HYPERLEDGER
BLOCKCHAIN TECHNOLOGIES FOR BUSINESS

Hyperledger Blockchain Performance Metrics

ABOUT HYPERLEDGER

Hyperledger (**http://hyperledger.org/**) is an open source effort created to advance cross-industry blockchain technologies. It is a global collaboration including leaders in banking, finance, Internet of Things, manufacturing, supply chains, and technology. The Linux Foundation, the nonprofit organization enabling mass innovation through open source, hosts Hyperledger. The Linux Foundation also enables a worldwide developer community to work together and share ideas, infrastructure, and code.

About This Paper

This is the first white paper from the Hyperledger Performance and Scale Working Group (**https://wiki.hyperledger.org/groups/pswg/performance-and-scale-wg**). The purpose of this document is to define the basic terms and key metrics that should be used to evaluate the performance of a blockchain and then communicate the results. This paper also serves as a platform-agnostic resource for technical blockchain developers and managers interested in using industry-standard nomenclature.

While we appreciate that there may be discrete definitions for the terms “blockchain” and “Distributed Ledger Technology (DLT),” for the purposes of this paper we will treat both terms synonymously and use the term “blockchain” throughout.

This document provides some guidance on selecting and evaluating workloads. We expect that refinements to these definitions and new blockchain-specific metrics will warrant future revisions of this document.

In future documents, the Working Group plans to discuss workloads in greater detail and to offer additional guidance on standard procedures and emerging best practices for evaluating blockchain performance. To provide your feedback and stay informed about subsequent versions of this paper, please join us in the Performance and Scale Working Group (**https://wiki.hyperledger.org/groups/pswg/performance-and-scale-wg**).

V1.01 published October 2018.

Acknowledgements
The Hyperledger Performance and Scale Working Group would like to thank the following people for contributing to this paper: Mahesh Adulla (Reddy), Salman Baset, Vipin Bharathan, Gordon Graham, Guy Hochstetler, Imre Kocsis, Todd Little, Dan Middleton, Mark Simpson, Harish Sukhwani, and Mark Wagner.

About the Performance and Scale Working Group (PSWG)
The goal of the Working Group is to ensure that the performance and scalability of all blockchain projects are measured in a fair and equitable manner using metrics that are defined, gathered, and reported in a consistent way. Note that the PSWG is open to all and we encourage you to join our efforts.

Introduction

While blockchains may appear similar to distributed databases, they are typically implemented without a central authority and central repository. Therefore blockchains provide some unique differences from everything that has come before. A blockchain survives faults and attacks by using redundant checking at multiple nodes. This resiliency goes far beyond replication, since it happens across the network without any central coordinator or intermediary.

In some other networked systems like web tiers, adding nodes serves to divide work among more resources and increase performance. This is often the reverse in blockchains, where more nodes increase the resilience of the system in terms of integrity and availability, generally at some expense to performance.

In Bitcoin, for example, the best-case scenario would be that the network would lose no performance as nodes are added.

Even this simple notion can be complicated when we look across the breadth of blockchains. Some blockchains specialize the roles of nodes. In those systems a “node” may no longer imply a unique participant (such as a company) with a share of the resiliency burden.

This makes measuring and comparing performance between different blockchains very difficult. To help precisely and consistently evaluate the unique performance attributes of blockchains, this paper defines many relevant terms and metrics, and discusses some complex issues.

When publishing the results of any blockchain evaluation, we strongly recommend that all reports include full details and qualifiers of all the terms, metrics, and issues outlined in this paper.

Performance evaluation vs. benchmarks

Performance evaluation is the process of measuring the performance of a system under test. This evaluation can cover system-wide measures such as response time or latency, or measure-specific activities such as the time to write a block to persistent storage.

The goal of any performance evaluation is to understand and document the performance of the system or subsystem being tested. This often involves measuring what happens when dependent variables are altered; for example, measuring the throughput of the system as the number of concurrent requests is varied.

Benchmarking is the process of making standard measurements to compare one system to another or to previous measurements of the same system.

The term **benchmark** originally referred to marks on workbenches used to measure standard lengths of materials, or notches cut into buildings used by surveyors to measure heights.

