BlockLite: Toward Accurate and Efficient Emulation of Public Blockchains in the Cloud

Xinying Wang, Abdullah Al-Mamun, Feng Yan, and Dongfang Zhao

University of Nevada, Reno, NV 89557, USA

Abstract. Blockchain is an enabler of many emerging decentralized applications in areas of cryptocurrency, Internet of Things, smart healthcare, among many others. Although various open-source blockchain frameworks are available in the form of virtual machine images or docker images on public clouds, the infrastructure of mainstream blockchains nonetheless exhibits a technical barrier for many users to modify or test out new research ideas in blockchains. To make it worse, many advantages of blockchain systems can be demonstrated only at large scales, e.g., thousands of nodes, which are not always available to researchers. This paper presents an accurate and efficient emulating system to replay the execution of large-scale blockchain systems on tens of thousands of nodes. In contrast to existing work that simulates blockchains with artificial timestamp injection, the proposed system is designed to be executing real proof-of-work workload along with peer-to-peer network communications and hash-based immutability. In addition, the proposed system employs a preprocessing approach to avoid the per-node computation overhead at runtime and thus achieves practical scales. We have evaluated the system for emulating up to 20,000 nodes on Amazon Web Services (AWS), showing both high accuracy and high efficiency with millions of transactions.

Keywords: Distributed systems · Blockchains · Consensus protocols.

1 Introduction

Blockchain, a decentralized and immutable database, has drawn a lot of research interests in various communities, such as security [22, 9], database [2, 14], network [16], distributed systems [29], and high-performance computing [1]. Although many existing blockchain frameworks [15, 20] are open-source and offer docker images accessible in major cloud vendors (e.g., Google Cloud, Amazon Web Services (AWS), and Microsoft Azure), there are yet more challenges for blockchains to be widely adopted, such as (i) the lack of resources to carry out large-scale experiments and (ii) much, if not prohibitive, engineering effort to modify sophisticated production (despite open-source) systems to timely test out new ideas.

To this end, multiple blockchain simulators were recently developed, two of the most popular ones being Bitcoin-Simulator and VIBES.

Bitcoin-Simulator [17] follows the same architecture and protocol of Bitcoin [7], the foremost application in cryptocurrency built upon blockchains. Users of Bitcoin-Simulator can specify various protocol and network parameters, such as the number of nodes and network bandwidth. The main goal of Bitcoin-Simulator is to study the trade-off between performance and security. Because of its design goal, Bitcoin-Simulator simulates the execution of a blockchain network at the block level rather than the transaction level. Bitcoin-Simulator does not provide a fine-grained control over the application, limiting its applicability for broader adoption. In addition, Bitcoin-Simulator simply inserts a series of static time stamps to simulate the proof-of-work (PoW) consensus protocol, which does not precisely characterize the behavior of real-world blockchain systems: for instance, Bitcoin dynamically adjusts the PoW difficulty and the nodes (as known as miners) usually complete the tasks in stochastic time intervals. Last but not least, Bitcoin-Simulator's network is built upon NS3 [26], a discrete-event network simulator, which limits the scalability on up to 6,000 nodes. Bitcoin network currently consists of more than 10,000 nodes [8], implying that Bitcoin-Simulator cannot simulate the entire network of Bitcoin as of the writing of this paper.

VIBES [28] extends Bitcoin-Simulator with the following improvements. First, VIBES supports a web-based interface for users to visually track the growth of the network. Second, VIBES improves the scalability of Bitcoin-Simulator by employing a fast-forwarding algorithm, which essentially designates a coordinator to control the events according to existing nodes' best guess on the block creation time. Such a centralized coordinator might be acceptable for a single-node simulator at small- or medium-scale, and yet could be a performance bottleneck for extreme-scale applications. Similar to Bitcoin-Simulator, VIBES takes the same approach of inserting time stamps to hypothetically carry out the PoW workload. Both Bitcoin-Simulator and VIBES are coarse estimators of real-world blockchain executions due to the lack of real PoW implementations or a decentralized architecture.

This paper presents **BlockLite**, the very first blockchain *emulator* with both high accuracy and high scalability. In contrast to Bitcoin-Simulator, BlockLite comprises a specific module to execute real PoW workload[1], supports fine-grained transaction management, and scales out to 20,000 nodes thanks to its efficient network communications built upon distributed queues along with PoW preprocessing that incurs negligible runtime overhead. Different than VIBES, BlockLite is fully decentralized with no single point of failure or performance bottleneck. It should be noted that even on a single node the decentralization philosophy of blockchains still holds for a blockchain *emulator* because each user-level thread is now considered as an individual node.

The remainder of this paper is organized as follows. §2 reviews important literature of blockchain systems. We describe the system design of the proposed emulator BlockLite in §3. §4 presents the implementation details and the inter-

[1] Thus making it an *emulator* rather than a simulator

face exposed to the users. We report the experimental results in §5, discuss some open questions in §6, and finally conclude the paper in §7.

