Systematic Performance Evaluation Using Component-in-the-Loop Approach

Imre Kocsis*, Attila Klenik*, András Pataricza*, Miklós Telek†, Flórián Deé* and Dávid Cseh*
*Budapest University of Technology and Economics,
Department of Measurement and Information Systems, Hungary
Email: {ikocsis,klenik, pataric}@mit.bme.hu
†Budapest University of Technology and Economics,
Department of Networked Systems and Services, Hungary
Email: telek@hit.bme.hu

Abstract

Timeliness and throughput critical applications require a framework offering predictable temporal characteristics. The best practice for estimating a prediction of the system dynamics relies on benchmarking, i.e., measuring the reaction of the system under evaluation by applying a representative workload to it. Each novel middleware solution needs such an evaluation as part of the development process to assure an appropriate throughput in the future use.

General purpose Blockchain frameworks are viable replacements for many current systems in several sectors – such as finance, healthcare, and IoT – by providing a fully distributed, secure, and non-repudiable ledger as a service. Blockchain technologies target domains with a large number of interactions, thus demanding strict performance guarantees in the form of formal Service Level Agreements. Engineering for performance targets in a trustworthy manner requires performance models. However, performance characteristics of Blockchain systems are highly unexplored due to the novelty of the technology.

This paper proposes a general-purpose, systematic methodology for the performance analysis of complex systems, such as Blockchain frameworks. A component-in-the-loop approach aids the identification of throughput bottlenecks, sensitivity analysis, and configuration optimisation. The Linux Foundation-hosted Hyperledger Fabric – a pilot reference implementation of a Blockchain framework – serves as a case study for the presented methodology.

Keywords: performance evaluation, Blockchain, systematic analysis, hierarchical analysis, component-in-the-loop, exploratory data analysis, sensitivity analysis

I. Introduction

Motivated by the success of Bitcoin [1], the interest for Blockchain technologies proliferated in the past years. The irrefutable way of committing and auditing transactions without a third trusted party makes Blockchain a promising new technology for numerous sectors, e.g., finance,[1] business workflow [2], healthcare,[2] the government sector,[3] Internet of Things [3], and Cyber-Physical Systems.

Bitcoin assembled some previously known concepts ingeniously in 2008, giving rise to a new cryptocurrency. The core idea is that peers in an open peer-to-peer network concurrently (logically and cryptographically) validate and track system transactions – the transfer of ”coins” between pseudonymous asset holders. Every peer stores the ordered transactions in blocks; each block receives a hash upon closure included in the next block as a backward link, as well.

To keep ”most” peers honest (and thus defend the currency against attacks such as double-spending), the closure of blocks is a race between peers; the first peer that finds a suitable block-hash receives a newly ”minted” coin. This computationally intensive operation is called mining. From the aspect of distributed system theory, this part of the Bitcoin protocol is a consensus approach (also referred to as ”Proof of Work”). Note, that maintaining the same chain of blocks – that is, the Blockchain – at all peers is fundamentally a state machine replication problem [4]. In contrast to traditional currencies – and most classic financial ”settlement-style” arrangements – there is no single trusted third party entity (e.g., the government, the stock exchange, or interbank communication cooperatives), rather, trust is distributed among the participants. As long as the majority of peers are ”honest,” the system tolerates even Byzantine – including actively malicious – faults in other peers [5].

The use cases of distributed, ”trustless,” Bitcoin-inspired peer-to-peer systems are wider than managing cryptocurrencies. This inspiration gave rise to a large number of continuously evolving, experimental systems. These all modify the ”Bitcoin architecture” (and protocol) in fundamental ways, but two aspects do not vary: a) there is a Blockchain and peers maintain a copy of it and b) peers participate in a distributed consensus protocol to maintain a consistent state of the system.

The primary modifications for enterprise use cases are the following:

- The leading Blockchain technologies are now ”programmable.” These systems support ”smart contracts,” i.e., user-defined transaction logic that can be far beyond the complexity of passing units of cryptocurrency.
- A permissioned system applies explicit authentication and authorisation instead of allowing pseudonymous participants.

[1]http://www.reuters.com/article/banking-blockchain-bonds-idUSL8N16A30H [Online. Accessed: 2017-06-23]
[2]http://qz.com/628889/this-eastern-european-country-is-moving-its-health-records-to-the-blockchain/ [Online. Accessed: 2017-06-23]
[3]https://www.gov.uk/government/publications/distributed-ledger-technology-blackett-review [Online. Accessed: 2017-06-23]

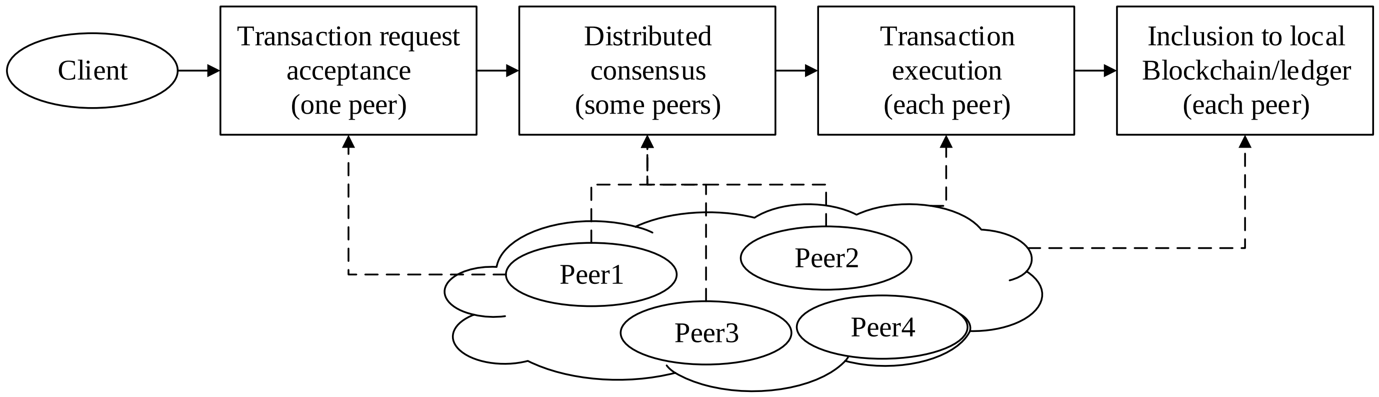
Client
Transaction request acceptance (one peer)
Distributed consensus (some peers)
Transaction execution (each peer)
Inclusion to local Blockchain/ledger (each peer)
Peer1
Peer2
Peer3
Peer4

Figure 1: General architecture of Blockchain systems.

- As the original Proof of Work-based approach used in Bitcoin puts a natural limit on throughput (in the order of ten transactions per second [6]), high-throughput applications consider other Byzantine consensus protocols.

Figure 1 illustrates the general architecture and transaction flow of Blockchain systems. Clients send transactions to peers in the network. The peer that receives a transaction distributes it on the network using, for example, some flooding method. The peers then globally order the received transactions using a distributed consensus protocol – Proof of Work in case of Bitcoin. Usually, they perform this ordering on a batch (also referred to as a block) of transactions for performance reasons. Once they establish a global order, every peer ”executes” the transactions in the agreed-upon order. Finally, every peer persists the transaction-induced changes in its local ledger, that is, they append the newly processed block to the end of the Blockchain. Note, that the global ordering and execution of transactions can interleave, or can be performed together – as is the case for Bitcoin.