Performance benchmarks are a specific type of performance evaluation often formalized by standards bodies such as Standard Performance Evaluation Corporation (SPEC), Securities Technology Analysis Center (STAC), and Transaction Processing Performance Council (TPC). These organizations carefully define workloads used to drive the system under test and specific performance measurements required to be made during the test.

Informal benchmarks may be used to look for any effects on performance as changes are made to a system’s code or design. For example, you can measure the performance of version 3.4 of a system, and then use that measurement as a benchmark to see if performance is better or worse in version 3.5.

In general, benchmarking is more controlled than performance evaluation, since benchmarks tend to run in a more standardized environment with a well-documented workload. Without benchmarks, comparisons between widely disparate environments are not very meaningful.

This paper will focus on blockchain performance evaluation and the associated metrics, rather than on benchmarking. This is a necessary first step before the industry can develop any formal benchmarks. We expect that this Working Group will begin to define benchmarks for measuring blockchain performance at some point in the future.

1. Performance Evaluation Terms

Figure 1 shows a typical configuration for a blockchain performance evaluation. The Test Harness on the left shows the program and system(s) used to generate load against the System Under Test (SUT) on the right. Each of the terms in Figure 1 is defined in this section.

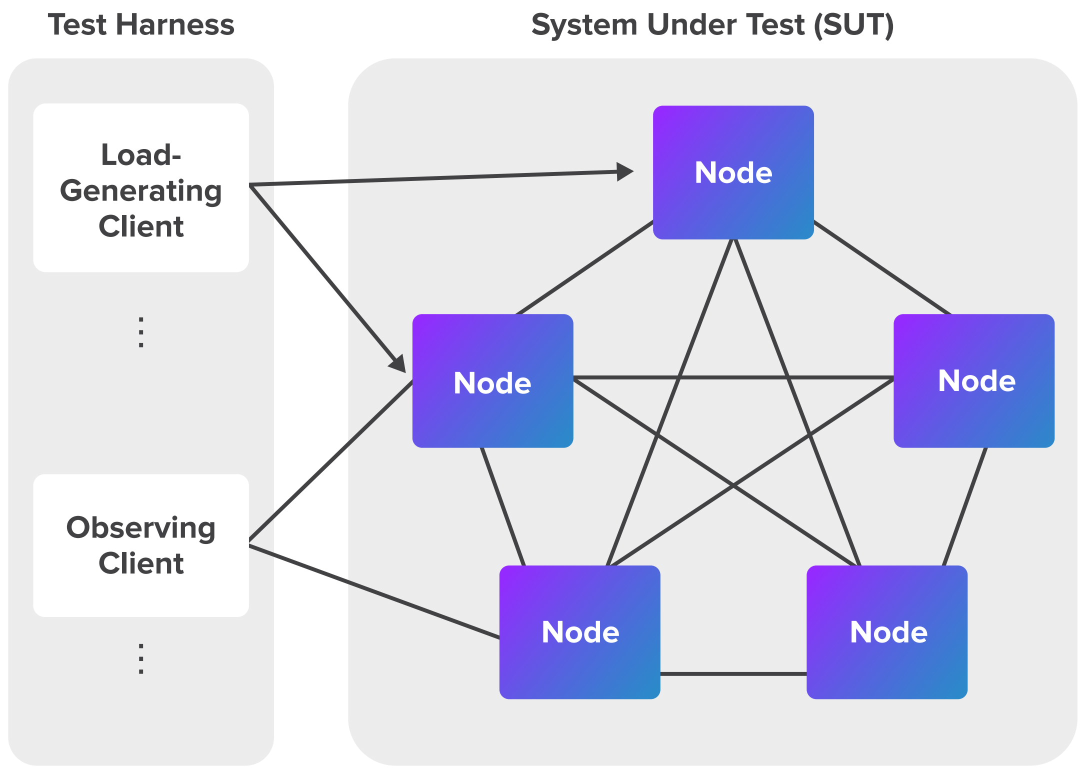

FIGURE 1: TYPICAL CONFIGURATION FOR A BLOCKCHAIN PERFORMANCE EVALUATION

Test Harness

The **test harness** is the hardware and software used to run the performance evaluation. This test harness will typically represent many clients that can inject workloads and make observations at any number of nodes.