2 Related Work

Several researches on blockchain are being under focus among the distributed computing community apart from the main stream blockchain systems like Bitcoin [7], Ethereum [15], and Hyperledger [20]. In order to mitigate the bottleneck with the storage a blockchain framework named Jupiter [18] is designed for mobile devices. Similarly, to alleviate storage bloating problem another framework [13] is proposed based on Network Coded Distributed Storage. To enable customization and enhancement in arbitrary scenarios Inkchain [21] is designed that is built with the flavor of permissioned blockchain based on Hyperledger.

Reliability and security in order to maintain data integrity is being considered another major concern for the distributed ledger technology. BigchainDB [6] is known to have all the features from database (i.e., indexing, querying structured data) and the blockchain properties (i.e., decentralization and immutability) while providing better fault tolerance. To improve the security at the Transport layer, Certchain [10] is proposed. Smart contract technology is leveraged in [19] to make sure the validity of data based on decentralized privacy preserving search scheme. Similarly, 2LBC [3] is designed to manage the data integrity in distributed systems based on leader rotation approach.

Distributed data provenance [11, 12] has been another research attraction among the file system and database communities. Most recently, a consensus protocol called proof-of-reproducibility (PoR) [1] is crafted to manage distributed in-memory ledger for HPC systems in order to support scientific data provenance. For storage-level provenance several I/O optimization techniques [11, 12] for file systems are proposed. There is also an emerging trend for conventional workloads, such as high-performance computing and networking, to move to the cloud platforms [24, 23, 31, 30]. Various solutions [5, 25] are designed for the improvement of provenance in database transactions.

Though, a recent work namely SimBlock [4] focuses to develop a blockchain network simulator that supports changing node behavior on run time in order to investigate behavior of nodes; to the best of our knowledge, this paper presents a blockchain framework for the first time that supports emulation with very large scales of nodes in terms of user level threads along with the plug-in facility of custom components (i.e., ad-hoc consensus protocols) for domain specific applications.

3 System Design

The objective of BlockLite is to provide blockchain researchers and practitioners an easy-to-use and lightweight emulator to develop new components and evaluate new ideas, such as ad-hoc consensus protocols customized for domain-specific applications. To achieve that objective, BlockLite is designed to be deployed to a single node, with loosely-coupled components for flexible customization.

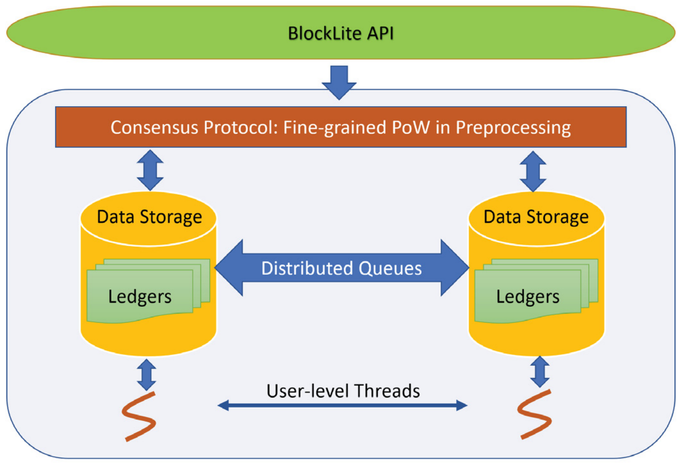


Fig. 1. BlockLite Architecture.

In its infrastructure, BlockLite has implemented all the building blocks for a basic blockchain system. This section details how these common facilities are designed, the challenges we encounter, and the approaches we take to build up the emulator.

The high-level architecture of BlockLite is illustrated in Figure 1. While the interface, i.e., BlockLite API, will be detailed in §4, the infrastructure can be broken down into three categories: storage, computation, and networking. In the context of blockchains, they are usually referred to as *distributed ledgers*, *consensus protocols* (e.g., PoW), and *network communications*.

Distributed Ledgers. The transaction data of a specific blockchain are replicated, either fully or partially, in distinct files each of which is associated to a hypothetical node. The data, also called ledgers, could have been partially duplicated if the following two conditions are satisfied: (i) more than 50% of nodes have agreed on that the new block (of transactions) is valid and (ii) the current node (other than those nodes who have voted) does not process any request regarding the new block. Regardless of specific consensus protocols, a blockchain requires only 50% votes supporting the new block's validity.

Consensus Protocols. A basic proof-of-work protocol is implemented from scratch in BlockLite. In contrast to other simulators where the difficulty is *simulated* by time delay and timestamps, BlockLite, as an *emulator*, conducts the real PoW workload by solving the puzzle. The puzzles we define in BlockLite are similar to the Nakamoto protocol in Bitcoin [7] in the sense of comparing blocks' hash values against predefined thresholds. Nonetheless, BlockLite exhibits an additional feature that preprocesses PoW allowing for fine tuning of puzzle difficulty, which is detailed in §3.1.

