Using architectural modelling and simulation to predict latency of blockchain-based systems

Rajitha Yasaweerasinghelage Mark Staples Ingo Weber

University of New South Wales, Australia
Data61, CSIRO, Australia
<firstname>.<lastname>@data61.csiro.au

Technical Report
UNSW-CSE-TR-201704
February 2017

UNSW
SYDNEY

School of Computer Science and Engineering
The University of New South Wales
Sydney 2052, Australia

Abstract

Blockchain is an emerging technology for sharing transactional data and computation without using a central trusted third party. It is an architectural choice to use a blockchain instead of traditional databases or protocols, and this creates trade-offs between non-functional requirements such as performance, cost, and security. However, little is known about predicting the behaviour of blockchain-based systems. This paper shows the feasibility of using architectural performance modelling and simulation tools to predict the latency of blockchain-based systems. We use established tools and techniques, but explore new blockchain-specific issues such as the configuration of the number of confirmation blocks and inter-block times. We report on a lab-based experimental study using an incident management system, showing predictions of median system level response time with a relative error mostly under 10%. We discuss how the approach can be used to support architectural decision-making, during the design of blockchain-based systems.

1 Introduction

Blockchain is an emerging technology which provides a shared distributed ledger of transactions, allowing untrusting participants to interact without relying on a central trusted third party. It is the underlying technology of the Bitcoin [17] system and digital currency, but has since been used in a number of other public and private blockchain systems [7] with different features and characteristics. A number of studies have been conducted to explore the use of blockchain concepts in areas such as financial markets [21], supply chains [25], and consumer and business-to-business services [29]. In general, a blockchain can be used as a database, or as a software connector [29], and with the support of *smart contracts* [19], it can be used as a programmable environment for executing business logic or conditional transactions on the blockchain network [25]. Blockchain is an architectural alternative to conventional technologies, and when choosing between these alternatives there are trade-offs between non-functional requirements such as data integrity, transparency, cost and performance. Some literature explores blockchain security [9, 12, 23] and throughput [11] but, there is little work on other non-functional requirements such as latency and cost. For many applications, these could be significant when considering whether or how to use a blockchain.

Consider latency, which can be a drawback for blockchains. In a blockchain using Nakamoto consensus (longest chain wins) [17], to confirm a transaction, it needs to be included in a block, which should be endorsed by dependent blocks, known as *confirmation blocks*. On Bitcoin [17], the inter-block time is about 10 minutes and 6 confirmation blocks are often used [23], while on the public Ethereum blockchain [7] the average block generation time is about 15s (as on 15 November 2016) [1], and 12[1] confirmation blocks are typically recommended. Clearly, the latency for initial inclusion of a transaction is already higher than for traditional systems, and a large number of confirmation blocks will multiply this delay. Transaction delays can also arise from network delays, the transaction fee offered, the number of transactions being processed, and the strategic decisions made by miners. So, transaction inclusion times can vary widely.

Although longer than in conventional systems, these transaction latencies may be acceptable for some use cases, if the other potential benefits of blockchain can be achieved, such as decentralised trust. Nonetheless, it will still be important to be able to accurately predict system-level latency during the design phase, to assess the impact of this limitation on system requirements. Additionally, when using a blockchain there are many subsidiary design decisions, such as choosing between public or private blockchain, deciding on the number of confirmation blocks, and determining the integration with off-chain communication and enterprise systems. An inability to predict overall performance may itself be a barrier to the adoption of blockchain technology.

In this paper, we propose a model-driven approach for predicting latency in blockchain-based systems. The aim is that the approach supports early lifecycle stages and helps compare design alternatives. Our contributions are:

- An approach for modelling blockchain-based systems using performance modelling and simulation tools

[1] http://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203#203

- An evaluation of the accuracy of system-level latency predictions
- A comparison, using the tools, of the impact of the number of confirmation blocks on latency
- A comparison, using the tools, of the impact of inter-block time on latency
- A comparison of candidate design alternatives for the use case's business process model

This technical report is a long version of paper [30] including additional details, background and details. This paper is structured as follows. In Section 2, we outline previous work and related technologies. Section 3 discusses our approach for modelling and benchmarking transaction latency in blockchains. In Section 5, we describe our approach to using the Palladio Component Model [3] for modelling and simulation of the latency of blockchain-based systems, using an incident management system as an illustrative example. The benchmark data from Section 3 is used to configure these models. The accuracy of the system-level predictions from this model is evaluated in Section 4 by comparing the simulation predictions with measurements of a laboratory-based exemplar system. We discuss some blockchain-specific modelling issues and future work in Section 6, before concluding.

2 Background

This section briefly reviews background material related to this paper. We describe blockchain technology, the use of blockchains for business process management, and performance modelling and simulation.

2.1 Blockchain

Blockchain is a technology introduced by Bitcoin [17] to support holding and transferring of the Bitcoin digital currency. A blockchain is a distributed shared ledger system which validates and stores the history of transactions. The blockchain data structure uses *blocks* which are collections of transactions and other block-related metadata. These blocks are *chained* into a linked list, where a pointer from within a block is the cryptographic hash of the value of the previous block. This creates a tamper-resistant historical record. There can be many (thousands) active participants (often called 'miners') which jointly operate a blockchain system, but the proper operation of any single authority is not required for the overall system to establish transactional integrity, non-repudiation, and consensus between participants about the contents of the ledger. Large numbers of participants can act maliciously within the system, without stopping its good overall operation.