Blockchain standardisation has begun,[4] and open source global collaborations started their work. Importantly, the Linux Foundation hosts the Hyperledger project.[5] The project is an umbrella for multiple distributed ledger technologies, among which the most mature is Hyperledger Fabric – previously IBM Open Blockchain. Many intended applications of Blockchain technologies require performance guarantees – irrespective of whether a Blockchain-based solution competes with legacy systems and approaches, or offers genuinely innovative functionality. However, to guarantee performance and to design deployments against performance targets, performance models are required that map workload, configuration and the characteristics of the operational environment into the ”engineered performance capacity” in a trustworthy manner.

The complexity of Blockchain systems makes it challenging to ensure certain requirements. Such requirements include temporal properties, like throughput and timeliness; dependability properties, like availability; and security aspects, like confidentiality and privacy. Employing model-driven development (MDD [7]) allows the application of model-based simulations and (formal) analyses to facilitate ensuring these requirements. A subset of these models captures performance-related behaviours enabling the performance prediction of complex systems [8]. Model-based performance analysis supports the identification of bottlenecks and execution of sensitivity analysis in the early stages of development, both on platform independent models (e.g., functional architecture) and platform dependent models (e.g., the configuration of system components).

Although mature and well-studied methodologies exist for performance analysis [9], applying them to complex systems is cumbersome. In this paper, our goal is to present a simple, manageable, systematic methodology for the performance analysis of such systems (including the Hyperledger Fabric Blockchain framework) that helps to reduce the complexity of the evaluation. The core concept of the methodology is the component-in-the-loop (CIL) approach that facilitates the isolation of components from the rest of the system (i.e. their environment), thus simplifying the evaluation and paving the road for compositional performance modelling [10].

The structure of the paper is the following. Section II outlines some of the major challenges that arise during performance analysis of Blockchain systems (and complex systems in general). In Section III, the authors propose a systematic methodology for overcoming these challenges, backed by the component-in-the-loop approach. Section IV presents the application of the methodology to the Hyperledger Fabric case study. Section V highlights further methods for aiding performance analysis and tuning.

II. Challenges

Model-based performance analysis of complex systems poses several challenges arising during the construction and evaluation of performance models. This section presents the main difficulties of the performance evaluation process along the example of an evolving Blockchain system.

[4]ISO/TC 307: Blockchain and electronic distributed ledger technologies: https://www.iso.org/committee/6266604.html [Online. Accessed: 2017-06-23]
[5]https://www.hyperledger.org [Online. Accessed: 2017-06-23]

A. Building the Model

General purpose Blockchain systems are still an evolving topic. Accordingly, the publicly available background literature is quite scarce. Missing or inadequate component specifications are typical for early-stage software. As a consequence, no reusable, off-the-shelf component models exist, requiring the manual construction of specific models at every level of the model hierarchy.

The construction of a faithful performance model requires identifying the critical parameters and aspects of the system. There are several target factors to consider:

Distributed systems tend to have a complex functional and structural architecture with sophisticated dynamics. The temporal overhead related to internode communication is comparable to that of computation. The performance model has to cover both these aspects in addition to the temporal properties of computation. The behaviour is further complicated by non-deterministic processes, like voting in a consensus network or unexpected failures.

Deployment choices can affect the system performance in unexpected ways. For example, deploying a system to a cloud may result in resource sharing problems and parasitic interference [11], both originating externally of the system, but still degrading its performance.

Distributed systems may have a changing architecture. Nodes can dynamically join or leave the network for various reasons, e.g., failure. Interdependent components may change their relations dynamically depending on use case or data dependency, for example, during runtime reconfiguration of components.

Parameters of the model are often unknown at the time of modelling and need to be approximated using preliminary measurements of system components (assuming the necessary components are already available at the time of modelling). Component-wise measurements can be facilitated using isolation-based methods, such as component-in-the-loop (CiL), as proposed in Section III-G.

B. Using the Model

The evaluation of the expected characteristics of a designated application needs the extension of the performance model of the system with its interactions with the environment. The creation and evaluation of such a closed-loop model targeting the prediction of the characteristics of the designated application usually start with a simplified open-loop model, which represents the environment by a workload. Prediction of the performance of the target application needs such a representative workload, that is similar to expected real-life scenarios.

Due to the nature of future, still underspecified applications (like the evolving Blockchain-based ones), there is a lack of publicly available, real-life workload scenarios generalising operational experience. Similarly, standard benchmarks representing typical application categories for Blockchain platforms are still unavailable at the moment.[6] A possibility is to use synthetic workloads [12] to evaluate the architecture of the framework, but guaranteeing their representativeness requires great care to derive meaningful results.

Even after the elaboration of a faithful performance model, the scalability and numerical properties of qualitative and quantitative analyses can significantly affect the tractability of the problem at hand. Practical applications suffer from the phenomenon called state space explosion, during which the size of the state space of a distributed, asynchronous system scales exponentially with the number of its components due to interleaving, often making the analysis intractable.

Another source of the difficulty is the temporal behaviour of the system as it evolves. Extreme differences in event frequencies can result in the numerical instability of the algorithms used to derive time-dependent performance metrics. Many techniques exist to alleviate these difficulties [13]–[15], but the problem is far from solved for large, practical models.

III. Proposed Methodology

Each component of a system may have very complex performance characteristics on its own – not to mention the potential feedbacks between the components. These characteristics make a direct analytic approach towards performance modelling of such systems (e.g. Blockchain frameworks) complicated and error-prone. Consequently, our approach (Figure 2) towards performance model structure discovery follows a somewhat sequential workflow. It systematically identifies the critical components of the system in a top-down manner. Every step has well-defined output artefacts, contributing to the manageability and traceability of the evaluation process. Note, that in the further discussion the notion of ”system” refers to the current component under evaluation, which will change as we drill down into the system between iterations of the methodology.

A. Operating Envelope

The operating envelope of a system defines the subset of its input space that keeps the system within desired operational conditions. The operating envelope lays the foundations for the following steps by constraining the system state in which we will evaluate the system. It helps to reduce the complexity of the analysis by focusing only on certain behavioural aspects of the system. It may also limit the set of components and requirements we need to take into account during the evaluation.