Client

A **client** is an entity that can introduce work into a system or invoke system behaviors. In a blockchain system, there are often multiple types of clients at multiple levels.

For example, in a supply chain application using smart contracts deployed to a blockchain, the supply chain application is a client of the blockchain platform. However, further clients of the supply chain application will often interact with the application via a browser. Another client might be the user of an auditing component of the supply chain application who is notified of policy violations via an email or SMS message.

For the purposes of this paper, we will focus on the entities that directly interact with the blockchain platform, not on any applications built on top of those platforms.

A **load-generating client** is a node that submits transactions on behalf of the user to the blockchain network (SUT), most often following an automated test script. The interface between the client and the SUT can range from a simple Representational State Transfer

(REST) interface to a comprehensive Software Development Kit (SDK). Depending on the interface, a client can be either stateless or stateful.

An **observing client** is a node that receive notifications from the SUT or can query the SUT regarding the status of the submitted transactions, most often following an automated test script. There can be more than one observing client. The observing client cannot submit any new transactions.

System Under Test (SUT)

Defining the System Under Test (SUT) is a complex issue. For the purposes of this document, the SUT is defined as the hardware, software, networks, and specific configurations of each required to run and maintain the blockchain.

The SUT does not include the client APIs, any associated libraries, or any external caches or databases being used to accelerate read times.

Node

In the context of a blockchain network, a node is an independent computing entity that communicates with other nodes in a network to work together collectively to complete transactions.

A node is a virtual entity, in the sense that it could be running on physical hardware, or as a VM or containerized environment. In the latter case, a node could share physical hardware with other nodes in the same network.

A set of nodes may be managed by the same organization. For example, think of a mining pool operated by one company where a group of machines run as individual nodes doing proof of work (PoW) on a network.

In most blockchain networks (Bitcoin, Ethereum, Hyperledger Burrow, Hyperledger Indy, Hyperledger Iroha, Hyperledger Sawtooth) each node plays a uniform class of roles in the network, such as generating blocks, propagating blocks, and so on. In these networks, nodes are usually referred to as peers. Such networks might require one or more nodes to take on a temporary role as a leader. This leadership role may be passed to other nodes in certain well-determined conditions.

However, in some blockchain networks (such as Hyperledger Fabric), nodes are assigned one or more possible roles, such as endorsing peers, ordering services, or validating peers.

2. Blockchain Terms

Blockchain is a new technology, with nomenclature still taking shape and evolving every day. Although blockchains are a new form of distributed database, some database terms have slightly different meanings when used in a blockchain context. And many common software terms have new nuances when used to describe a blockchain. This section describes some of the most common terms used to discuss blockchain technology.

Consensus

In the context of blockchain, consensus is the distributed process by which a network of nodes provide a guaranteed unique order of transactions and validates the block of transactions.

For more details in the context of Hyperledger, please refer to the white paper on consensus, Volume I from the Hyperledger Architecture Working Group, available here: **https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf**.

Commit

In a traditional database, a **commit** refers to the point when a transaction is written to the database. A blockchain has the additional complication of defining when this happens across multiple nodes. Typically a block is considered the record of committed transactions, and those transactions are committed when the block has been circulated through the network and applied by all nodes. The rules for the agreement on that block are dictated by consensus, which may have unique rules for finality.

Finality

Finality means that once a transaction is committed, it cannot be reversed, i.e. the data cannot be rolled back to the previous state. Different blockchain systems may provide different types of finality. Typically, this is defined in the consensus protocol. Different types of consensus exist, such as voting-based consensus with immediate finality and lottery-based consensus with probabilistic finality.

For testing purposes, formal thresholds of finality probability should be set. See Appendix A for further specifics.

Network Size

Network size is the number of validating nodes participating in consensus in the SUT. Network size is meant to express the total number of nodes actively participating in the blockchain network that support the unique features of a blockchain.

This count should not include observer nodes, or other nodes not actively participating in both consensus and the validation of transactions.