Network Communications. It is one of the most challenging components to emulate the networking in BlockLite that is designed to be working on a single node. Fortunately, BlockLite is designed for emulating public blockchains that

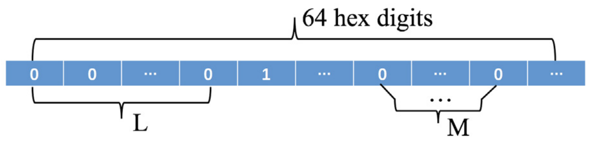


Fig. 2. Two-phase Puzzle for Efficient Preprocessing of PoW in BlockLite.

are based on PoW, which is compute-intensive rather than network-intensive[2]. Therefore, the real network impact for PoW-based blockchains lies in the network infrastructure's latency rather than bandwidth. BlockLite applies a statistical estimation of time delays for transmitting the messages between nodes, each of which is emulated by a user-level thread whose requests are buffered in a distributed queue. We will discuss the distributed queue in more detail in §3.2.

3.1 PoW Preprocessing for Fine-grained Calibration across Heterogeneous Systems

One cornerstone of Nakamoto consensus protocol, or any PoW variants, is the puzzle-based winner selection:[3] the hash value of the (block of) transactions is compared against the predefined "small" number. Because BlockLite is designed to be running on an arbitrary node that can be heterogeneous case by case, we must provide an efficient yet flexible mechanism to ensure the compatibility across heterogeneous machines. To this end, we design BlockLite puzzles as follows. A puzzle's difficulty is expressed by two sub-fields, L and M, in the form of $L.M$ (assuming SHA256 [27] is used as the hash function): L indicates the required number of leading zeros in the 64-hex (i.e., 256-bit) hash value; M indicates the minimal number of zeros in the middle of the hash value.

Figure 2 illustrates how $L.M$ is constructed. The first part L is semantically equivalent to the Nakamoto protocol: checking whether a hash value is smaller than a predefined threshold is essentially the same to counting the number of leading zeros in the binary or hex form of the hash value. L is a coarse-level adjustment of difficulty because the same L might imply a wide spectrum of computation time, and this is exactly why Bitcoin dynamically adjusts the difficulty every 2016 blocks [7]. To address that, BlockLite introduces the M part to allow the system to check whether there are M zeros in the middle of the hash value satisfying the following conditions: (i) Any leading zeros in L are not considered; (ii) Tailing zeros, by definition, are counted towards M; and (iii) Zeros need not be continuous.

The benefit of the additional M-zero checking is that we can adjust the puzzle difficulty under the same meta-difficulty, i.e., same L but different M's. In addition, M is positively correlated to the puzzle difficulty: a larger M implies more computation time. To see this, we can think of a larger M representing a

[2] Private blockchains are indeed network-intensive due to the quadratic number of messages.

[3] As known as "leader" in the context of distributed systems

super-set of the sets of less zeros with smaller M's. As a consequence, a smaller M has a higher chance to meet the requirement—the difficulty is lowered.

While the flexibility is significantly improved, one limitation of this $L.M$ two-phase puzzle is that the two arbitrary difficulty numbers do not follow partial orders in terms of computation time. That is, if $T(\cdot)$ indicates the computation time of a specific difficulty, it is possible that

$$T(L_1.M_1) > T(L_2.M_2) \texttt{ and } L_1 < L_2,$$

if M_1 is significantly larger than M_2. The root cause of this counter intuition is that L and M are, essentially, incomparable. For instance, if L is much smaller than M, then finding out M zeros, despite from random positions, is still much harder than locating a few leading zeros. As an extreme case, if we have two setups as $L_1.M_1 = 1.63$ and $L_2.M_2 = 2.1$, obviously the former case is much harder where we will seek for a hash value with all 64 zeros, as opposed to finding a hash value with two leading zeros and another zero from any of the remaining 62 hex digits.

3.2 Optimization for Extreme-scale Networking through Distributed Queues

In contrast to existing blockchain simulators, BlockLite does not simply insert timestamps for the the completion of PoW; instead, it solves the real puzzle to accurately *emulate* a real blockchain system. The downside of this approach is the cost and overhead for large-scale systems. For instance, Bitcoin has about 10,000 mining nodes as of January 2019 [8]. A single machine, despite its multi- or many-cores, is not able to efficiently emulate tens of thousands of nodes each of which works on a compute-intensive puzzle such as finding out a qualified hash value.

BlockLite overcomes the scalability challenge by delegating one node (thread) to solve the puzzle in a preprocessing stage and when the real application runs at a specific difficulty, the assigned nodes (or, threads) simply replicate the behavior of the delegation node. In doing so, BlockLite achieves the best of both worlds: real execution of PoW and low overhead (i.e., high scalability). Since the calibration is carried out in a preprocessing state, no runtime overhead is introduced.

The second technique taken by BlockLite to achieve high scalability is the usage of queue-based network communication. Specifically, we implement a priority queue who manages all the events in the order of their creation time. That is, the head of the queue always points to the earliest event, followed by later events each of which is requested by a specific node. Therefore, the queued events implicitly determine the the orders of nodes completing their tasks (e.g., submitting transactions, solving puzzles, appending blocks), which significantly reduces the network traffic.