[6]http://www.coindesk.com/isitc-blockchain-standards-benchmarks/ [Online. Accessed: 2017-06-23]

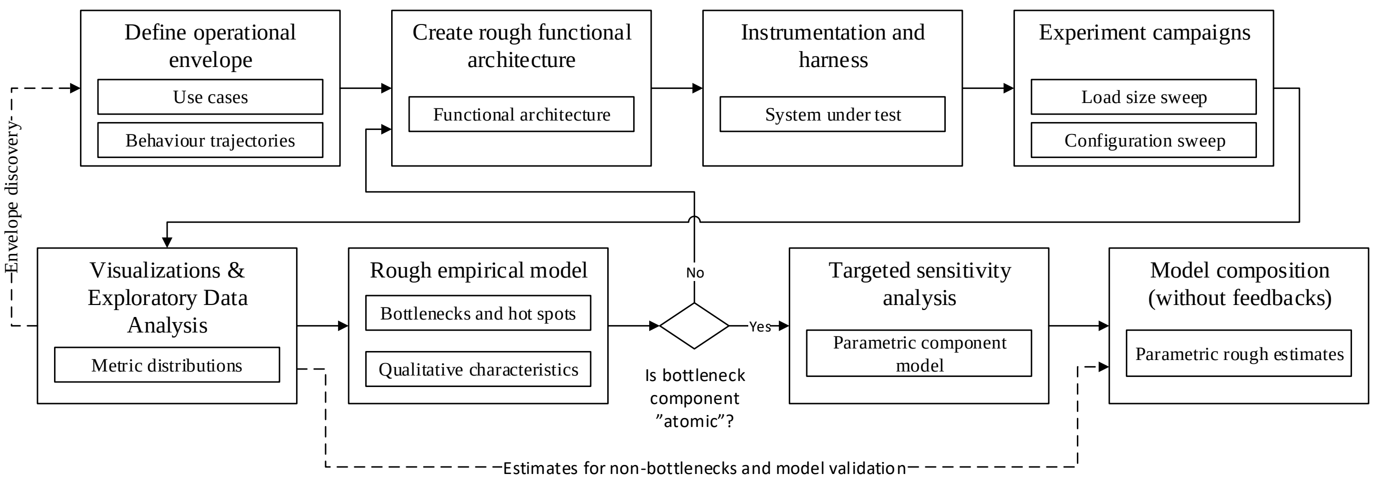
Define operational envelope
Use cases
Behaviour trajectories
Create rough functional architecture
Functional architecture
Instrumentation and harness
System under test
Experiment campaigns
Load size sweep
Configuration sweep
Envelope discovery
Visualizations & Exploratory Data Analysis
Metric distributions
Rough empirical model
Bottlenecks and hot spots
Qualitative characteristics
No
Yes
Is bottleneck component "atomic"?
Targeted sensitivity analysis
Parametric component model
Model composition (without feedbacks)
Parametric rough estimates
Estimates for non-bottlenecks and model validation

Figure 2: Performance evaluation methodology.

For example, if we wish to observe the error-free, stationary state of the system, then we must not overload it with requests during the experiments. Furthermore, we can omit certain requirements and components associated with, for example, overload handling or other error corrections since by definition, the system will not enter those states.

Defining an operating envelope usually consists of selecting some use cases associated with the system (e.g. browsing products, buying products) and limiting the behavioural trajectories that we deem allowed during the evaluation. The trajectories may be limited both qualitatively and quantitatively, e.g., request type and request rate, respectively.

B. *Rough Functional Architecture*

Once constraining the system behaviour according to an operating envelope, the next step is to identify a rough functional architecture model of the relevant parts of the system. This model will serve as a basis for further steps by defining the components considered for instrumentation, measurement, bottleneck analysis, etc.

At the first iteration of the methodology, this architecture model will describe the high-level components of the system and their mutual relations, but not details their internal operations. At further iterations, this architectural and functional model will be component-wise refined in a top-down way.

Refinement continues until a proper modelling resolution is reached giving a sufficiently accurate match between the model and the observations. It is important to note that the resulting model must conform to the use cases and behavioural trajectories defined in the previous step. Every component affected by the operating envelope must be present (at an appropriate abstraction level) in the architecture model. Ensuring this requirement can be simplified, for example, by using a (semi-)formal language for the first two phase of the methodology, like UML, and taking advantage of existing consistency checking methods [16], [17]. Finally, addressing multiple use cases is achieved by merging the different models corresponding to different operating envelopes into a single model.

C. *Instrumentation and Harness*

The model estimated and refined in the previous phase will identify the places of interest and correspondingly pinpoint the necessary instrumentation. Without loss of generality, let assume that the components of the system communicate through a predefined interface in an observable manner. The observation can occur through code instrumentation (e.g., adding performance counters) or other non-invasive methods, like processing operational logs or using internode communication monitoring.

Accordingly, the location of observations will be the interfaces of the components as specified by the architecture model. "Atomic" components prohibit further decomposition and internal instrumentation, thus form a hard limit of this gradual model refinement-based approach. A typical example for such "atomic" components are commercial-off-the-shelf (COTS) components without the option of architectural refinement and offering at most phenomenological performance modelling corresponding to the different externally controllable operation modes.

D. *Experiment Campaigns*

This step consists of defining experiment campaigns regarding the workload and configuration of the system. There are four, nearly orthogonal aspects to consider:

1) the type of the workload (queries, updates or insertions in the case of database loads),
2) the intensity of the workload (the rate of the incoming request),

3) the available resources (number and speed of CPU cores, network throughput, etc.),
4) the system configuration.

However, performing a full combinatorial evaluation of these factors can be overly costly and time-consuming. Note, that the type of the workload usually narrows the set of adequate system configurations. Accordingly, treating them independently from the intensity and resource aspects can significantly reduce the number of necessary experiments.

E. Visualization & Exploratory Data Analysis

The results of the experiment campaigns drive the next step, Exploratory Data Analysis (EDA). EDA offers a wide range of tools and methods [18], [19] enabling the better understanding of larger datasets, in this particular case of performance modelling, the observed behaviour of the evaluated system. EDA is especially useful in pure observation-based system identification without any a priori knowledge of the system. The purpose of a subsequent Confirmatory Data Analysis is the validation of the generated hypotheses and models.

EDA is a highly intuitive process performed typically by domain experts combining their experiences and the observations to build a consistent model. Proper visualisation techniques facilitate the direct involvement of domain experts without the need of deep mathematical background. Moreover, EDA can reveal discrepancies among the observations and expectations. Root cause analysis can be carried out in parallel with the model refinement process described above.

This step also serves as a validation for the experiments by identifying subsets of the campaigns that are outside the defined operating envelope. For example, EDA can reveal an overloaded state during the observation period, which lies outside the original operating envelope defining an error-free system in its stationary state. Removing these subsets of observations can result in more accurate models and may help the discovery of additional operating envelopes.

F. Rough Empirical Model

The next step involves building empirical models based on the EDA results. The deduced information from the observations – coupled with the use cases and trajectories defined by the operating envelope – allows for various further analyses, such as bottleneck and hotspot identification and qualitative characterisation of system behaviour.

Note, that these results (e.g., identified bottleneck) may be too high level to serve as a solid base for sensitivity analysis and formal performance modelling, especially in the early iterations of the methodology. In this case, performing the methodology again for every identified bottleneck will yield more detailed insights into the critical components of the system. The iterations (i.e. refinements) continue until we have a sufficiently ”low-level” understanding of the bottlenecks to begin the next step of the methodology.

G. The Proposed Component-in-the-Loop Approach

At the first iteration, the subject of the methodology is the entire system. However, the refinement step (that defines new inputs for the methodology) requires the re-evaluation of previous observations in more detail. One way to achieve this is to perform the same experiment campaigns on the entire system while gathering lower-level information about the critical components.

