://algorand.foundation/algorand-protocol/core-blockchain-innovation.

transaction to be committed before returning to the client. Although it makes it natural to use this blocking call to detect the commit of each transaction, Diablo was too demanding, hence we made Diablo poll every appended block to detect transaction commits, which improved significantly Algorand's performance.

We experimented the Algorand version with commit `116c06e` dated from Nov. 23^{rd} 2021 and available at https://github.com/algorand. Note that further optimizations were made in the meantime [22]. We wrote a version of each DApp of §3 in PyTeal because Algorand only supports the Transaction Execution Approval Language (Teal), which is a bytecode language and requires a conversion from the PyTeal higher level language. Unfortunately, we could not implement the video sharing DApp in Teal as we needed data structures that were too large to be stored in the state whose space is limited by a key-value store with 128 bytes per key-value pair.

Avalanche. Avalanche [30] is a blockchain offering probabilistic safety and the possibility to spawn subnets. There are now three blockchain protocols in Avalanche: one featuring the Ethereum Virtual Machine (C-Chain), one supporting only native transfers (X-Chain), and another one for metadata management. To evaluate the DApps of §3, we used C-Chain, which exposes a web socket streaming API (shared by Ethereum and Quorum) to access the current blockchain head or the latest block.

For the experiments, we used the `master` branch with commit number `7840200` available at https://github.com/ava-labs/avalanchego. More precisely, validators only have to validate the primary network of the C-Chain and as any subnet is an independent network, we decided not to spawn any subnet. We initially tried to setup the Avalanche experiments using the RSA4096 cryptographic signature scheme as recommended by Avalanche. However, this signing process was taking too long due to the scale of our experiments. As we could not make Avalanche work after replacing RSA4096 by ED25519, we opted for using ECDSA instead. Avalanche supports the London release improvement of Ethereum (improvement #1559 of August 5th, 2021) with the new gas fee structure with tips, which means the gas fee is computed dynamically (differently from Ethereum's original method). Avalanche limits the gas per block to 8M gas and seems to require a period between blocks of at least 1.9 seconds.[8]

Diem. Diem, formerly known as the Libra blockchain [9], was initiated by Facebook. It features a variant of the leader-based HotStuff protocol that solves the consensus problem deterministically (hence avoiding forks) while reducing the communication of traditional consensus protocols in good executions. Like Ethereum, Diem requires that each transaction contains a sequence number, i.e., a monotonically incremented integer. The difference with Ethereum is that Diem nodes only accept a maximum of 100 transactions from the same signer in their memory pool, limiting the rate at which a unique signer can submit transactions. To bypass this limitation, we made workloads submit from 2,000 different accounts in most deployment configurations, however, we noticed that the provided setup tools would fail systematically after creating 130 accounts. This is why we restricted the number of accounts to 130 in the `community` and `consortium` configurations. We explicitly indicate when this caused issues in §6.

We experimented the testnet branch from Aug. 21^{st} 2021 with commit number `4b3bd1e` of the Diem repository https://github.com/diem/diem. Diem testnet branch is dated Aug. 20 of 2021, while the main branch was updated at the time of writing (Feb. 27, 2022). Even though the testnet branch seems outdated, the official Diem tutorial still recommends using the testnet branch for development purpose: https://developers.diem.com/docs/tutorials/tutorial-my-first-transaction/.

Ethereum. Ethereum [41] is the second largest blockchain in market capitalization. As the default version of Ethereum uses the proof-of-work cryptopuzzle resolution, which inherently limits its throughput (to the amount of gas allowed per block divided by the block period), we exclusively used the Ethereum proof-of-authority consensus protocol, called Clique, as available in `geth`. This version still requires a minimum period between consecutive blocks [16]. Just like Avalanche, the Ethereum API exposes a web socket streaming API to access the current blockchain head or the latest block.

We evaluated the `geth` version from the `master` branch with commit hash `72c2c0a` from Dec. 12 of 2021 available at https://github.com/ethereum/go-ethereum. In August 2021, the "London" update to the gas calculation introduced the notion of tips. With this new version, the gas fee changes at every block, which can impact the execution of transactions: when the fee increases then the transaction risks to be underpriced. This is why, we tried to adjust the fee dynamically during the execution of the benchmark—this implied signing transactions online.

Quorum. Quorum [12] is a blockchain initiated by J.P. Morgan and currently maintained by Consensys. It features different consensus algorithms: Raft, which only tolerates crash failures, and IBFT and QBFT, which both tolerate Byzantine failures and partial synchrony. As Quorum features the `geth` Ethereum Virtual Machine, with the latest changes from the Berlin upgrade (April 15th, 2021), it also features the Clique proof-of-authority consensus algorithm, however, it does not feature the more recent London gas fee computation used by Ethereum and Avalanche.

We experimented the `master` branch with commit hash `919800f` of Quorum from 2 Nov. of 2021 available at https:

[8]https://snowtrace.io/chart/blocktime.

//github.com/ConsenSys/quorum. Given that Clique is vulnerable to message delays [16] and Raft is vulnerable to arbitrary failures, we exclusively run Quorum with IBFT in our experiments. Similar to Ethereum and Avalanche, Quorum exposes a web socket streaming API to access the current blockchain head.

Solana. Solana is a recent blockchain that is highly optimized for special features (e.g., Intel instructions). Similar to Ethereum, Solana may fork and needs to wait for 30 confirmations (additional appended blocks) before a stored transaction can be considered final [24]. Its algorithm builds upon proof-of-history and "depends on messages eventually arriving to all participating nodes within a certain timeout" [43]. To append a block every 400 milliseconds, Solana replaces the Merkle Patricia Trie of Ethereum with a simplified data structure and replaces the ECDSA signature scheme with EdDSA (ED25519).