In systems that separate consensus and validation logic into different nodes, the network size should be expressed as the smaller node count of the two. This is because the resilience of the system (the key aspect of blockchains) is limited by the lesser number.

For example, a system deployed with a single consensus node but many other types of nodes has a single point of failure. By our definition, such a network has a size of one.

Query

Querying is the ability to run ad-hoc operations or searches against the dataset contained within the blockchain. The SUT may not be built to execute these queries in a performant way.

Any off-chain databases, or caches that support efficient querying, can still be measured. But we assume that those are not standard parts of the blockchain. Therefore, query performance is beyond the scope of this version of this document.

Reads

Read operations differ from transactions in that there is no change to state. Reads can be an internal mechanism of a blockchain node, such as fetching data in the process of validating transactions. For the purposes of this document, we are only interested in defining measurements for external reads on the system, such as retrieving a transaction or querying a balance. Any internal reads will be an implicit part of the transactional measurements in any particular workload.

Many deployments use a system or tier beside a blockchain node to support more complex queries or to provide caching that improves read throughput. For the purposes of this document, we are only interested in the primitive fetching provided by the blockchain node itself.

Definitions of workloads should consider whether and to what degree “pure” read operations are intermingled with transactions. And all performance evaluation reports should disclose any external databases and how these are configured.

State and Global State

You can think of most blockchain systems as **state** machines, where each transaction or block of transactions is a state transition. The state itself is the contents of the database at a point in time. The log of transactions (the blockchain data structure) is not generally considered part of state, but simply the definitions of the transitions between states.

Global state is a state that is shared across all nodes. Note that not all blockchain systems implement global state. Some systems restrict information agreement for privacy or other reasons.

Transaction

A **transaction** is a state transition that changes data in the blockchain from one value to another. That data could represent an asset amount, several IoT sensor readings, or any other type of data tracked using a blockchain.

Transactions are typically proposed by **clients** and then evaluated by the blockchain system against a list of rules, sometimes called a **smart contract**. If valid, the system will **commit** the transaction, which makes the state change.

A transaction may fail the verification for one of the reasons listed in Appendix B. Some blockchain systems also record those invalid transactions in blocks, but most do not. When evaluating the performance of a blockchain, the measure of valid transactions is more meaningful. For this reason, the total number of invalid transactions should be subtracted when calculating **throughput**.

3. Definitions of Key Metrics

This section defines several common metrics that apply to blockchains, along with a mathematical formula where appropriate. See Appendix A for further discussion of latency in the blockchain context. And see Appendix C for any experimental metrics that are interesting but not yet mature enough to be included in the main body of this document.

Read Latency

Read Latency = Time when response received – submit time

Read latency is the time between when the read request is submitted and when the reply is received.

Read Throughput

Read Throughput = Total read operations / total time in seconds

Read throughput is a measure of how many read operations are completed in a defined time period, expressed as reads per second (RPS). This metric may be informative, but it is not the primary measure of blockchain performance. In fact, systems will typically be deployed adjacent to the blockchain to facilitate significant reading and queries.

Transaction Latency

Transaction Latency = (Confirmation time @ network threshold) – submit time

Transaction Latency is a network-wide view of the amount of time taken for a transaction's effect to be usable across the network. The measurement includes the time from the point that it is submitted to the point that the result is widely available in the network. This includes the propagation time and any settling time due to the consensus mechanism in place.

To account for both of those factors and give the network-wide view, the delay should be measured using all nodes in the SUT. Eyal et al provide a helpful definition for the amount of time for a percentage of the network to commit the transaction1, which we have simplified somewhat here. The latter part of this definition, the percentage of the network, is most significant for consensus protocols like PoW. In such systems a threshold like 90% may be desired, whereas in a non-probabilistic protocol like PBFT 100% is the only meaningful threshold.

This metric is computed per transaction, but in most cases reports should provide various statistics over all transactions like the average, high, low, and standard deviations.

Transaction Throughput

Transaction Throughput = Total committed transactions / total time in seconds @ #committed nodes

Transaction throughput is the rate at which valid transactions are committed by the blockchain SUT in a defined time period. Note that this is not the rate at a single node, but across the entire SUT, i.e. committed at all nodes of the network. This rate is expressed as transactions per second (TPS) at a network size.