However, evaluating a component together with the entire system is cumbersome, e.g., deploying the system or applying configuration changes can be non-trivial and performing the experiments at the system level may lead to a significant time overhead. Furthermore, it may also introduce additional non-determinism into the measurements, compromising their repeatability. This limitation requires the isolation of the component under evaluation.

We propose to use a component-in-the-loop (CiL) approach to achieve the desired isolation. Accordingly, the component must be integrated into a simulated (artificial) environment, and a stimulus must be defined with respect to the workload generated by the original environment. This method is analogous to the well-known software-in-the-loop and hardware-in-the-loop approaches frequently used in the domain of embedded system verification.

When evaluating the entire system, its boundary is well-defined, just like its interaction with the environment. Accordingly, defining experiment campaigns is more of a theoretical challenge rather than a technical one. However, mapping these experiments to a component level poses several challenges.

First of all, significant distortions can happen to the workload by the time it reaches the component under evaluation. Determining the exact nature of the changes is, more often than not, a tedious process. It usually requires detailed knowledge of the inner functioning of the component, which could go beyond the scope of the current iteration of the evaluation. Second of all, even if the exact nature of the distortions is known, direct derivation of a component workload from a system workload is usually not feasible.

Our proposal is the integration of a record-replay capability at the boundary of the component to alleviate the difficulties posed by the component-in-the-loop approach. This way the system-level workload has to be repeated only once, to record it at the component level. This recording will serve as the basis for the component-level experiment campaigns, making the

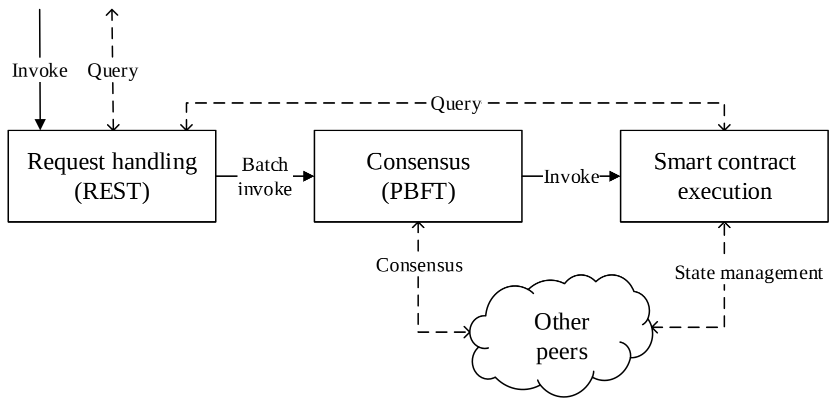

Figure 3: High level functional architecture of a Hyperledger Fabric peer.

decoupling of the rest of the system possible. This approach reduces the campaign definition to a technical problem, rather than remaining a theoretical one.

The drawback of the CiL and record-replay methods is their applicability to higher-level components of the system. The isolation of low-level components (e.g., databases) is usually a straightforward process because these components do not depend on further building blocks of the system. However, a higher-level service/component can depend on other parts of the system, making it difficult to isolate it entirely while preserving its performance characteristic. In this case, the methodology is subsystem-based, rather than purely component-based. The applicability of mocking methods used in traditional software unit testing [20] requires careful consideration and is out of the scope of this paper.

H. Targeted Sensitivity Analysis

Performance tuning aims at the elimination or reduction of bottlenecks identified during the previous process of benchmarking. There are two means to improve the performance of a particular component:

- Configuration sweeping of multi-mode devices searches the best mode of operation in multi-mode components.
  A typical example of this kind of setting is the selection of the cache policy in RocksDB. Depending on the ratio and frequency distribution of read and write operations, write-through-style so-called synchronisation or a copy-back styled buffering mode of cache management may deliver highly different dynamic characteristics.
- Parametric tuning of the setup by sensitivity analysis over a configuration space searches the best quantitative parametrisation. For instance, it can assess the impact of increasing buffer sizes in a RocksDB on the global throughput.

IV. Hyperledger Fabric Case Study

This section presents our case study about the performance evaluation of Hyperledger Fabric, version 0.6[7], systematically following the methodology proposed in Section III. Most of the technical details are omitted for the sake of simplicity and readability. The detailed working of Hyperledger Fabric and the formal performance modelling steps are out of the scope of this paper.

A. Evaluation Process

Operating Envelope: The operating envelope of interest for this evaluation is a fault-free, steady-state behaviour of the system. Accordingly, some complex operation of the system, e.g., state synchronisation among peers, can be omitted, simplifying the evaluation process. The case of an overloaded system is briefly discussed later in Section IV-C.

Functional Architecture: Figure 1 presents a general architecture for Blockchain-based systems. Figure 3 refines this by naming the concrete technologies used for the steps. The operation of the system consists of two main steps: Transaction batch ordering with a (pluggable) consensus protocol and sequential smart contract execution. This architecture will serve as the subject of the first iteration in the evaluation.

During consensus, the project employs the Practical Byzantine Fault Tolerance (PBFT) [21] protocol to order the incoming transactions while providing a certain degree of fault tolerance against Byzantine faults. PBFT also handles changes in the system state, e.g., peer failure or inconsistent (stale, corrupt) state of peers. After consensus, Hyperledger Fabric uses a software stack during the execution of smart contracts (referred to as chaincode in the project) to ensure modularity. The details of this software stack are omitted in the first version of the architecture.

[7]https://github.com/hyperledger/fabric/tree/v0.6 [Online. Accessed: 2017-06-23]

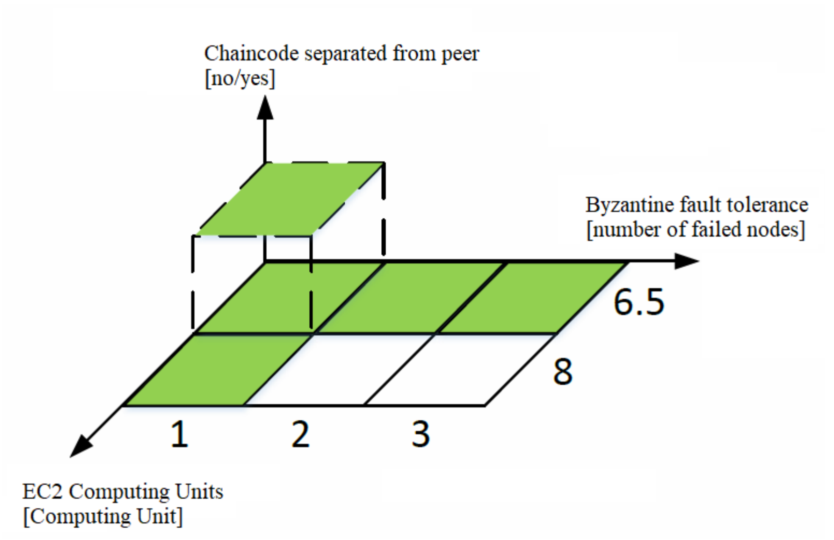

Figure 4: Parameter space of the configuration sweep.