We experimented the commit number `0d36961` of the `master` branch of Solana from March 12 of 2022, as available at https://github.com/solana-labs/solana. Solana uses its own API, also based on a web socket, that allows the client to specify a commitment level. The clients listen for new blocks with the desired commitment level by subscribing to a web socket interface. Interestingly, Solana fetches the block hash before issuing transactions because the last block hash needs to be signed as part of the issued transaction. Previous tests ran by the Solana team all consisted of requesting the last block hash before issuing concurrently transactions withdrawing from different accounts. We could not use this technique while evaluating realistic DApps because Solana requires the hash to be created less than 120 seconds before the transaction request is received while DApps can run for longer. To cope with this limitation, the Solana-Diablo interface periodically fetches the last block hash.

5.3 Diablo configuration

To measure the impact of geo-distribution on the blockchain performance we deployed both the blockchains and Secondaries in the deployment configurations of §5.1 as depicted in Table 3. In all cases we applied the same geo-distribution strategy to the blockchain nodes and to the Secondaries: each Secondary submits its requests to its collocated blockchain node so as to mimic requests being routed from a client towards its closest blockchain node. In all these configurations, a single Primary was used for setting up the experiment and gathering the performance results. As the Primary is not involved during the performance monitoring phase, its location does not impact the experimental results.

```
diablo primary -vvv --port=5000 \
   --env="accounts=accounts.yaml" \
   --env="contracts=dapps-directory" \
   --output=results.json --compress --stat \
   10 setup.yaml workload.yaml
```

To run the Primary, we specify the verbosity level, port number for the secondaries to connect to, path to the accounts file, DApps source codes, output file path, compress output (with gzip), printing statistics to standard output, number of Secondaries (10 in the example), blockchain setup file, and workload specification file.

```
diablo secondary -v --tag="us-east-2" \
    --port=5000 127.0.0.1
```

To run the Secondary, we again specify the verbosity level, port and address of the Primary to connect to and a tag to indicate the Secondary location for collocation with blockchain nodes.

6 Evaluation Results

In this section, we stress test the blockchains described in §5.2 under the realistic DApps of §3. Due to their decentralized nature that offers enhanced security, we observe that, on modern commodity hardware, the blockchains we evaluated cannot yet handle the workloads experienced by centralized applications. We then run a combination of synthetic workloads and real but less demanding workload traces to compare the scalability, robustness, universality and availability of these blockchains.

6.1 Motivating blockchain improvements

To provide an overview of blockchains performance executing realistic DApps, we deploy each DApp of §3 in the `consortium` deployment configuration (200 machines with 8 vCPUs and 16 GiB of memory spread over 10 countries in 5 continents) and generate the workload associated with each of these DApps. For each run, we make sure that Diablo uses enough Secondaries to not be the bottleneck.

Figure 2 shows the average throughput, average latency and the proportion of committed transactions for each blockchain-DApp pair. Note that these values are extracted from a time series produced during the execution of each workload of Table 2 that we inspected to make sure of the statistical relevance of our measures. We observe that for the Exchange DApp, which has the lowest average workload, Nasdaq, of 168 TPS only, Avalanche and Quorum commit more than 86% of the transactions, all the other blockchains commit 47% or less of the transactions. Although the fact that none of the evaluated blockchains could commit all transactions may seem quite pessimistic, note that recent experiments already demonstrated that some blockchain could commit all of them in the same setting [40].

For the most demanding workload, the YouTube workload, the proportion of commits is lower than 1% for all evaluated blockchains. In addition, when the average workload is of 852 TPS (like the Uber workload), or 3,483 TPS (like the Fifa workload), only Quorum maintains a throughput higher than 622 TPS while the other blockchains have a throughput lower than 170 TPS. For higher workloads (like

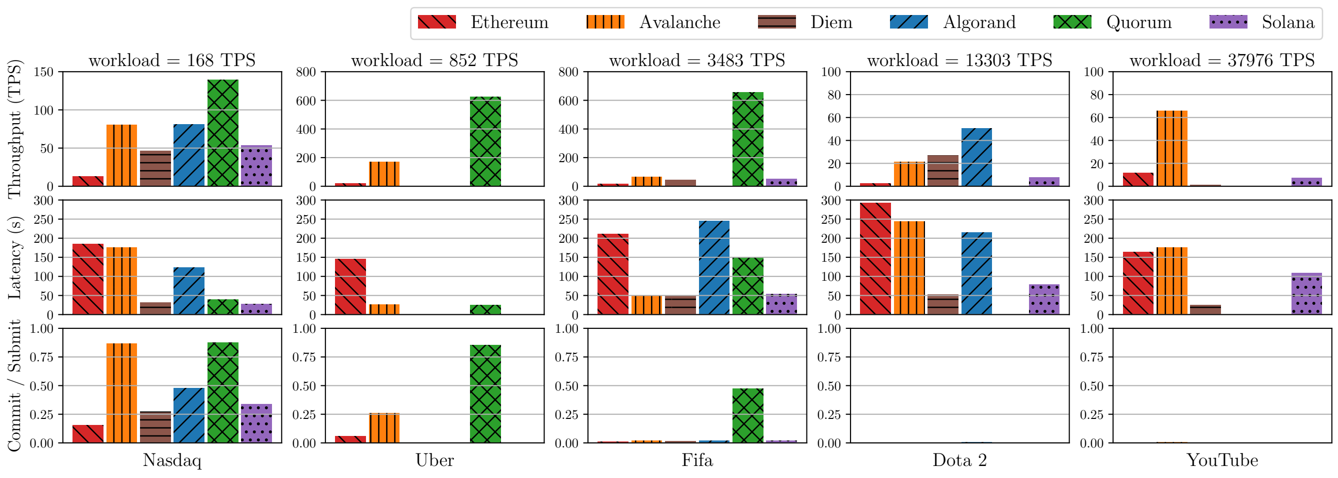


Figure 2. Evaluation of blockchain performance when executing realistic DApps. For each DApp (column), we list the average workload effectively submitted by Diablo (top of each column), average throughput (first row), average latency (second row) and proportion of committed transactions (third row) for each blockchain. Each blockchain is deployed on 200 machines, each with 8 vCPUs and 16 GiB of memory, spread among 10 datacenters. The absence of a bar indicates that the blockchain cannot even commit few requests.