4 Implementation and Interface

BlockLite is implemented in Java with about 2,000 lines of code. We have been maintaining a website for the BlockLite project at `https://hpdic.github.io/`

`blocklite`; the source code will be released to `https://github.com/hpdic/blocklite`.

Because users' machines are equipped with different resources, the very first step to deploy BlockLite is to calibrate the parameters in accordance to the system's specification. For instance, a throughput of 10 transactions per second might require 7.x difficulty on a high-end server with 32 cores, and the same throughput might require 4.x difficulty on a mainstream laptop with four cores. The calibration, also called PoW preprocessing, is to allow BlockLite to adjust the difficulty by considering factors input by users (e.g., expected throughput, consensus protocols) as well as system specification (e.g., number of cores, memory size).

When BlockLite runs for the first time, it generates a `difficulty-time` map between difficulty levels and the execution time. This map is implemented as a HashMap and is accessible to all the nodes. Whenever a node is waken up according to the consensus protocol, the node will consult with the difficulty-time map and replay the behavior with controlled randomness.

Specifically, the emulator starts by asking the user to specify the values of two parameters: MAX_DIFF and MAX_SUBDIFF. The emulator then goes on to repeatedly mine the blocks in a nested loop as shown in Algorithm 1. The complexity of the algorithm is $O(n^2)$ by observing the two levels of loops, one for the main difficulty and the other for the sub difficulty.

Algorithm 1 Calibration

```
1: function runMillsOfDiffcults
2:     $mainDifficult \leftarrow 1$
3:     for $i \in \{0 \cdots MAX\_DIFF\}$ do
4:         $subDiffcult \leftarrow 0$
5:         for $j \in \{0 \cdots MIN\_DIFF\}$ do
6:             Start Timer
7:             $powProof \leftarrow newProofWork(i, j)$;
8:             mineBlock();
9:             End Timer
10:        end for
11:    end for
12: end function
```

Algorithm 2 illustrates the mechanism of BlockLite to mine a specific block. The main difficulty `mainDiff` corresponds to L in Figure 2, `subDiff` indicates the auxiliary difficulty corresponding to M, and `miner()` is the implementation method of mining. The overall complexity is therefore $O(n)$, where n indicates the number of attempts before we find out the qualified nonce number.

BlockLite provides an easy-to-use interface for users to plug in application-specific components. Listing 4 illustrates a simplified code snippet of the interface with two core methods. Users can implement both methods to inject customized consensus protocols. For instance, `generateProof` solves the puzzle; in PoW, this means to check many nonce numbers until the hash value of the combined data satisfies the condition. Therefore, when the node wants to create a new Block,

Algorithm 2 Mine Block

```
1: function mineBlock
2:     $nonce \leftarrow 0$
3:     $n \leftarrow mainDifficult + subDiffcultf$
4:     $target \leftarrow$ A string of $\lfloor mainDifficult \rfloor$ 0's
5:     while !blockID.startwith(target) or countOfZero(blockID) <n do
6:         nonce ++;
7:         $blockID \leftarrow calculateHash(lastBlockID + timeStamp + nonce + ...)$;
8:     end while
9: end function
```

the program will call `mineBlock()` in block.java to solve the puzzle, i.e., select the appropriate difficulty to control the mining time according to the difficulty calibration map.

```java
public interface Provable {
    public boolean verifyProof(Block);
    public String generateProof(Block);
}
```

Listing 1.1. BlockLite Plug-in Interface.

Both aforementioned methods take as input a `Block` object, whose fields are explained in Table 1. The class comprises all the necessary information for the system to manipulate the blocks and more important, the transactions—the dominant data format in blockchain-based applications. A more detailed declaration of the class can be found in the source code, which is open-sourced at the project website: `hpdic.cse.unr.edu/blocklite`.

Table 1. Block Member Variable.

Variable	Definition
blockID	ID of the Block
creationTime	Creation Time of Current Block
creatorID	Creation ID (Node ID) of Current Block
parentBlock	Parent Block of Current Block
depth	Depth of Current Block
previousHash	Hash Value of Parent Block
childList	Child List of Current Block
numChild	Numbers of Current Block Children
txnList	Numbers of Current Block Transactions
proof	Consensus protocol (Nakamoto by default)

5 Experimental Evaluation

5.1 Experimental Setup

We perform extensive experimental evaluation of BlockLite on AWS. We pick four different instance types, the processors of which represent a wide spectrum

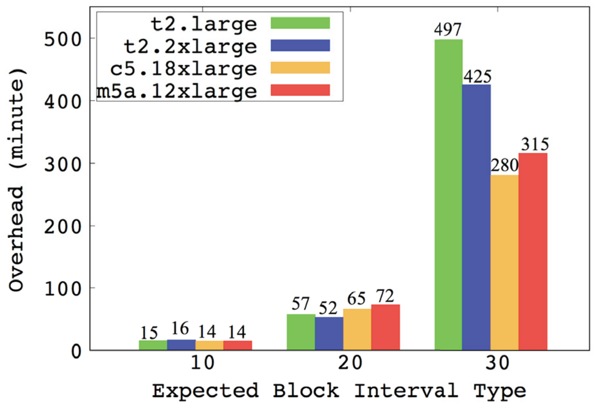


Fig. 3. Overhead of Various Instances.

of CPUs from both Intel and AMD. Table 2 lists the instance types along with their processor specification.