Instrumentation and Harness: The identified component interfaces were instrumented at their boundaries to measure the service times of the respective operations. The harness solution builds on Amazon Web Services[8] and deployments are highly configurable. Notable features compared to the Vagrant-based development environment[9] include:

- The ability to use a dedicated cloud virtual machine (VM) instead of running the whole peer-to-peer system in a single VM (with peers deployed into containers).
- The ability to use an open-loop workload generator.
- The ability to be easily adapted to representative benchmarks, whenever they become available.

Experiment Campaigns: The performed experiments consisted of a workload size sweep (in the range of $32 - 304$ transactions/second) using various configurations of the system. Note, that at this level of detail only a few configuration points are available, e.g., the number of nodes in the system, the used computing power or other deployment choices.

Figure 4 highlights the performed experiments (marked by the filled rectangles) with respect to the different combinations of three configuration aspects. The vertical axis indicates whether the chaincode way deployed on the same virtual machine as the peer or not. The horizontal axis denotes the number of Byzantine node failures the deployed configuration can tolerate, which determines the size of the peer network according to the formula $n = 3 * f + 1$, where f denotes the number of faulty nodes the network should tolerate and n is the total number of nodes in the network. The diagonal axis indicates the integer processing power of the virtual machines measured in EC2 Computing Units.[10]

Visualization & EDA: Figure 5 shows the distribution of end-to-end latencies of transaction executions for various incoming task rates. End-to-end latency here means the time that elapses between issuing an invoke request to the system and that request appearing on the blockchain. The discrete scale of the horizontal axis corresponds to different measurements performed at a given load, measured in transaction invokes per second. The vertical axis denotes the end-to-end latencies in milliseconds, each latency value representing a per-second average of multiple transactions. The visualisation uses boxplots [22] to provide an easy-to-interpret descriptive statistic of the distributions of transaction latencies.

Table I summarises the observations for the primary operations that each peer performs, obtained for a request rate by around 300 transactions per second, which proved to be the highest rate that keeps the system in a stationary state.

Transaction admission means the receipt of requests to the public system interface, in the form of Representational State Transfer (REST [23]) operations. Handling of an incoming request through this interface is a fast, highly parallel operation which makes its overhead negligible compared to other activities.

The employed consensus protocol (PBFT [21]) performs global ordering on batches (blocks) of transactions for performance reasons. A batch is created either after a specific time (i.e. timeout) or after a given number of transactions, set to 1 second and 500 transactions during the evaluation, respectively. Batching makes the consensus highly adaptable to different network sizes and incoming request rates.

The transaction execution phase consists of the ordered, sequential execution of the corresponding smart contracts on each peer. Upon the successful completion of a transaction batch, the induced changes are persisted in the Blockchain synchronously by creating a new block.

[8]https://aws.amazon.com/ [Online. Accessed: 2017-06-23]
[9]http://hyperledger-fabric.readthedocs.io/en/v1.0.0-beta/dev-setup/devenv.html [Online. Accessed: 2017-06-23]
[10]Hardware Information section, Question 3 at: https://aws.amazon.com/ec2/faqs [Online. Accessed: 2017-06-23]

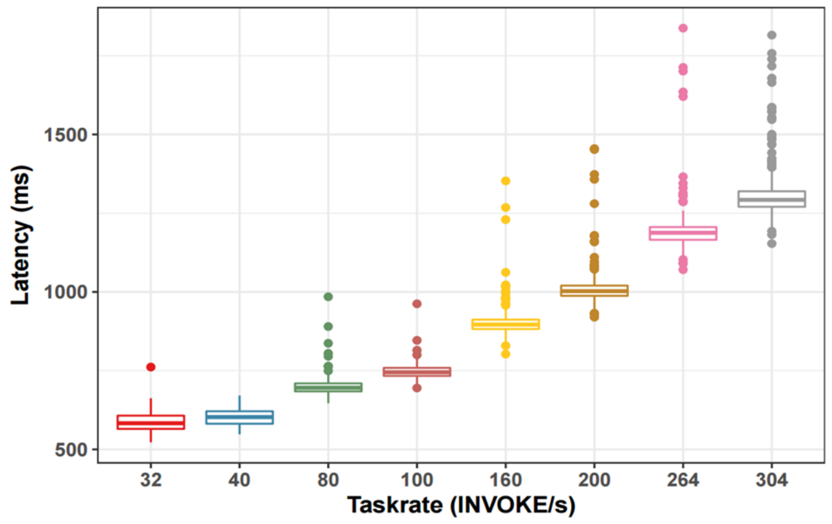

Figure 5: End-to-end latencies of transaction executions, per-second averages.

Operation	Service time	Notes
Transaction admission	1.5 ms/transaction	Highly parallel
Batch consensus	45 ms/batch	Highly adaptable
Transaction execution	540 – 750 ms/batch	Sequential execution
Block creation	3 – 42 ms/batch	Highly depends on workload

Table I: Typical request service times by peer activities.

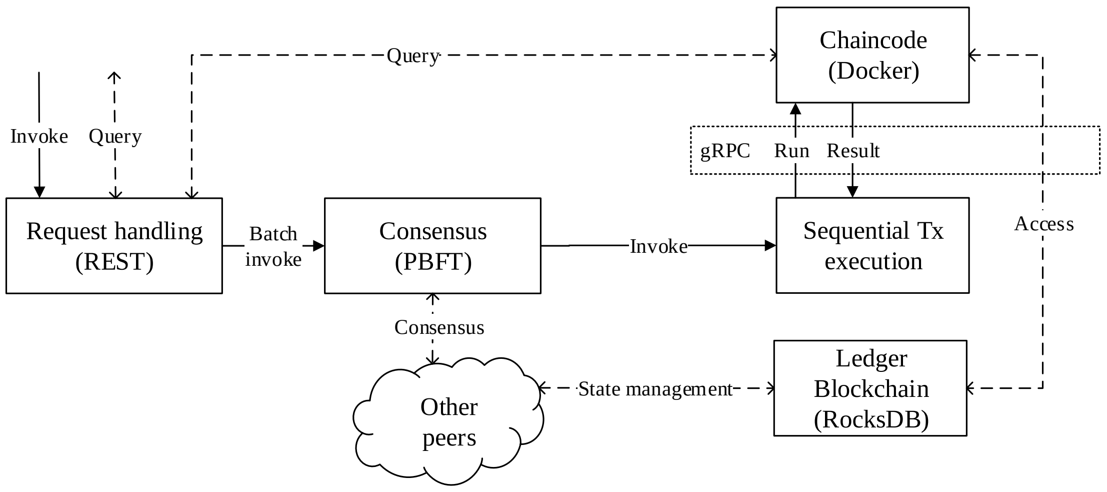

Figure 6: The refined architecture of Hyperledger Fabric.

Empirical Model: The previous observations and data analysis identified the transaction execution phase as the bottleneck in the system. The distributed consensus also can have a significant service time, although within the defined operating envelope it is scalable enough not to be the bottleneck. Section IV-C presents a case where the consensus acts as the component with the significant overhead.

This result selects the execution phase as the target of detailed sensitivity analysis. However, the current level of the functional architecture is too abstract to perform meaningful analysis to construct a component performance model. Therefore, the methodology is performed a second time, but this time its subject is the transaction execution phase.

Functional Architecture, second iteration: The second iteration of the methodology starts by refining the transaction execution phase. The new model will serve as the subject of further performance evaluation. Figure 6 presents the refined architecture.