Remember that the total number of invalid transactions should be subtracted from the total transactions to yield the total committed transactions. See Appendix A for further discussion of how to calculate different examples of latency.

4. Considerations for Blockchain Performance Evaluation

This section discusses several key considerations when designing any blockchain performance evaluation, including the test environment, observation points, transaction characteristics, workloads, and faultloads. To create a valid and reproducible design, developers must weigh these considerations carefully, record their decisions clearly, and report their results fully.

These characteristics should be noted as part of the test results, since revealing these details makes it easier to compare performance tests across platforms.

Test Environment

Test results should be independently reproducible. To support this goal, all environment parameters and test software, including any workload, should be documented and made available.

Here are some added considerations when assessing the environment in which performance testing will occur:

- **Consensus protocol.** Describe the consensus protocol used during the testing, such as RAFT, Practical Byzantine Fault Tolerant (PBFT), and so on.
- **Geographic distribution of nodes.** Describe the physical setup of the network. Are all nodes co-located or geographically dispersed? If dispersed, then describe how and where.
- **Hardware environment of all peers.** Indicate the processor speed, number of cores, memory, and so on.
- **Network model.** Any information about the complexity of the network employed between peers might reveal potential bottlenecks that could be highlighted. For example, understanding the number and type of firewalls to be traversed. Any sort of complex network components should be called out.
- **Number of nodes involved in the test transaction.** Some blockchain platforms broadcast transactions to all nodes, while others restrict the interaction to a subset of the nodes (e.g. Hyperledger Fabric Endorsers).
- **Software component dependencies.** Does the test or platform require any additional components for the platform or the test itself to function? All components should be described.
- **Test tools and framework.** Describe which tools were used, and what testing framework devised to generate load and capture results. Describe how the test load was driven, and the location of the client load driver relative to the platform node.
- **Type of data store used.** The data store affects performance, especially if both reads and writes are included. Different platforms offer pluggable choices for the underlying ledger data store, such as CouchDB, H2, Postgres, and the major vendor databases.
- **Workload,** the code used to produce and validate the transaction. See the subsection below for a further discussion of workloads.

These characteristics should be noted as part of the test results, since revealing these details makes it easier to compare performance tests across platforms.

Observation Points

Wherever possible, measurements should be taken from the perspective of the test harness which is ideally located outside of the SUT. This will be closer to the viewpoint of the end user. When this cannot be achieved, the measurement point should be clearly identified.

Consider spreading the workload entry points. Realistically, transactions represent heterogeneous interactions among diverse sets of nodes. Thus, an aspect of assessing scalability would be to observe how a platform improves or degrades when the workload injection is spread across different nodes, rather than from a single node entry point.

Transaction Characteristics

- **Complexity:** How compute-intensive is the logic of the smart contract?
- **Data access patterns:** Do the patterns of reads and writes model the production use?
- **Dependencies:** Do the transactions model the transaction and data dependencies of your production use?
- **Size:** How large are the transactions? This has implications for the propagation times across the network.

Workloads

A workload defines how the SUT is exercised. In any performance testing it is critical that the workload is representative of the actual production usage. For example, if a workload simply creates new entries in the database, but the production usage predominantly modifies existing entries, then the workload will not tell us anything about how to expect the system to behave in production.

There are multiple ways to design a workload. Two examples are replaying production transactions and making use of pre-existing database benchmarks and applying them to a blockchain environment.

We expect multiple workloads to exist that help evaluate different types of usages. Each of those workloads should be informed by and representative of some production usage. Defining any specific workload is outside the scope of this document.

Faultloads

All blockchains should be designed to provide ledger immutability, cryptographic authenticity, and tolerance against faults and attacks as core properties. Such features are enabled in the system by allocating extra resources. When such features are enabled or faults are present, the system's peak performance diminishes.

Since many blockchain networks are expected to operate in an environment where faults and attacks are ordinary events, the performance benchmarks should include faulty and even malicious transactions. Faultload represents a set of faults and stressful conditions that emulates real faults experienced in the field.[2]

Performance benchmarking activity should be conducted in an environment similar to the one in which the networks will operate. Even within the permissioned blockchain networks, the environment could vary ranging from closed small-scale networks to highly-distributed internet-deployed networks.