Table 2. Different AWS instances used for evaluating BlockLite.

Instance Types	CPU Specifications
t2.large	2 Intel(R) Xeon(R) CPU E5-2686 v4 @2.30GHz
t2.2xlarge	8 Intel(R) Xeon(R) CPU E5-2686 v4 @2.30GHz
c5.18xlarge	72 Intel(R) Xeon(R) Platinum 8124M CPU @3.00GHz
m5a.12xlarge	48 AMD EPYC 7571

The data we use for our evaluation are transactions in the same format of Bitcoin trades. Specifically, more than two million transactions are fed into the emulator for real-scale applications. We repeat all experiments for five times and report the average; we do not report the variance if it is unnoticeable.

5.2 Overhead

We report the overhead to find the map between the difficulty and the block-creation time in Figure 3. Specifically, we build the maps of four VMIs for the following three example intervals: 10 minutes, 20 minutes, and 30 minutes. It should be noted that, however, these overheads are incurred only during the preprocessing stage and would not be applied to the real-time execution.

We can observe that the overhead increases at an exponential rate with respect to the intervals; take the `t2.large` instance for example, the overhead increases by 3.7× from 10 to 20 minutes and then increases by 8.7× from 20 to 30 minutes. Another observation is that these VMIs do not exhibit much difference in overhead at shorter intervals like 10 or 20 minutes; and yet, for longer interval like 30 minutes, the smaller instances (i.e., `t2.large` and `t2.2xlarge`)

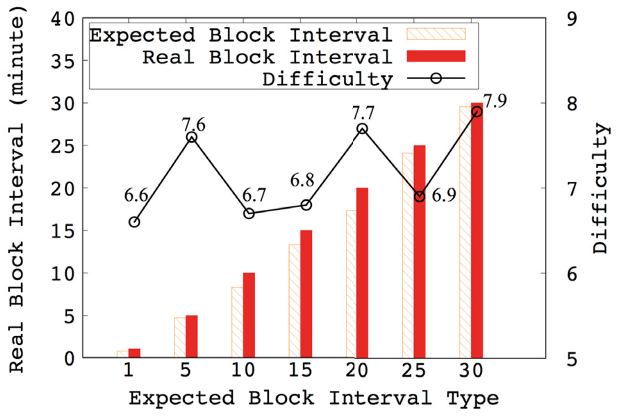


Fig. 4. Difficulty v.s. Block Creation Time

indeed incur much longer overhead (than `c5.18xlarge` and `m5a.12xlarge`). This phenomenon can be best explained by the fact that smaller VMIs are equipped with slower CPUs that need to solve the same puzzle in a longer time.

5.3 Accuracy

Figure 4 shows the calibration map of the *m5a.12xlarge* instance. For practical time intervals, e.g., ten minutes or more, the emulated processing times are close to the expected time intervals. We also plot the preprocessing difficulties in the figure; the result suggests that most difficulty values range in between six and eight, covering mining time from 1 to 30 minutes.

We then report the accuracy on various instance types in Figure 5. The for sake of clarity, we do not plot the corresponding difficulty in the figure. As we can see from the figure, not all instance types exhibit the same accuracy; in our tested VMIs, the `c5.18xlarge` seems to achieve the highest accuracy except for the trivial case of 1- and 5-minute intervals.

5.4 Sensitivity

This section evaluates the sensitivity of BlockLite when deployed to various VMIs. Figure 6 shows the puzzle-solving time on four different instances with respect to practical difficulties from four to eight. For clarity, we only compare the `main difficulty` between VMIs in the figure (we will report `sub difficulties` later on). The figure clearly shows that smaller instances (`t2.large`, `t2.2xlarge`) take more time than the larger instances for all difficulties, which, again, can be explained by the different CPU performance on these instances.

In Figure 7, we report the puzzle-solving time of both main and sub difficulties. Indeed, the highest computation time appears on the top-right corner of the map in all cases, as that corner represents both the highest main difficulty and the highest sub difficulty; similarly the lowest value appears at the bottom-left corner. Nonetheless, we do observe different gradients from these four heat-maps.