Hyperledger Fabric uses a software stack during the execution of smart contracts (referred to as chaincode in the project) to ensure modularity. A remote procedure call (RPC, [24]) library called gRPC[11] accomplishes network communication between

[11]https://grpc.io/ [Online. Accessed: 2017-06-23]

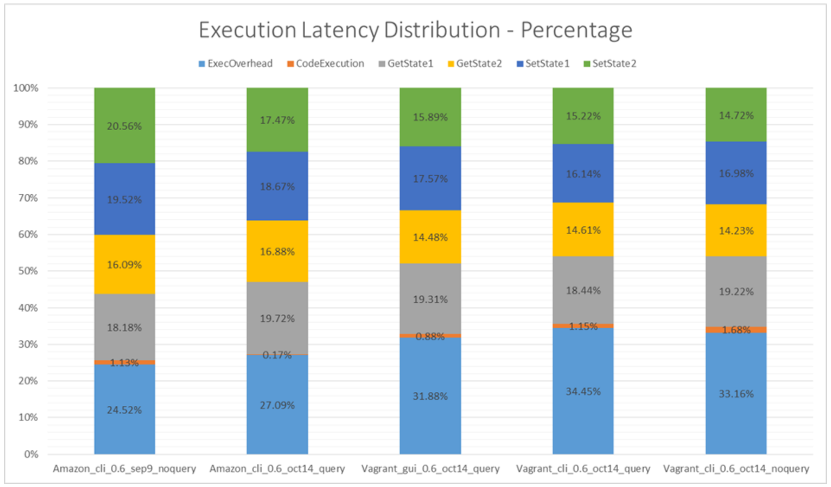

Figure 7: Execution latency distributions for different deployment configurations.

the chaincode and the peer logic layer, where the former is isolated (virtualised) using a Docker container [25]. At the ”bottom” of the software stack, RocksDB[12] provides persistent storage for system state (e.g., the blockchain) and related metadata.

To further facilitate the upcoming experiment campaigns, it is good practice to decouple the component under evaluation from the rest of the system, as described in Section III-G. Fortunately, Hyperledger Fabric allows the use of a ”noop” consensus and a single peer, meaning that the peer receives the transaction batch without performing any distributed consensus beforehand, thus eliminating a complex service from the workflow and any measurement noise it could cause.

Instrumentation and Harness, second iteration: For the second iteration, the executed chaincode and the communication layer of the peer were instrumented to derive more detailed measurements. The observations were persisted using the built-in logging capabilities of the framework, necessitating only a few invasive instrumentation points.

Experiment Campaigns, second iteration: The performed experiment campaigns were the same as in Section IV-A, but using only a single peer in the network.

Visualization & EDA, second iteration: Figure 7 visualises the percentage of time spent during the execution of deployed chaincodes. The observed distribution is consistent across deployments. Note, that the time of the actual code (business logic) execution is negligible compared to the database accesses performed.

Empirical Model, second iteration: The component that executes the chaincodes uses a software stack that performs data transformation, remote communication and storage operations in a blocking, synchronous manner. These steps are each suitable candidates for further performance analysis. However, general experience suggests that it is the storage layer that significantly affects performance in similar setups (e.g., in multi-tier architectures).

For this reason, RocksDB was selected as the target of further evaluations. Moreover, the data transformation and communication parts of the system do not offer much tuning possibilities, even though they create more overhead than the database. Since RocksDB is a third party library, it can be considered an ”atomic” component in the system and the methodology can proceed to its targeted sensitivity analysis.

Targeted Sensitivity Analysis: RocksDB is an embedded (i.e. hosted by the user process), highly configurable, persistent key-value store. Its flexibility allows it to be used in various environments while retaining its performance. Moreover, it comes with a similarly configurable benchmark tool that facilitates the comparison of different configurations. RocksDB is integrated into Hyperledger Fabric through its C application programming interface (API), which defines a clear boundary for the component, facilitating its isolation from the system.

Numerous standard benchmarks are available for traditional online transaction processing (OLTP) systems [26], [27], but not for key-value stores, as key-value stores gained popularity only recently and they lack a common standardised structure, i.e., they are highly specialised for certain workloads.

When a chaincode issues a read operation towards the database, that call will go through many software layers before it reaches the RocksDB. This ”call distance” in the software stack further complicates the definition of a representative workload. Security-related data transformations are employed, and additional metadata is calculated between the two layers, both distorting the workload of the database compared to the ”pure” chaincode-generated workload. Accordingly, the definition of a proper database workload requires the incorporation of the load generated by the middle layer additionally to the chaincode instructions.

To overcome these problems, RocksDB was instrumented with API level request recording. The instrumentation is capable of recording most of the RocksDB API requests used by upper layers in the system. The recording serves multiple purposes:

[12]http://rocksdb.org/ [Online. Accessed: 2017-06-23]

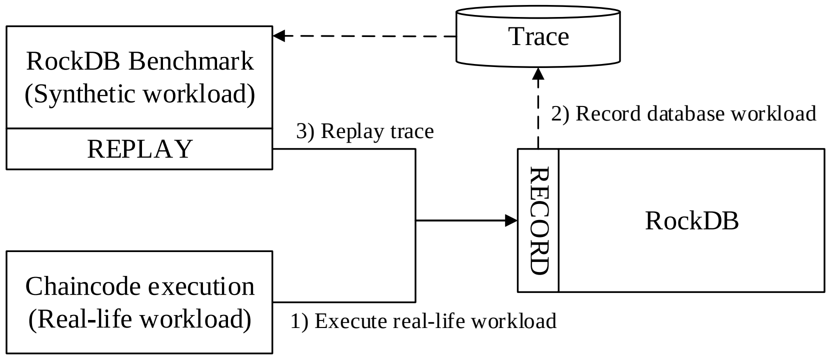
Trace
RockDB Benchmark (Synthetic workload)
REPLAY
3) Replay trace
2) Record database workload
RECORD
RockDB
Chaincode execution (Real-life workload)
1) Execute real-life workload

Figure 8: The record-replay setup of RocksDB.

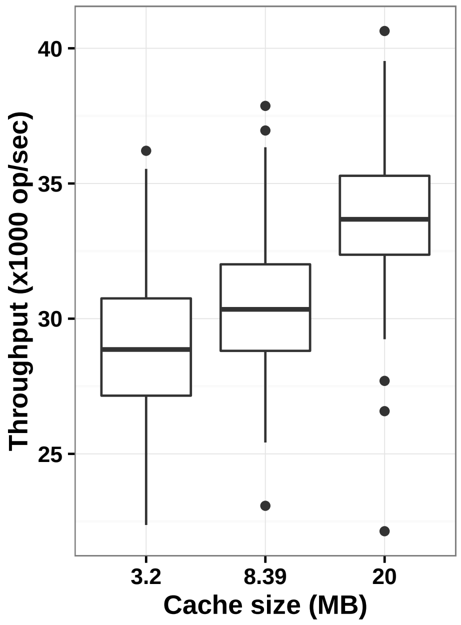

Figure 9: Boxplots of RocksDB throughput observations by cache sizes.