Dota 2), no blockchain maintains a throughput higher than 66 TPS. Finally, among all DApps, no blockchains commit with a latency lower than 27 seconds. These results contrast with the claimed performance that we describe in Table 1. We indicate below what are the causes of this performance gap and how a blockchain developer can use Diablo to find the causes of such performance results.

6.2 Scalability and deployment

Using Diablo, we quantify *scalability* as the ability to allow a large number of unprivileged users to participate to the blockchain execution. To this end, we deploy the blockchains on networks of different sizes composed of machines ranging from enterprise grade hardware with high computational power (`datacenter`) to commodity hardware with modest computational power (`community`). We then measure their performance when stressed with a synthetic workload.

More precisely, after having tried the `consortium` deployment configuration (§6.1), we now deploy each blockchain in the four remaining configurations of Table 4: `datacenter`, `testnet`, `devnet` and `community`. For each configuration, we use Diablo to emulate clients sending native transactions to the blockchain during 120 seconds at a constant rate of 1000 TPS, which is the same order of magnitude as the average load of the Visa system[9]. We measure the average throughput and average latency for each blockchain. If the measured throughput is close to the workload of 1000 TPS then we conclude that the blockchain handles the simple payment use case for the configuration.

[9]Visa claims 150 million transactions per day = 1,736 TPS on average (https://usa.visa.com/run-your-business/small-business-tools/retail.html)

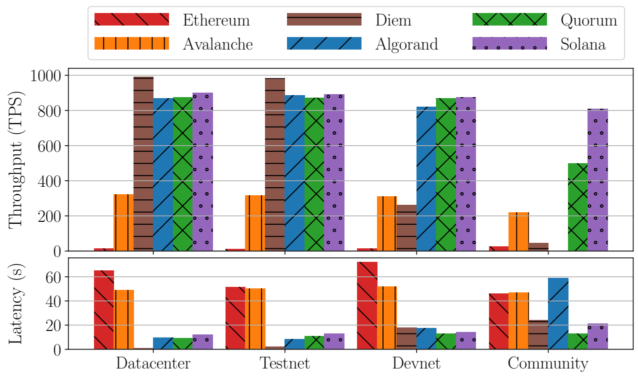


Figure 3. Average throughput and average latency of each blockchain when stressed with a constant workload of 1,000 TPS on different configurations, from the least challenging (`datacenter`) to the most challenging (`community`).

Figure 3 shows the average throughput and average latency for each blockchain on these four configurations. We observe that only Solana handles a 1000 TPS constant workload for all configurations while maintaining a throughput higher than 800 TPS with a latency below 21 seconds. Solana uses an eventually consistent consensus based on a verifiable delay function which puts away all communication steps but a broadcast. By using a verifiable delay function, Solana makes the block generation delay independent from the number of cores the participant uses. By removing most of the communication steps, Solana also performs relatively well in a large-scale configuration like `community`.

Quorum also stands out in the `community` configuration with a throughput of 499 TPS for a latency of 13 seconds. Quorum uses a deterministic consensus algorithm that does not introduce artificial delays and provides immediate finality. In addition, Quorum benefits from many blockchain specific optimizations by using `geth` as a base code. For all blockchains there is no significant difference between the `datacenter` and the `testnet` configurations. Over all the configurations, Diem achieves the highest throughput (more than 982 TPS) and the lowest latency (2 seconds or less) but only on configurations with a local setup. We conjecture that Diem is designed to provide very low latency and is optimized to run on network setups with a low round-trip time (RTT).

Among the remaining blockchains, only Algorand achieves a throughput higher than 820 TPS when deployed on the `devnet` configuration, which is a geo-distributed network. In particular, the best average throughput that Algorand reaches in 885 TPS on the `tesnet`. Avalanche delivers a relatively low throughput, even in the `community` configuration that uses precisely the amount of vCPUs and memory that Avalanche recommends[10]. This could be due to the period between its consecutive blocks[11], similar to the `block-period` used in Ethereum Clique [16]. For this reason, we conjecture that Avalanche and Ethereum are designed to run at a relatively low throughput regardless of the available computational power or network bandwidth.

6.3 Robustness and denial-of-service attacks

To better understand whether blockchains are *robust* to high demand, we used Diablo to inject high workloads and test whether the blockchain collapses or continues treating requests further. Intuitively, a blockchain is more robust than another if the workload needed to negatively affect its latency and throughput is higher than the other. This property is desirable to guarantee a certain service level agreement despite an increasing demand and to better cope with denial-of-service attacks. The test consists of deploying the blockchain in a deployment configuration where it performs well under moderate workloads and of observing whether a higher workload leads to performance degradations.

To compare the blockchain robustness, we deploy each blockchain in the configuration it performed best (see §6.2) and observe its performance when stressed with a high workload. To this end we configured Diablo to send native transactions to the blockchain during 120 seconds at a constant rate of 10,000 TPS, which is 10× higher than the sending rate in the deployment challenge. Although Diablo can send transactions at higher rates, we found this workload to be sufficient to show some interesting behaviors of the tested blockchains.

Figure 4 compares the throughput and latency of each blockchain when stressed with workloads of 1000 TPS and

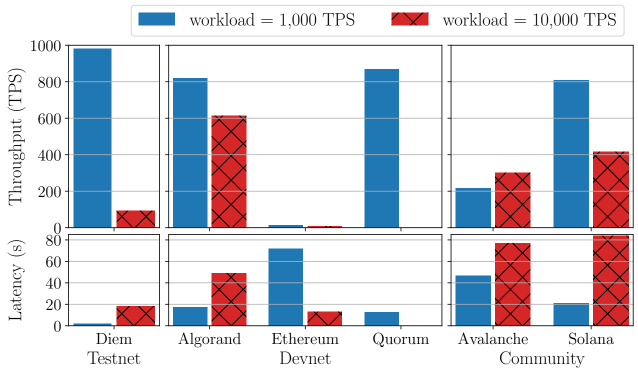


Figure 4. Throughput and latency of each blockchain when stressed with a constant workload of 1000 TPS (left bar) or 10,000 TPS (right bar). Each blockchain is deployed in the configuration where it relatively performs best under a workload of 1000 TPS.