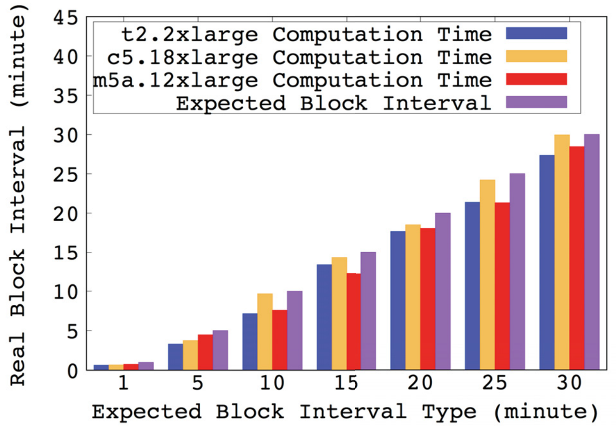


Fig. 5. Block Creation Rate with Various Instances Type.

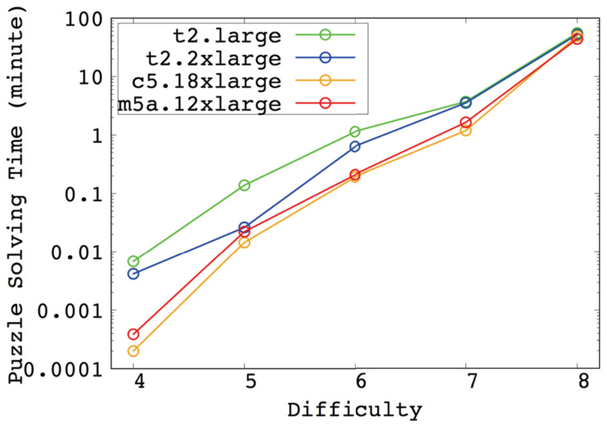


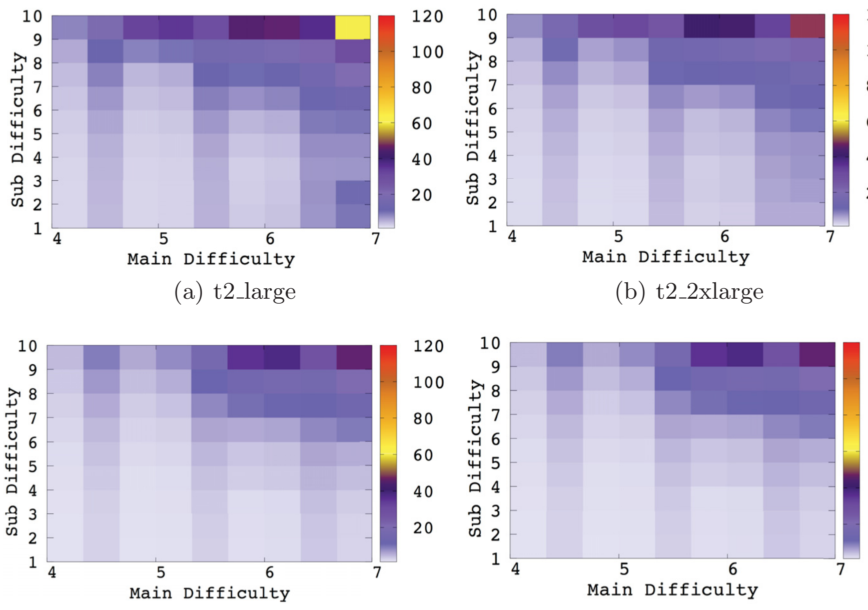
Fig. 6. Difficulty and Puzzle Solving Time

For instance, the largest instance (`m5a.12xlarge`) seems to have the most low-value cells (i.e., shorter puzzle-solving time).

5.5 Scalability

We tested BlockLite's scalability by emulating up to 20,000 nodes on the instance with 48 AMD EPYC cores and 192 GB memory (`m5a.12xlarge`, see Table 2). The workload is comprised of more than one million transaction data.

Figure 8 shows BlockLite's real-time executions, along with memory footprint, on 5,000, 10,000, 20,000, and 40,000 nodes, respectively. The puzzle difficulty is set to one for the sake of fast demonstrations. We can observe that even at the the real scale of Bitcoin—10,000 nodes, BlockLite can finish the emulation in 13 seconds with reasonable memory footprint of less than 4 GB.



(a) t2_large (b) t2_2xlarge

(c) c5_18xlarge (d) m5a_12xlarge

Fig. 7. Puzzle-solving time of both main and sub difficulties.