- It provides greater insight into the workload distortions happening in other middleware layers, without explicitly inspecting other components.
- It completely separates the origin of the workload from the database, whether it is a benchmark-generated or a real-life workload. Once the database workload has been recorded, its source can be left out from the consecutive evaluations.
- The recorded workload can be modified to derive additional, similar scenarios, differing, for example, in arrival rate.

Beside request tracing, the benchmark tool of RocksDB was also extended with the functionality of replaying previously recorded request. The advantage of this integration is the reusability of other benchmark functionalities, e.g., gathering statistics about operation executions and configuring workload independent aspects of the database, leveraging the configuration capabilities of the benchmark tool. Figure 8 shows the envisioned workflow for the record-replay methodology.

The evaluation was performed multiple times in different operational modes of the component. This configuration sweep is useful for two things: identifying critical parameters and determining a better configuration. As the applied configuration changes, we can perform sensitivity analysis on the database parameters to see how their values affect the overall performance of the component. This information will help guide the performance modelling later. Moreover, by comparing the performance results of different configurations, the best one can be selected for the current workload (and probably similar ones).

Due to the lack of real-life, representative workloads, during the evaluation of RocksDB, a built-in benchmark workload was used that generated a given number of random updates on the database records. The benchmark was configured and run by an external tool that performed a sweep over a predefined parameter space and collected the performance results. We compared the different results using exploratory data analysis (EDA), focusing on the throughput and response time of the component, based on certain parameters.

Figure 9 presents a comparison of throughputs in the case of different cache sizes. A boxplot encapsulates every measurement

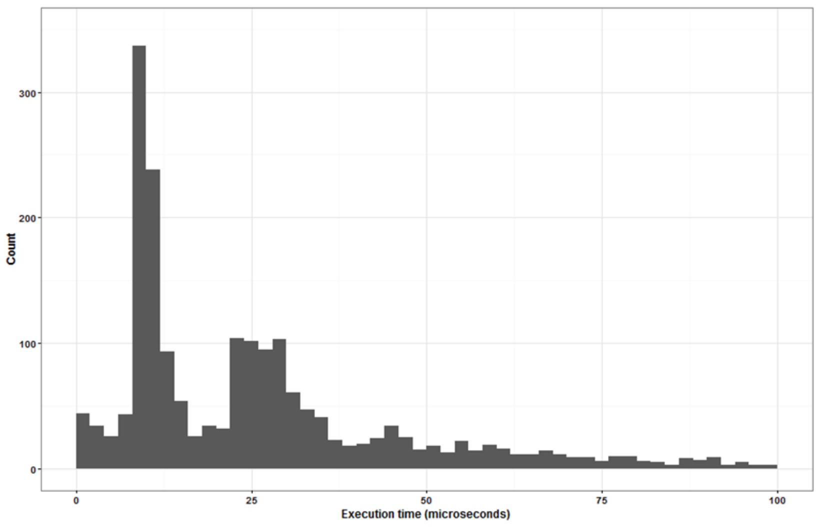

Figure 10: The latency distribution of database requests.

where the cache size is the same (e.g., 20MB). Inspecting the distribution of achieved throughputs of different measurements with the same cache size, we can formulate the following hypothesis: despite the random access mode of the benchmark, the throughput increases with greater cache sizes, regardless of other parameter settings. Observations acquired through this manner with EDA can help in guiding further, more focused sensitivity analyses on important parameters.

Figure 10 shows the observed latency distribution of database accesses. Note, that latencies also occurred in the order of 800 microseconds, but these observations were omitted for the sake of clarity of the diagram. This long-tailed distribution makes latency predictions less precise.

B. Summary of Initial Findings

We have applied the above approach to the 0.6 branch of Hyperledger Fabric. In the current absence of open, representative performance requirement sets and workloads, we have started with requirements that in our experience are ”reasonable” for performance-critical systems; a chaincode that simply moves assets between accounts; and constant rate load generation.

A natural and usual requirement towards performance-critical systems is that engineering their deployment and configuration are the main vehicles of influencing their performance capacity. This is not the case for Hyperledger Fabric 0.6; there is a cap around 300 transactions/second, even for completely fault-free scenarios. We have found that the issue is the strictly sequential chaincode execution of the ordered requests. This is not an absolute necessity - in theory, e.g., consensus could become problematic before sequential execution reaches its limit; however, the Docker-based chaincode execution also uses an RPC mechanism that in its default configuration is very slow (see Table I).

Additionally, chaincode execution time should be predictable; not only because certain scenarios may be latency-sensitive, but also because chaincode executions are subject to time off on each peer. Slow queries may fail to execute a transaction on one or more peers, as the ”database backend”, the RocksDB-backed ledger (and blockchain) is the critical component. With targeted sensitivity analysis, we have found that a) the configuration settings heavily influence RocksDB latency, b) latencies have a long-tailed distribution what due to the strictly sequential execution can have a serious global performance impact.

C. Behaviour in Case of Overload

A usual requirement towards performance-critical systems is that they manage overload situations in a predictable and well-defined way and with maintaining service at least partially. The simplest of the applicable patterns is actively rejecting new requests that are over the capacity of the system. This is not the case for Hyperledger Fabric 0.6. The following can be observed:

- For served requests, the write-to-ledger delay runs off (Figure 11).
- A heavily increasing number of requests get silently dropped, despite their active acceptance for execution.
- One peer gets ”stuck” in a state resynchronisation loop, thus decreasing the remaining fault tolerance to zero, or two peers get ”stuck” that disables consensus and thus, the entire system.

V. Conclusion

We have proposed a methodology for the practical performance characterisation of emerging Blockchain technologies and complex systems in general. The core idea is hierarchically determining and characterising the components with the most