10,000 TPS. Diem and Quorum are the most negatively affected by the higher workload: Diem throughput is divided by 10 while Quorum throughput drops to 0. Interestingly, Diem and Quorum are the only blockchains we evaluated that use a deterministic leader-based Byzantine fault tolerant (BFT) consensus. These algorithms were originally designed to commit as many client requests as possible, a behavior that easily leads to saturate memory pools or network queues when exposed to high workloads. This behavior can increase the vulnerability of the blockchain to DoS attacks.[12] Finally, note that this conclusion does not generalize to all deterministic BFT consensus protocols, as other experiments [40] recently showed that Smart Red Belly Blockchain, which relies on a leaderless Byzantine fault tolerant consensus protocol, is immune to this problem.

Algorand and Solana are more robust than Diem and Quorum as their throughputs are divided by 1.45 and 1.94, respectively, while the latencies of Algorand and Solana are multiplied by 2.43 and 4, respectively. These results show that despite being affected by a high workload, these two blockchains do not completely collapse and the performance drop likely results from the inability of the underlying hardware to handle too many requests. Interestingly, Avalanche throughput is not negatively affected by the higher workload, as its throughput is multiplied by 1.38 which makes it comparable to Solana throughput for the same workload. This confirms the conjecture of §6.2 that Avalanche throttles its throughput. It is hard to say something about Ethereum results since this blockchain only commits 0.09% of the transactions when the workload is 10,000 TPS.

6.4 Universality and DApp executions

To understand whether a blockchain is *universal* in that it can handle requests that are made arbitrarily complex, we

[10] https://github.com/ava-labs/avalanchego#installation.

[11] https://snowtrace.io/chart/blocktime.

[12] Generating 10,000 TPS with Diablo costs less then 8 USD/hour on AWS.

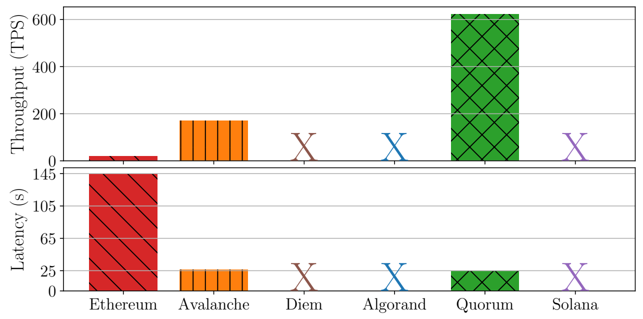


Figure 5. Throughput and latency of each blockchain when stressed with the Uber workload of 810 TPS to 900 TPS where each transaction invokes the computationally intensive Mobility Service DApp. Each blockchain is deployed in the `consortium` configuration of 200 geo-distributed machines and a cross indicates that the blockchain cannot run the Mobility Service DApp.

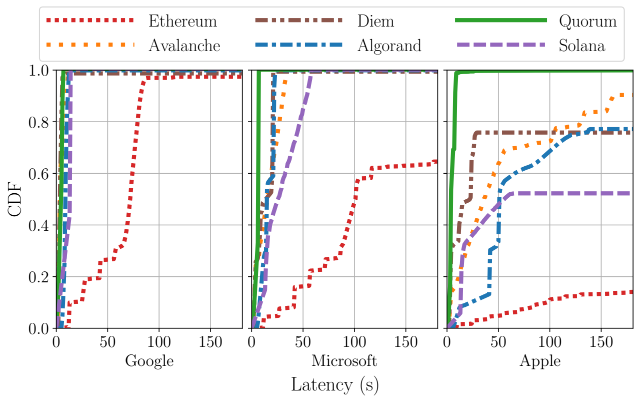


Figure 6. CDF of the transaction latencies for each blockchain when stressed with a peak load of 800 transactions (Google), 4000 transactions (Microsoft) or 10,000 transactions (Apple) followed by a low workload.

test whether the blockchains can handle a DApp with a potentially complex execution logic.

To test if a blockchain can execute arbitrary programs, we use the Mobility service DApp (§3), which is CPU intensive and generates a 810–900 TPS workload during 120 seconds. We test whether the blockchains can provide the service delivered by Uber by measuring the throughput and latency and verify that it matches the demand. As expected, this workload is more demanding than the aforementioned native transfer workload. We thus deploy the blockchains in the `consortium` configuration (see Table 3), which has the same number of machines and the same network as the `community` configuration but with more powerful machines.

Figure 5 shows the throughput and latency for each blockchain running the Mobility service DApp on the `consortium` configuration. When the blockchain is unable to execute the smart contract, Figure 5 shows an **X** letter instead. Algorand, Diem and Solana are unable to execute the DApp because the client reports an error of type `"budget exceeded"` indicating that the execution ran out of gas or timed out. This execution limit is hard-coded and cannot be lifted by paying a higher gas fee in the transaction. We conjecture that this limit is hard-coded to prevent a rich adversary from slowing down or completely stopping the blockchain by executing compute intensive tasks in smart contracts. Interestingly, the three blockchains able to execute the DApp use the `geth` implementation of the Ethereum Virtual Machine (EVM), which comes with no hard limit on gas budget of a transaction. Over these three blockchains, Quorum has the highest throughput of 622 TPS, which is close to the average workload, while the two other blockchains, Avalanche and Ethereum, have a throughput lower than 169 TPS.

6.5 Availability despite load peaks

We measure a very specific notion of the *availability* of a blockchain as its ability to commit submitted transactions in a timely manner even when stressed with load peaks. A blockchain is more available when it handles more intense bursts of transactions with low latency and without dropping any transaction. This property is desirable for a blockchain to handle realistic workloads where users are likely to send many transactions to the blockchain at the same time and expect to receive a confirmation from the blockchain within a reasonable delay. To measure the availability of the blockchains, we first deployed each blockchain in the `consortium` configuration (see Table 3) and then generates short bursts of transactions of varying intensities, extracted from the exchange DApp / Nasdaq workload (§3). In particular, we configured Diablo to evaluate the blockchains when sending separately the stock trade workloads of Google, Microsoft and Apple. Finally, we measured the proportion of dropped transactions and the latencies of committed transactions.