5.6 Monetary Cost

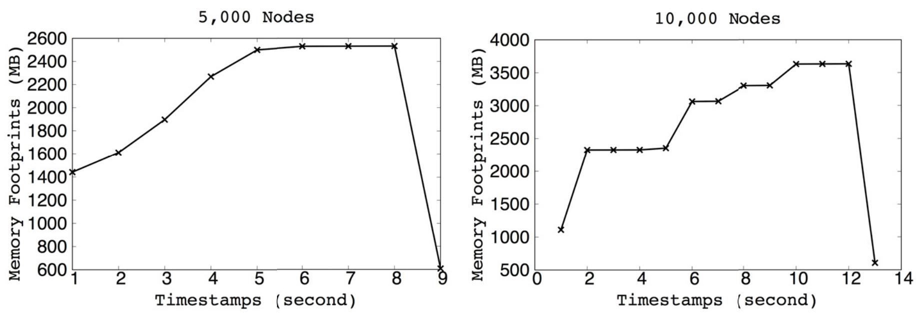
This section evaluates the cost incurred at the cloud computing vendors when various CPU or instance types are selected. As we can see in Figure 9, small instance `t2.large` incur the lowest cost in all intervals while the `c5.18xlarge` instance is the most costly one. This can be explained by the high unit price of the `c5.18xlarge` instance: although the execution time is comparable with the other large instance (`m5a.12xlarge`), the overall charge is higher because of the different unit price.

6 Discussion

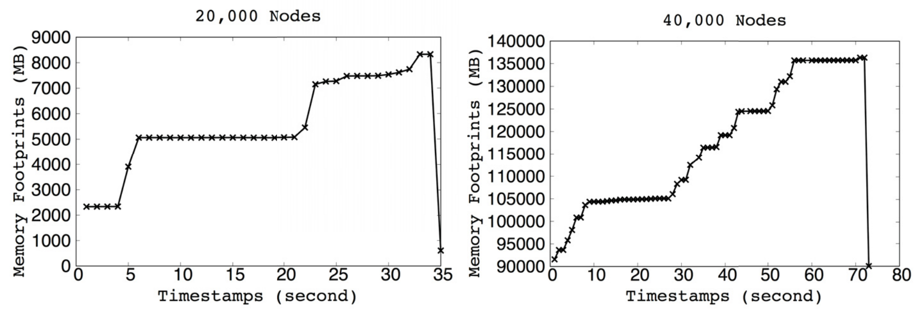
There are a few optimizations that could have been applied to current BlockLite design, which are not supported at the writing of this paper. Although there are many more open questions to be answered, we list two of most interesting optimizations here in the following for the sake of limited space.

One possible optimization to the current BlockLite implementation is to leverage the underlying multi- or many-cores in modern processors. It should be noted that in many computation-based consensus protocols, the workload is embarrassingly parallel. Therefore, we should be able to parallelize the proof-of-work (POW) protocols, which will be studied in our future work.

Another possible extension to this work is to implement an inter-node communication such that the emulator can be distributed over a cluster of nodes.



(a) 5,000 nodes (b) 10,000 nodes



(c) 20,000 nodes (d) 40,000 nodes

Fig. 8. Scalability and Memory Footprint of BlockLite.

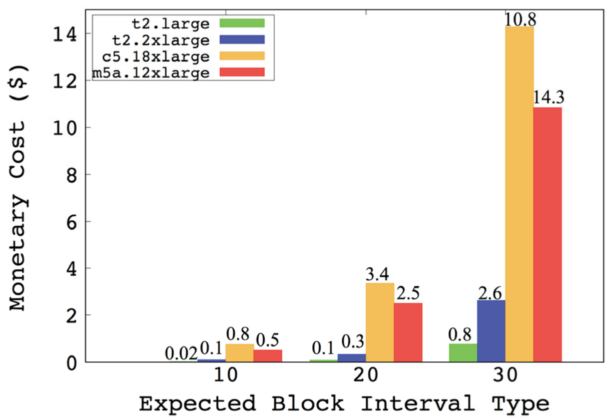


Fig. 9. Monetary Cost of Various Instances.

This would further improve the adaptability of BlockLite if a single node is not capable of conducting the intensive computation expected by some large difficulties.

7 Conclusion and Future Work

This paper presents BlockLite, the very first system who can emulate large-scale public blockchains on up to 20,000 nodes. BlockLite achieves such a high scalability through an offline calibration of PoW execution and distributed queues. The emulation also achieves high accuracy through a two-phase adjustment over the underlying consensus protocol. In terms of usability, BlockLite provides an easy-to-use interface to plug in application-specific components such as ad-hoc consensus protocols.

Future research directions include extending BlockLite into a general emulating platform applicable for a wider spectrum of blockchain types such as permissioned blockchains and field blockchains. The loosely-coupled design philosophy of BlockLite would likely emulate these non-public blockchain systems with high accuracy and high efficiency on par with public blockchains.

Acknowledgement

This work is in part supported by a Google Research Award and the National Science Foundation under contracts CCF-1756013 and IIS-1838024 (using resources provided by Amazon Web Services as part of the NSF BIGDATA program).