Appendix A: Latency Calculation Examples

This appendix describes three different cases that create different issues for measuring latency in blockchains. One case has immediate finality, and two have probabilistic finality with known and unknown topologies. Numbers are used for illustration purposes only and do not reflect the performance of any particular platform.

Case 1: SUT with Immediate Finality

Systems employing voting-based consensus have immediate finality. Once a transaction is committed, the state is guaranteed to be irrevocable. As with the remaining cases, the test process should measure commitment at each node to ensure full commitment across the distributed system.

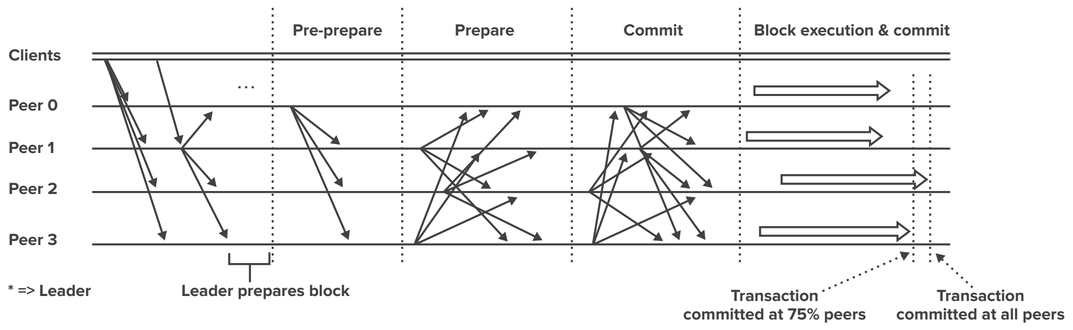

FIGURE 2: TRANSACTION FLOW FOR BLOCKCHAIN PLATFORMS USING PBFT CONSENSUS. SOURCE: https://github.com/tallharish/PerfMetricsPSWG/

Example: In a 10-node network running Min-BFT, 9 nodes commit a transaction after 11 seconds. The tenth node has a higher latency connection to the leader and takes an additional second. This transaction has an 12-second delay at 100% of the nodes.

Case 2: SUT with Probabilistic Finality and Known Topology

Systems using lottery-based consensus have probabilistic finality. In controlled performance testing, the topology will be known. This will likely be the case in most permissioned deployments. In this case, a transaction is confirmed only when multiple nodes reach the same state.

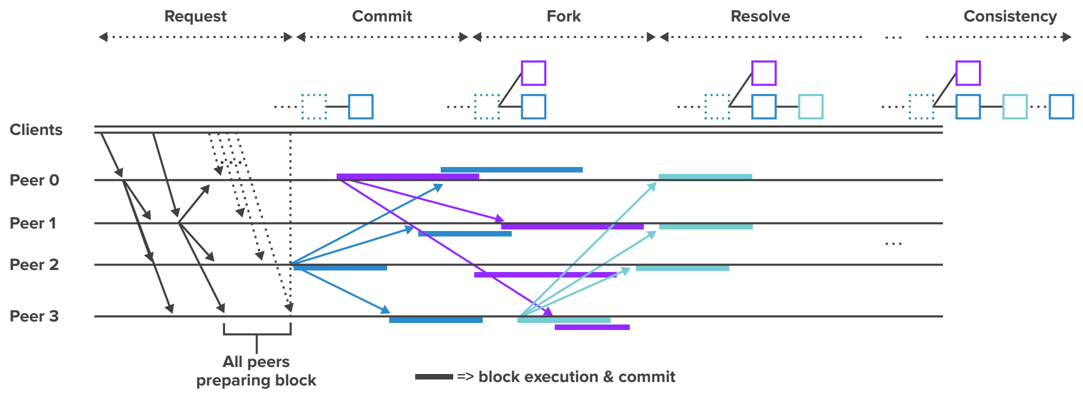

Figure 3: TRANSACTION FLOW FOR BLOCKCHAIN PLATFORMS USING RANDOM LEADER ELECTION CONSENSUS. SOURCE: https://github.com/tallharish/PerfMetricsPSWG/