Figure 6 shows the cumulative distribution function (CDF) of the transaction latencies for all blockchains under these three workloads. Only Quorum commits all the transactions in the three workloads. Specifically, when stressed with the Apple workload, which consists of a load peak of 10,000 transactions during the first second, Quorum commits all transactions, among which 91% of the transactions are committed with a latency of 8 seconds or less. Interestingly, Quorum commits its transactions with similar latencies of 7 seconds or less when stressed with lower load peaks. Quorum uses IBFT, a deterministic BFT consensus that was historically designed to never drop a client request. We conjecture that this design choice is still present in the Quorum blockchain as we already mentioned in §6.3.

The other blockchain based on a deterministic BFT consensus, Diem, only commits 75% of the transactions, all of them in less than 30 seconds. Diem drops transactions during the load peak because of the limited size of the mempool on each blockchain node. While this dropping mechanism prevents Diem from committing all transactions during high load peaks, it also makes it less prone to completely collapse during constant loads, as opposed to Quorum (see §6.3). Algorand and Solana also drop transactions, as shown by their CDF plateauing at 77% and 52% of committed transactions, respectively, whereas Avalanche and Ethereum keep committing transactions until the end of the experiment. While it takes up to 162 seconds for Avalanche to commit some of the transactions, this blockchain manages to commit 90% of the submitted transactions. Despite its low throughput, Avalanche is the second blockchain to commit the most transactions.

As opposed to the Apple workload, the Google workload presents an initial load peak of 800 transactions during the first second. As a result, all the blockchains commit more than 97% of the Google workload transactions. In addition, all the blockchains but Ethereum commit all the Google workload transactions in less than 14 seconds while Ethereum does it in 118 seconds. The Microsoft workload has a moderate load peak of 4000 transactions during the first second. On this workload, while all blockchains but Ethereum commit all transactions, they take more time to do so with the exception of Quorum, which commits all of its transactions with a latency of 7 seconds. Specifically, Solana has its maximum latency rising from 1 second for the Google workload to 59 seconds while Algorand, Avalanche and Diem have their maximum latency going from 10-14 seconds to 22-37 seconds. On the Microsoft workload, Ethereum commits only 64% of the transactions.

6.6 Discussion

In this section, we summarize the key results of the evaluation (§6). Although the evaluated blockchains are not yet ready to handle demanding workloads found in centralized services, our in-depth analysis identifies key factors of performance and shows that some blockchain promises are fulfilled.

In §6.2, it appears that a blockchain using eventual consistency, like Solana, scales more easily to networks with many nodes. A decent throughput is also achieved by Quorum, a blockchain based on long studied consensus protocols but which also benefits from modern engineering techniques. More importantly, two blockchains, Diem and Avalanche, fail at using more challenging configurations, most likely because they simply do not consider these configurations as a use case: high RTT networks for Diem and large hardware resources for Avalanche.

In §6.3, the two blockchains using deterministic BFT consensus protocols, namely Quorum and Diem, are the most impacted by constantly high workloads. It could be due to their leader-based BFT consensus protocol design that is typically known to suffer from scalability limitations [19]. As an example, Smart Red Belly Blockchain, which builds upon a leaderless deterministic BFT consensus protocol, was recently shown to perform well under high workloads [40]. The Algorand, Avalanche and Solana blockchains, which offer probabilistic or eventually consistent guarantees, maintain a non negligible throughput when stressed with high constant workloads.

In §6.5, Quorum and Diem, the least robust blockchains in the face of peak loads are also the blockchains committing the largest portion of transactions under reasonable delay. This seems to indicate that there is a tradeoff between robustness and availability. In §6.4, only the three blockchains using the `geth` implementation of the EVM, Avalanche, Ethereum and Quorum, execute smart contracts with a complex and computationally demanding logic. The other blockchains having a virtual machine with a hard limit on the computational cost of a transaction are unable to provide complex services.

7 Related Work

Hyperledger Caliper [3] 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 the future. Caliper provides pre-defined workloads, specifying the calling contract, functions and the rate of transaction sending over time. Unfortunately, all these workloads are synthetic and we are not aware of any pre-defined DApps with realistic workloads that can be used with Caliper.

Blockbench [15] is one of the most notable benchmarking frameworks for blockchains, as it supports a number of micro and macro benchmarks. An important feature of Blockbench is that it features the notable SmallBank [6] database benchmark and the YCSB [13] cloud benchmark. Aimed at private blockchains, Blockbench evaluates 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 evaluates blockchains using smart contract workloads from 2016. Smart contracts have since then matured into more complex forms of DApps, like decentralized exchanges. In addition, smart contract workloads at the time were not comparable to the demand of centralized applications. For example, after seven years of existence, Ethereum receives now 30× more transactions than at the end of 2016[13]. These workloads remain significantly lower than the demand of centralized applications, like on the stock exchange.

The variety of Ethereum adapted blockchains motivated the development of Chainhammer [23], 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 at measuring the throughput in extreme situations. Its design is specialized as 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 the analysis of transaction costs and block structures.

Most evaluations that were made on blockchains are ad hoc and do not aim at comparing very different blockchain designs on the same ground. Some blockchain evaluation [20] focused on comparing Ripple, Tendermint, Corda and Hyperledger Fabric to evaluate their scalability potential in the context of Internet of Things. Another experimental evaluation [33] compared the performance of Burrow, Quorum and Red Belly Blockchain to evaluate the performance of blockchains relying exclusively on Byzantine fault tolerant consensus protocols. A benchmark was also developed to compare Ethereum and Hyperledger Fabric on top of an emulated network [31].