References

1. Al-Mamun, A., Li, T., Sadoghi, M., Zhao, D.: In-memory blockchain: Toward efficient and trustworthy data provenance for hpc systems. In: Proceedings of the 6th IEEE International Conference on Big Data (BigData) (2018)
2. Allen, L., et al.: Veritas: Shared verifiable databases and tables in the cloud. In: 9th Biennial Conference on Innovative Data Systems Research (CIDR) (2019)
3. Aniello, L., Baldoni, R., Gaetani, E., Lombardi, F., Margheri, A., Sassone, V.: A prototype evaluation of a tamper-resistant high performance blockchain-based transaction log for a distributed database. In: 13th European Dependable Computing Conference (EDCC) (2017)
4. Aoki, Y., Otsuki, K., Kaneko, T., Banno, R., Shudo, K.: Simblock: A blockchain network simulator. CoRR **abs/1901.09777** (2019)
5. Arab, B.S., Gawlick, D., Krishnaswamy, V., Radhakrishnan, V., Glavic, B.: Using reenactment to retroactively capture provenance for transactions. IEEE Transactions on Knowledge and Data Engineering (TKDE) **30**(3), 599–612 (March 2018)
6. BigchainDB: `https://github.com/bigchaindb/bigchaindb` (Accessed 2018)
7. Bitcoin: `https://bitcoin.org/bitcoin.pdf` (Accessed 2019)
8. Bitcoin Scale: `https://bitnodes.earn.com` (Accessed 2019)
9. Camenisch, J., Drijvers, M., Dubovitskaya, M.: Practical uc-secure delegatable credentials with attributes and their application to blockchain. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS). pp. 683–699 (2017)
10. Chen, J., Yao, S., Yuan, Q., He, K., Ji, S., Du, R.: Certchain: Public and efficient certificate audit based on blockchain for tls connections. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications (2018)
11. Dai, D., Chen, Y., Carns, P., Jenkins, J., Ross, R.: Lightweight provenance service for high-performance computing. In: International Conference on Parallel Architectures and Compilation Techniques (PACT) (2017)

12. Dai, D., Chen, Y., Kimpe, D., Ross, R.: Provenance-based object storage prediction scheme for scientific big data applications. In: IEEE International Conference on Big Data (BigData) (2014)
13. Dai, M., Zhang, S., Wang, H., Jin, S.: A low storage room requirement framework for distributed ledger in blockchain. IEEE Access **6**, 22970–22975 (2018)
14. Dinh, T.T.A., Wang, J., Chen, G., Liu, R., Ooi, B.C., Tan, K.L.: Blockbench: A framework for analyzing private blockchains. In: ACM International Conference on Management of Data (SIGMOD) (2017)
15. Ethereum: `https://www.ethereum.org/` (Accessed 2018)
16. Eyal, I., Gencer, A.E., Sirer, E.G., Van Renesse, R.: Bitcoin-ng: A scalable blockchain protocol. In: Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation (NSDI) (2016)
17. Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H., Capkun, S.: On the security and performance of proof of work blockchains. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS) (2016)
18. Han, S., Xu, Z., Chen, L.: Jupiter: A blockchain platform for mobile devices. In: IEEE International Conference on Data Engineering (ICDE) (2018)
19. Hu, S., Cai, C., Wang, Q., Wang, C., Luo, X., Ren, K.: Searching an encrypted cloud meets blockchain: A decentralized, reliable and fair realization. In: IEEE INFOCOM 2018 - IEEE Conference on Computer Communications (2018)
20. Hyperledger: `https://www.hyperledger.org/` (Accessed 2018)
21. Inkchain: `https://github.com/inklabsfoundation/inkchain` (Accessed 2018)
22. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE Symposium on Security and Privacy (SP) (2016)
23. Li, T., Keahey, K., Wang, K., Zhao, D., Raicu, I.: A dynamically scalable cloud data infrastructure for sensor networks. In: Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud) (2015)
24. Li, T., Zhou, X., Wang, K., Zhao, D., Sadooghi, I., Zhang, Z., Raicu, I.: A convergence of key-value storage systems from clouds to supercomputer. Concurr. Comput. : Pract. Exper. (2016)
25. Niu, X., Kapoor, R., Glavic, B., Gawlick, D., Liu, Z.H., Krishnaswamy, V., Radhakrishnan, V.: Provenance-aware query optimization. In: IEEE 33rd International Conference on Data Engineering (ICDE) (2017)
26. NS3: `https://www.nsnam.org/tutorials/NS-3-LABMEETING-1.pdf` (Accessed 2019)
27. SHA-256: `https://en.bitcoin.it/wiki/SHA-256` (Accessed 2018)
28. Stoykov, L., Zhang, K., Jacobsen, H.A.: Vibes: Fast blockchain simulations for large-scale peer-to-peer networks: Demo. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference (Middleware) (2017)
29. Zhang, K., Jacobsen, H.: Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In: 38th IEEE International Conference on Distributed Computing Systems (ICDCS) (2018)
30. Zhao, D., Mandagere, N., Alatorre, G., Mohamed, M., Ludwig, H.: Toward locality-aware scheduling for containerized cloud services. In: IEEE International Conference on Big Data (BigData). pp. 273–280 (2015)
31. Zhao, D., Yang, X., Sadooghi, I., Garzoglio, G., Timm, S., Raicu, I.: High-performance storage support for scientific applications on the cloud. In: Proceedings of the 6th Workshop on Scientific Cloud Computing (ScienceCloud) (2015)