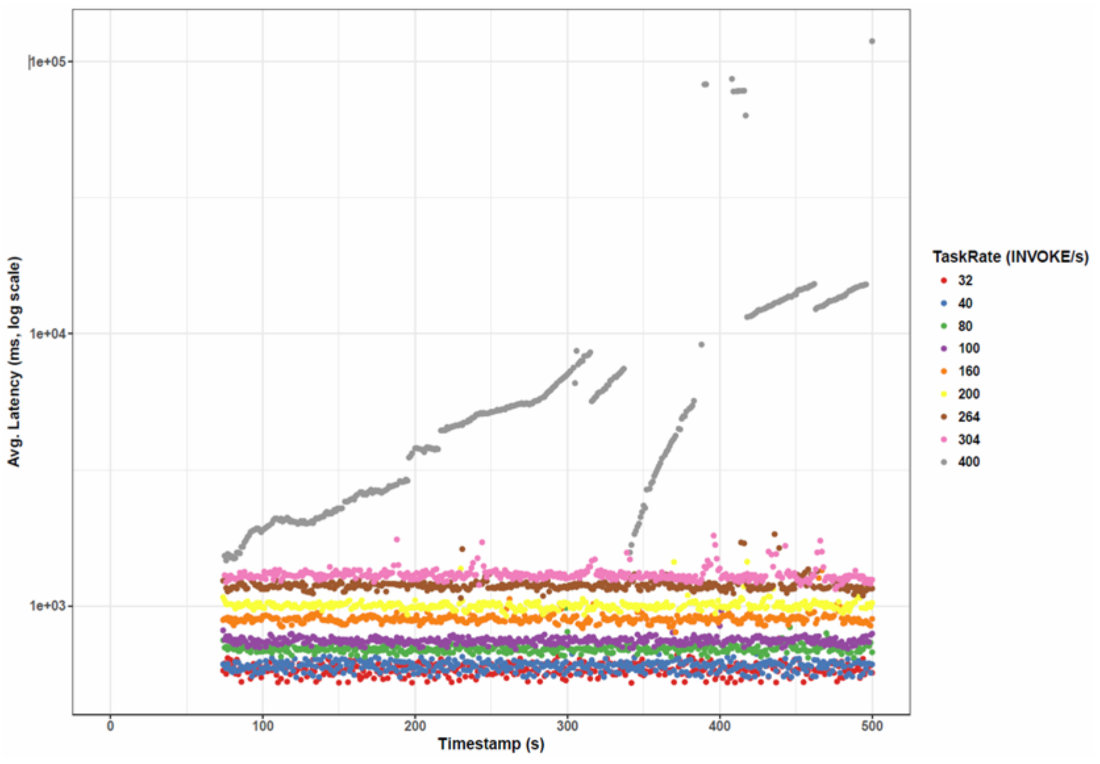

Figure 11: The delay from task acceptance to ledger modification.

influence on system performance in and around the assumed operating envelope of the system. Due to the experiment-driven nature of the hierarchical process and the fact that experiments are performed in harnesses mimicking the true systemic context, we dubbed the approach component-in-the-loop. We have applied the methodology to Hyperledger Fabric 0.6 to demonstrate its viability. We plan to repeat the process on the recently released 1.0 version of Hyperledger Fabric and compare the performance characteristics of its reworked architecture with the old one.

REFERENCES

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, “Untrusted Business Process Monitoring and Execution Using Blockchain,” in *Business Process Management: 14th International Conference, BPM 2016, Rio de Janeiro, Brazil, September 18-22, 2016. Proceedings*, Marcello La Rosa, Peter Loos, and Oscar Pastor, Eds. Cham: Springer International Publishing, 2016, pp. 329–347.

[3] K. Christidis and M. Devetsikiotis, “Blockchains and Smart Contracts for the Internet of Things,” *IEEE Access*, vol. 4, no. 99, pp. 2292–2303, 2016.

[4] F. B. Schneider, “Implementing fault-tolerant services using the state machine approach: A tutorial,” vol. 22, no. 4, pp. 299–319. [Online]. Available: http://doi.acm.org/10.1145/98163.98167

[5] I. Eyal and E. G. Sirer, *Majority Is Not Enough: Bitcoin Mining Is Vulnerable*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 436–454.

[6] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, “On Scaling Decentralized Blockchains,” in *Financial Cryptography and Data Security*. Springer, Berlin, Heidelberg, Feb. 2016, pp. 106–125.

[7] B. Selic, “The pragmatics of model-driven development,” vol. 20, no. 5, pp. 19–25.

[8] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni, “Model-based performance prediction in software development: a survey,” vol. 30, no. 5, pp. 295–310.

[9] G. A. Hoffmann, K. S. Trivedi, and M. Malek, “A best practice guide to resource forecasting for computing systems,” vol. 56, no. 4, pp. 615–628.

[10] X. Wu and M. Woodside, “Performance modeling from software components,” in *Proceedings of the 4th International Workshop on Software and Performance*, ser. WOSP ’04. ACM, pp. 290–301. [Online]. Available: http://doi.acm.org/10.1145/974044.974089

[11] X. Pu, L. Liu, Y. Mei, S. Sivathanu, Y. Koh, and C. Pu, “Understanding performance interference of i/o workload in virtualized cloud environments,” in *2010 IEEE 3rd International Conference on Cloud Computing*, pp. 51–58.

[12] A. Bahga and V. K. Madisetti, “Synthetic workload generation for cloud computing applications,” vol. 04, no. 7, p. 396. [Online]. Available: http://www.scirp.org/journal/PaperInformation.aspx?PaperID=5842&#abstract

[13] G. Ciardo, G. Lttgen, and R. Siminiceanu, “Saturation: An efficient iteration strategy for symbolic statespace generation,” in *Tools and Algorithms for the Construction and Analysis of Systems*, ser. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 328–342. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-45319-9_23

[14] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper, “Model-checking large structured markov chains,” vol. 56, no. 1, pp. 69–97. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S156783260200067X

[15] E. d. S. e. Silva and H. R. Gail, “Transient solutions for markov chains,” in *Computational Probability*, ser. International Series in Operations Research & Management Science, W. K. Grassmann, Ed. Springer US, no. 24, pp. 43–79. [Online]. Available: http://link.springer.com/chapter/10.1007/978-1-4757-4828-4_3

[16] B. Litvak, S. Tyszberowicz, and A. Yehudai, “Behavioral consistency validation of UML diagrams,” in *First International Conference onSoftware Engineering and Formal Methods, 2003.Proceedings.*, pp. 118–125.

[17] M. Usman, A. Nadeem, T. h. Kim, and E. s. Cho, “A survey of consistency checking techniques for UML models,” in *2008 Advanced Software Engineering and Its Applications*, pp. 57–62.

[18] S. Morgenthaler, “Exploratory data analysis,” *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 1, no. 1, pp. 33–44, 2009. [Online]. Available: http://dx.doi.org/10.1002/wics.2

[19] A. Pataricza, I. Kocsis, Á. Salánki, and L. Gönczy, *Empirical Assessment of Resilience*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–16. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-40894-6_1

[20] M. R. Marri, T. Xie, N. Tillmann, J. De Halleux, and W. Schulte, “An empirical study of testing file-system-dependent software with mock objects,” in *Automation of Software Test, 2009. AST’09. ICSE Workshop on*. IEEE, 2009, pp. 149–153.

[21] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI ’99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186.

[22] D. F. Williamson, R. A. Parker, and J. S. Kendrick, “The box plot: a simple visual method to interpret data,” *Annals of internal medicine*, vol. 110, no. 11, pp. 916–921, 1989.

[23] R. T. Fielding and R. N. Taylor, “Principled design of the modern web architecture,” *ACM Trans. Internet Technol.*, vol. 2, no. 2, pp. 115–150, May 2002. [Online]. Available: http://doi.acm.org/10.1145/514183.514185

[24] A. D. Birrell and B. J. Nelson, “Implementing remote procedure calls,” *ACM Trans. Comput. Syst.*, vol. 2, no. 1, pp. 39–59, Feb. 1984. [Online]. Available: http://doi.acm.org/10.1145/2080.357392

[25] D. Merkel, “Docker: Lightweight linux containers for consistent development and deployment,” *Linux J.*, vol. 2014, no. 239, Mar. 2014.

[26] M. Vieira and H. Madeira, “A dependability benchmark for OLTP application environments,” in *Proceedings of the 29th International Conference on Very Large Data Bases - Volume 29*, ser. VLDB ’03. VLDB Endowment, 2003, pp. 742–753.

[27] T. Hogan, *Overview of TPC benchmark E: The next generation of OLTP benchmarks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 84–98.