A preliminary version of Diablo appeared earlier in a technical report [10]. It did not feature the diversity of DApps that are presented here, but was used to evaluate crash fault tolerant systems, like Hyperledger Fabric. In this paper, we instead focused our study on secure systems capable of tolerating some form of Byzantine failures.

8 Conclusion

We proposed a new benchmark suite called Diablo and presented the most extensive evaluation of blockchain performance to date. The results indicate that none of the tested blockchains can treat all requests from any of the realistic DApps we proposed: gaming, web service, exchange, mobility service, video sharing. Our in-depth analysis is however positive and outlines interesting design decisions that affect performance. We believe that Diablo will be instrumental in helping improve the current blockchain designs and evaluate blockchains in a more transparent manner.

Availability

The code of Diablo and its DApps is open source and can be found along with its documentation on its website at https://diablobench.github.io.

[13] https://etherscan.io/chart/tx

Acknowledgments

We wish to thank Harold Benoit for his contributions to the initial version of Diablo and Aymeric Bacuet for his contributions to the DApps development. We also wish to thank the Solana development team for confirming that c5.xlarge instances have insufficient resources to run Solana, the Diem development team for telling us that we could not speedup the creation of accounts, the Avalanche development team for confirming that Avalanche integrated the London update and the Algorand development team for confirming that the 1000+ TPS throughput they observed was the peak throughput obtained from load tests. This work is supported in part by the Australian Research Council and by the Innosuisse project (46752.1 IP-ICT). Vincent Gramoli is a Principal Investigator of the Algorand Centre of Excellence SIP. The work was done while Andrei Lebedev was located at the Technical University of Munich.

References

[1] 2021. Avalanche: Blazingly Fast, Low Cost, & Eco-Friendly. https://www.avax.network/ Accessed:2021-12-06.
[2] 2021. CoinMarketCap. https://coinmarketcap.com/ Accessed:2021-05-06.
[3] 2021. Hyperledger Caliper. https://hyperledger.github.io/caliper/ Accessed:2021-05-06.
[4] 2022. Nasdaq. https://www.nasdaq.com/ Accessed: 2022-20-04.
[5] 2022. Solana (SOL). https://help.coinbase.com/en/coinbase/getting-started/crypto-education/SOL Accessed: 2022-19-04.
[6] Mohammad Alomari, Michael Cahill, Alan Fekete, and Uwe Rohm. 2008. The Cost of Serializability on Platforms That Use Snapshot Isolation. In *2008 IEEE 24th International Conference on Data Engineering*. 576–585. https://doi.org/10.1109/ICDE.2008.4497466
[7] 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. 2018. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. In *Proceedings of the Thirteenth EuroSys Conference*.
[8] Avalanche. 2022. Build Ethereum dApps on Avalanche. Build Without Limits. https://www.avax.network/developers Accessed: 2022-21-03.
[9] Shehar Bano, Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, Francois Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. 2019. State Machine Replication in the Libra Blockchain. https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-\in-the-libra-blockchain.pdf Accessed: 2019-10-01.
[10] Harold Benoit, Vincent Gramoli, Rachid Guerraoui, and Christopher Natoli. 2021. *Diablo: A Distributed Analytical Blockchain Benchmark Framework Focusing on Real-World Workloads.* Technical Report 285731. EPFL.
[11] Abel Brodeur and Kerry Nield. 2018. An empirical analysis of taxi, Lyft and Uber rides: Evidence from weather shocks in NYC. *Journal of Economic Behavior & Organization* 152 (2018), 1–16.
[12] JPMorgan Chase. 2019. Quorum Whitepaper. https://github.com/ConsenSys/quorum/blob/master/docs/Quorum%20Whitepaper%20v0.2.pdf Accessed: 2020-12-04.

[13] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking Cloud Serving Systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. 143–154.

[14] Tyler Crain, Christopher Natoli, and Vincent Gramoli. 2021. Red Belly: a Secure, Fair and Scalable Open Blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (S&P'21)*.

[15] Tien Tuan Anh Dinh, Ji Wang, Gang Chen, Rui Liu, Beng Chin Ooi, and Kian-Lee Tan. 2017. BLOCKBENCH: A Framework for Analyzing Private Blockchains. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 1085–1100.

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

[17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proc. 26th Symp. Operating Syst. Principles*. 51–68.

[18] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. 2007. Youtube Traffic Characterization: A View from the Edge. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement (IMC)*. 15–28.

[19] Vincent Gramoli. 2022. Blockchain Scalability and its Foundations in Distributed Systems. Springer.

[20] Runchao Han, Gary Shapiro, Vincent Gramoli, and Xiwei Xu. 2019. On the Performance of Distributed Ledgers for Internet of Things. *Internet of Things* 10 (Aug 2019).

[21] Thomas Hay. 2021. Hyperledger Besu: Understanding Proof of Authority via Clique and IBFT 2.0 Private Networks (Part 1). https://consensys.net/blog/quorum/hyperledger-besu-understanding-proof-of-authority-via-clique-and-ibft-2-0-private-networks-part-1/ Accessed: 2022-09-05.

[22] Rotem Hemo. 2022. Interoperability, Speed, and On Chain Randomness. https://www.algorand.com/resources/blog Accessed: 2022-11-09.

[23] Andreas Krüeger. 2017. Chainhammer: Ethereum benchmarking. https://github.com/drandreaskrueger/chainhammer Accessed:2021-05-06.

[24] Solana labs. 2022. Solana confirmations. https://github.com/solana-labs/solana/blob/master/programs/vote/src/vote_state/mod.rs#L34 Accessed: 2022-20-04.

[25] Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafał Malinowsky, and Jed McCaleb. 2019. Fast and Secure Global Payments with Stellar. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*. 80–96.

[26] Silvio Micali. 2020. Algorand 2021 Performance. https://www.algorand.com/resources/algorand-announcements/algorand-2021-performance Accessed: 2022-21-03.

[27] Christopher Natoli and Vincent Gramoli. 2017. The Balance Attack or Why Forkable Blockchains are Ill-Suited for Consortium. In *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*. 579–590.