Figure 3 depicts transaction flow for platforms using random leader election consensus like PoW or proof of elapsed time (PoET). This form of consensus does not have discrete rounds like PBFT. However, the phases depicted in the diagram represent conceptual operations:

- Request—A client submits a transaction to the network.
- Commit—One validator publishes the transaction in a block to the network.
- Fork—Zero or more validators inadvertently publish competing blocks.
- Resolve—Upon receipt of the winning block, other validators resolve the fork.
- Consistency—Subsequent reads to any validator produce consistent results.

Since the eventual consistency of the network is probabilistic, this gives rise to using a percentage of the network as a threshold for measuring latency.

Example: In a 20-node network, running PoET, 19 nodes commit a transaction within 25 seconds after the client submitted it. The final node publishes a competing block which it rolls back during fork resolution, causing an additional 10-second delay. Table 1 lists the commit times at all 20 nodes.

As shown in Table 2, this transaction has a 25-second delay at a 95% threshold of nodes and a 35-second delay at a 100% of the nodes.

A company may require a 100% threshold, in which case the latency should be reported as 35 seconds at 100% of nodes.

Node ID	Commit Time (seconds)	Node ID	Commit Time (seconds)	Node ID	Commit Time (seconds)
Node 0	10	Node 7	25	Node 14	19
Node 1	14	Node 8	4	Node 15	20
Node 2	9	Node 9	23	Node 16	15
Node 3	20	Node 10	17	Node 17	19
Node 4	22	Node 11	18	Node 18	8
Node 5	14	Node 12	12	Node 19	35
Node 6	10	Node 13	6		

TABLE 1: EXAMPLE TRANSACTION LATENCY DATA AT 20 NODES

Commit Time (seconds)	Threshold
20	80%
22	85%
23	90%
25	95%
35	100%

TABLE 2: EXAMPLE TRANSACTION LATENCIES AT VARIOUS NETWORK THRESHOLDS

Case 3: SUT with Probabilistic Finality and Unknown Topology

Such a system is usually designed to run in an untrusted environment with anonymous "miners." Only limited access points are revealed to the client. In this case, the confirmation should be delayed until enough blocks are chained behind the block/transaction.

For example, in Bitcoin, a delay of six blocks is usually considered sufficient to confirm the transaction. A performance test tool can estimate this by recording the block in which each transaction shows up. This data can be used to do a percentile analysis, so we can estimate the percentile of transactions committed up to a certain block, as shown in Figure 4 below. An organization can use this metric for risk assessment of their transactions.

Example: A transaction submitted when block x is the highest can be found in block (x + 2) with 50th percentile chance, in block (x+5) with 90th percentile chance, in block (x+7) with 99th percentile chance. Thus the transaction latency is 2 blocks at 50th percentile, 5 blocks at 90th percentile, and 7 blocks at 99th percentile respectively. In case blocks are generated with variable frequency, this can also be visualized by time.

Note that this measurement would vary with the average block size (in kB or average number of transactions per block), block frequency, and network size.

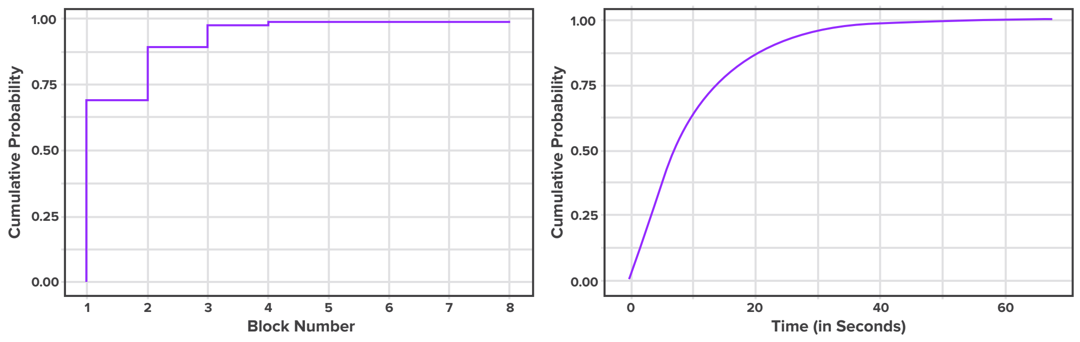

FIGURE 4: TRANSACTION CONFIRMATION PROBABILITY.
SOURCE: https://github.com/tallharish/PerfMetricsPSWG