[28] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Jorge Esteves Veríssimo. 2019. *Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure*. Technical Report 1908.08316. arXiv. http://arxiv.org/abs/1908.08316

[29] Rocky Rock. 2022. Avalanche. What is transactional throughput? https://support.avax.network/en/articles/5325146-what-is-transactional-throughput Accessed: 2022-02-05.

[30] Team Rocket. 2018. *Snowflake to Avalanche: A Novel Metastable Consensus Protocol Family for Cryptocurrencies*. Technical Report. https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV Accessed on 2021-11-26.

[31] Dimitri Saingre, Thomas Ledoux, and Jean-Marc Menaud. 2020. BCTMark: a Framework for Benchmarking Blockchain Technologies. In *2020 IEEE/ACS 17th International Conference on Computer Systems and Applications (AICCSA)*. 1–8.

[32] Roberto Saltini and David Hyland-Wood. 2019. *IBFT 2.0: A Safe and Live Variation of the IBFT Blockchain Consensus Protocol for Eventually Synchronous Networks*. Technical Report 1909.10194. arXiv.

[33] Gary Shapiro, Christopher Natoli, and Vincent Gramoli. 2020. The Performance of Byzantine Fault Tolerant Blockchains. In *Proceedings of the 19th IEEE International Symposium on Network Computing and Applications (NCA'20)*. 1–8.

[34] Solana. 2022. History. https://docs.solana.com/history Accessed: 2022-21-03.

[35] Solana. 2022. Validator Requirements. https://docs.solana.com/running-validator/validator-reqs Accessed: 2022-16-09.

[36] Statista. 2021. Number of peak concurrent Steam users from January 2013 to September 2021. https://www.statista.com/statistics/308330/number-stream-users/ Accessed: 2022-20-04.

[37] Statista. 2022. Hours of video uploaded to YouTube every minute as of February 2020. https://www.statista.com/statistics/259477/hours-of-video-uploaded-to-youtube-every-minute/ Accessed: 2022-20-04.

[38] Statista. 2022. Number of rides Uber gave worldwide from Q2 2017 to Q4 2020. https://www.statista.com/statistics/946298/uber-ridership-worldwide/#:~:text=In%20the%20fourth%20quarter%20of,percent%20year%2Don%2Dyear Accessed: 2022-20-04.

[39] Steam. 2013. Dota 2. https://store.steampowered.com/app/570/Dota_2/ Accessed: 2022-20-04.

[40] Deepal Tennakoon and Vincent Gramoli. 2022. *Smart Red Belly Blockchain: Enhanced Transaction Management for Decentralized Applications*. Technical Report 2207.05971. arXiv.

[41] Gavin Wood. 2015. ETHEREUM: A Secure Decentralised Generalised Transaction Ledger. Yellow paper.

[42] Anatoly Yakovenko. 2019. Tower BFT: Solana's High Performance Implementation of PBFT. https://medium.com/solana-labs/tower-bft-solanas-high-performance-implementation-of-pbft-464725911e79 Accessed: 2022-09-05.

[43] Anatoly Yakovenko. 2021. Solana: A new architecture for a high performance blockchain v0.8.13. https://solana.com/solana-whitepaper.pdf Accessed: 2021-12-06.

[44] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*.

A Artifact Appendix

A.1 Abstract

In this appendix, we present the artifact as the software, scripts and documentation that were submitted to run the Diablo framework. This artifact, also available at https://diablobench.github.io/artifact, was judged available and functional by the artifact evaluation committee. After discussion with the committee, we concluded that the scale of some of our experiments is beyond the scope of what this committee can reasonably reproduce.

A.2 Description & Requirements

To run Diablo easily, we provide a VirtualBox image with all dependencies as indicated in a screencast. We also provide steps in order to do a fresh install on the geo-distributed machines specified in the paper.

A.2.1 How to access.

All the code necessary to reproduce our experiments is available publicly online. In particular, it contains: The Diablo benchmarking framework source code, including every decentralized application benchmark and their associated realistic workloads. A set of scripts called minion that helps deploying the code to the availability zones of Amazon Web Services where our paper deployed the evaluated blockchains. The links to the source code of each of the blockchains we evaluated:

- Algorand (https://github.com/algorand)
- Avalanche (https://github.com/ava-labs/avalanchego)
- Diem (https://github.com/diem/diem)
- Ethereum (https://github.com/ethereum/go-ethereum)
- Quorum (https://github.com/ConsenSys/quorum)
- Solana (https://github.com/solana-labs/solana)

In order to simplify the tests, we made a VirtualBox image, step-by-step markdown documentation and a screencast available at https://diablobench.github.io/redo-howto.

A.2.2 Hardware dependencies.

We recommend using an x86-64 architecture processor to simplify the evaluation. We used a hardware machine with Intel Core i5 Comet Lake, 16 GB RAM, and at least 10 GB of free disk space on which we ran VirtualBox. For a fresh install, one can use as many as 200 machines, each with up to 36 vCPUs and 72 GiB of memory, spread across 5 continents.

A.2.3 Software dependencies.

VirtualBox, which is used in our screencast, does not support ARM processors, we thus recommend allocating to the virtual machine at least: 8 GB RAM, 4 vCPUs (cores) and 10 GB storage space. There is no additional software dependency required, besides VirtualBox, if you download the VirtualBox image to run Diablo. Otherwise, it is recommended to use Ubuntu OS and the following software as indicated in https://diablobench.github.io/fresh-install: `make`, `gcc`, `perl`, `perlbrew-5.34.0`, `pyenv 3.10.6`, `python`, `ssh`, `git`, `minion`, `diablo`.

A.2.4 Benchmarks. In addition to executing native transfers on the 6 blockchains, we also used realistic workloads: Dota 2, Fifa, NASDAQ, Uber, YouTube. They are separated from the source code and can be downloaded at the fresh install page.

A.3 Set-up (1h30min)

We present a simple setup that builds upon a VirtualBox image that allows to deploy the blockchain protocols, shows how to run two simple workloads and collect the results. We defer the instructions to run experiments on up to 200 virtual machines in 5 continents to the online page https://diablobench.github.io/fresh-install.

The setup consists of downloading a VirtualBox image and following the step-by-step documentation at https://diablobench.github.io/redo-howto or the screencast at the same page to generate results in a simplified setting.

- Install VirtualBox.
- Download the image from https://diablobench.github.io/redo-howto.
- Start the image with VirtualBox with login and password as `vagrant`.
- Run the experiments on the blockchains with a simple native transfer workload, `workload-native-10.yaml`:

Listing 1. Running native transfers on the blockchains

```
cd minion
./bin/eurosys --skip-install workload-native
    -10.yaml setup.txt
```

- Run now the experiments with a DApp workload, `workload-contract-10.yaml`:

Listing 2. Invoking smart contracts on the blockchains

```
./bin/eurosys --skip-install workload-contract
    -10.yaml setup.txt
```

Note that the workloads corresponding to each DApp of the paper are available as well. These experiments aggregate the performance results of all blockchains located in files of the name: `[blockchain_name]-[region_number]-[#secondaries]-[#blockchain_nodes]-[workload_name]-[timestamp].results.tar.gz`

For example, let us look at Algorand's performance results after running these experiments:

Listing 3. Retrieving the Algorand performance results

```
tar xf0 algorand-1-1-1-native-10_2022
    -08-21-22-48-58.tar.gz algorand-1-1-1-
    native-10_2022-08-21-22-48-58/
```

This command outputs the performance obtained by Algorand by inspecting the standard output log of the Diablo primary node that aggregated the results. For example, during the tests recorded in the screencast, we could see that 299 transactions were sent to Algorand, out of which 187 were successfully committed. As none were aborted, the rest of the transactions were pending. The average load was 10 transactions sent per second, and the average throughput was 6.3 transactions per second with an average latency of 12.2 seconds and a median latency of 11.4 seconds.

One can also inspect more detailed results by moving the results to the scripts folder to convert them from the JSON format to the CSV format:

Listing 4. Analyzing the performance results

```
mv *native*.tar.gz ~/scripts/results/native/
mv *contract*.tar.gz ~/scripts/results/
    contracts/
cd ~/scripts
```

```
./csv-results results results.csv
```

On each line of `results.csv`, we can now see the performance results of an archive for each given blockchain as depicted in the page https://diablobench.github.io/redo-howto#results-specification. The latencies are expressed in seconds and follow the transaction submission times. In the screencast example, the first submitted transaction for Algorand at time 0.10 second took 0.53 seconds to commit (first line).

A.4 Evaluation workflow

A.4.1 Major Claims. We enumerate here the major claims (Cx) of the paper.

- *(C1): We demonstrate that the performance of 6 state-of-the-art blockchains, including Algorand, Avalanche, Ethereum, Diem, Quorum and Solana is heavily dependent on the underlying experimental settings in which they are evaluated.*
- *(C2): Real DApps may not even execute successfully as some of their functions would consume more than the maximum allowed gas per transaction.*
- *(C3): The Algorand, Avalanche, Ethereum, Diem, Quorum and Solana blockchains are not capable of handling the demand of the selected centralized applications when deployed on modern commodity computers from individuals across the world.*

A.4.2 Experiments. We now describe how to verify the claims C1-C3 using the following experiments E1-E3:

Experiment (E1): The first experiment shows that the type of workload impacts the performance of a blockchain.
[Preparation] Follow up the instructions in Section A.3.
[Execution] Execute both workloads, with 10 transactions sent per second and 100 transactions sent per second:

Listing 5. Run different workloads

```
./bin/eurosys --skip-install workload-native
    -10.yaml setup.txt
./bin/eurosys --skip-install workload-100.yaml
    setup.txt
```

[Results] You can see that both workloads led to different set of results, for example in Algorand:

Listing 6. Compare the results

```
tar xfO algorand-1-1-1-native-10_2022
    -08-21-22-48-58.results.tar.gz algorand
    -1-1-1-native-10_2022-08-21-22-48-58.
    results/diablo-primary-127.0.0.1-out.log
tar xfO algorand-1-1-1-native-100_2022
    -08-21-23-08-04.results.tar.gz algorand
    -1-1-1-native-100_2022-08-21-23-08-04.
    results/diablo-primary-127.0.0.1-out.log
```

This confirms that different experimental settings lead to different results.

Experiment (E2): The second experiment shows that certain realistic DApps cannot be executed because of the gas per transaction cap.
[Preparation] Follow up the instructions in Section A.3.
[Execution] Execute Uber workload:

Listing 7. Run different workloads

```
./bin/eurosys --skip-install workload-uber.yaml
    setup.txt
```

[Results] The `out` log of Solana do not show any committed transactions while the `err` log indicates "Computational budget exceeded".

Listing 8. Compare the results

```
tar xfO solana-1-1-1-smart-uber_2022
    -08-21-22-48-58.results.tar.gz solana
    -1-1-1-smart-uber_2022-08-21-22-48-58.
    results/diablo-primary-127.0.0.1-out.log
tar xfO solana-1-1-1-smart-uber_2022
    -08-21-22-48-58.results.tar.gz solana
    -1-1-1-smart-uber_2022-08-21-22-48-58.
    results/diablo-primary-127.0.0.1-err.log
```

Similarly, the Algorand `err` log indicates "budget exceeded". This confirms that some blockchains cannot execute some realistic DApps due to transaction gas limit.

Experiment (E3): This experiment consists of observing that none of the tested blockchains can commit all transactions of any DApp workloads on a fresh install on 200 AWS VMs of type c5.2xlarge (8 vCPUs, 16 GB memory) spread equally across the following availability zones: Cape Town, Tokyo, Mumbai, Sydney, Stockholm, Milan, Bahrain, São Paulo, Ohio and Oregon. For more details, follow the fresh install described at https://diablobench.github.io/fresh-install.