Appendix B: Failed Transactions

There are several possible reasons why blockchain transactions can be rejected, including consensus errors, syntax errors, and version errors.[3]

Consensus errors

Validation logic (VSCC in the case of Hyperledger Fabric)

Policy failure (endorsement policy not satisfied in the case of Hyperledger Fabric)

Syntax errors

Invalid input (smart contract id, unmarshalling errors, and so on)

Unverifiable client or endorsement signature

Repeated transaction (due to error or replay attack)

Version errors

By version control (readset version mismatch, writeset is unwritable)

For the current version of this document, we consider **throughput** the same as **goodput**.

Since different blockchain platforms handle consensus and transaction validation differently, it is hard to align error classes across platforms. And assessing the reasons why any transaction is rejected requires a deeper analysis of sub-system metrics. While this could be interesting to the platform's developers, this is outside the scope of this document.

Appendix C: Experimental Metrics

This appendix defines any metrics which may offer interesting insights into blockchain performance, but have not yet reached a mature-enough definition to be included in the main body of this document.

The Performance and Scale Working Group would like feedback on whether and how these experimental metrics are being used in enterprise blockchains. We also hope this serves to seed some research interest in developing blockchain-specific metrics.

Blockchain Work

Blockchain Work = f(Transaction throughput, network size)

Blockchain Work is a function of the transaction throughput and the number of validating consensus nodes. This metric is intended to most completely express the amount of work accomplished by a blockchain network in a single figure.

Consider a blockchain with a single node: This may achieve high throughput, but no guarantee of availability or integrity. The blockchain work metric provides a meaningful characterization of blockchain performance, rather than simply database performance.

At publication time the Working Group had not identified an ideal function. One suggested function is below:

Product: Transaction Throughput * Log(Network Size)

References

1. Ittay Eyal, Adam Efe Gencer, Emin Gün Sirer, and R. Van Renesse. **Bitcoin-NG: A Scalable Blockchain Protocol** (**https://www.usenix.org/system/files/conference/nsdi16/nsdi16-paper-eyal.pdf**). Usenix Conference on Networked Systems Design and Implementation, ser. NSDI'16. Berkeley, CA, USA. USENIX Association. 2016. pp. 45–59.
2. Karama Kanoun and Lisa Spainhower. **Dependability Benchmarking for Computer Systems**. Wiley-IEEE Computer Society Press. 2008.
3. **Hyperledger Architecture, Volume 1: Consensus** (**https://www.hyperledger.org/wp-content/uploads/2017/08/Hyperledger_Arch_WG_Paper_1_Consensus.pdf**).

Further Reading

The following sources were referenced during the preparation of this document.

Miguel Castro and Barbara Liskov. **Practical Byzantine Fault Tolerance** (**http://pmg.csail.mit.edu/papers/osdi99.pdf**). Proceedings of the Third Symposium on Operating Systems Design and Implementation. New Orleans, USA. February, 1999.

Kyle Croman, Christian Decker, Ittay Eyal, Adam Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. **On Scaling Decentralized Blockchains. Financial Cryptography and Data Security** (**https://link.springer.com/chapter/10.1007/978-3-662-53357-4_8**). FC 2016. Lecture Notes in Computer Science, vol 9604. Springer, Berlin, Heidelberg. pp. 106–125.

Ingo Weber, Vincent Gramoli, Alex Ponomarev, Mark Staples, Ralph Holz, An Binh Tran, and Paul Rimba. **On Availability for Blockchain-Based Systems** (**https://ieeexplore.ieee.org/document/8069069/**). IEEE Symposium on Reliable Distributed Systems (SRDS). 2017. pp. 64–73.