When a properly formed transaction is received by a participant, it will be validated and passed to their peers and so on, until it eventually reaches every participant. Participants may include a number of transactions into a block to append to the blockchain. Because there are many participants, there may be multiple competing candidate "next blocks". The Bitcoin blockchain (and Ethereum) resolves this using *Nakamoto consensus* [17]: the longest chain

of blocks seen by a participant is by convention taken by them to be the authoritative chain. In Bitcoin (and Ethereum), a valid block must also include a solution to a difficult cryptographic puzzle. This *proof-of-work* mechanism results in random leader election amongst participants for the creation of the next block, because the time required to solve the cryptographic puzzle varies probabilistically and in proportion to the computing power used by the participant. Proof-of-work also demonstrates a kind of economic investment in the operation of the blockchain system which at least partially aligns the incentives of participants with the integrity of the system. A participant who successfully creates a block can claim transaction fees and other mining rewards allowed by the system, as a return on their investment in the operation of the system. Other consensus mechanisms are possible. Proof-of-stake is similar to proof-of-work in using Nakamoto consensus, but instead of using a computationally (and electrically) expensive cryptographic puzzle, leader election and incentive alignment are achieved by committing stakeholdings of the blockchain digital currency. In private blockchains, traditional distributed consensus algorithms can be used instead of Nakamoto consensus. Conventional algorithms provide stronger transactional integrity properties and can be faster, but usually, are limited to a smaller and potentially less-hostile group of participants.

The focus of the Bitcoin blockchain is digital currency transactions, but as a general-purpose technology, blockchains can validate and store transactions of any type. *User-defined programs* can also be carried in a transaction and executed during validation. These programs, often called *smart contracts*, run on the blockchain network infrastructure in a pay-per-executed-instruction mode. Bitcoin allows only very simple straight-line smart contracts, but Ethereum [7] provides a Turing-complete language called *Solidity*. Smart contracts in Ethereum are first class elements which have access to blockchain states, and the values being transferred. According to Omohundro et al. [19] Ethereum is intended to create a basis for decentralised applications with business logic defined in smart contracts.

2.2 Blockchain for Business Processes

In prior work, Xu et al. [29] explored a way of using blockchain as a connector in architectures for blockchain-based systems. Weber et al. [25] further demonstrated the architectural range of blockchains, by showing how they could be used as a neutral ground for the model-driven execution of business processes. This approach allows the integration of organisations without the need of a trusted central coordinating authority. Weber et al. [25] use of blockchain to facilitate a collaborative process in two ways: as a 1) choreography monitor which stores the process execution status and provides immutable data storage; or 2) active mediator by coordinating the collaborative process.

As the smart contracts in a blockchain can not interact directly with the external world, a *trigger* component connects the processes executing on blockchain to enterprise systems by acting as an agent for an organisation. The trigger manages keys, keeps track of the data payload in API calls, and can interact with external services including other databases or web services. In addition to executing and validating the core business process logic on the blockchain, some of the framework logic is also managed and executed on the blockchain using preconfigured smart contracts. For example, instances of a process monitor smart

contract on the blockchain are created by a transaction invoking a *factory smart contract*, which contains a blueprint of the business logic. The blockchain-based system we use below in our experimental evaluation is a business process system using the approach from [25], which also measured transaction inclusion time and utilized the incident management exemplar use case we use here.

2.3 Architectural Performance Modelling

Architectural models can be used to predict the non-functional properties including latency, throughput, resource usage, and cost. These models can be used by analytical solvers or simulation engines to predict non-functional performance of a system at various stages of the development life cycle [5].

There are two types of performance models:

- Analytical performance models: capture performance aspects of the system and serve as input for the analytical solvers. Petri nets (PN) [16], Queueing Networks (QN) [4], Layered Queueing networks (LQN) [10] are examples for common analytical models.
- Architecture-level performance models: capture key factors influencing the performance of a system. Examples are the Palladio Component Model (PCM) [3], UML profile for Schedulability, Performance and Time [28], and Descartes Modelling Language [13]. Architectural models can be either simulated or automatically converted to analytical models. Generally, simulations take a longer time than the solvers to execute but may be more flexible.

In this paper, we have used the Palladio workbench [3] for architecture modelling of the latency of blockchain-based systems. Palladio was selected because it is freely available, supports the simulation of architecture models, has a 'UML-like' interface for model construction, and has proven flexibility for extensions such as architectural optimisation [14, 8] and new qualities [26]. The modelling concepts are well-aligned with component-based development and support the re-use of constructed models and components. Other advantages include the rich add-on development environment [2], ongoing support, and the large body of previous work. PCM models can be used directly for simulations with engines such as SimuBench or can be converted into an analytical model and solved using tools such as Line Solver [20].

3 Performance Model Construction

This section first describes our approach to benchmarking transaction inclusion times for blockchain. We then describe our approach for system-level performance modelling, targeting systems using Weber et al.'s [25] method for business process execution on the blockchain. These performance models are configured using the benchmarking results.

3.1 Benchmarking Transaction Inclusion on Blockchain

A key parameter for our architectural performance model is the *transaction commit time*: the time taken from submitting a transaction until we have sufficient

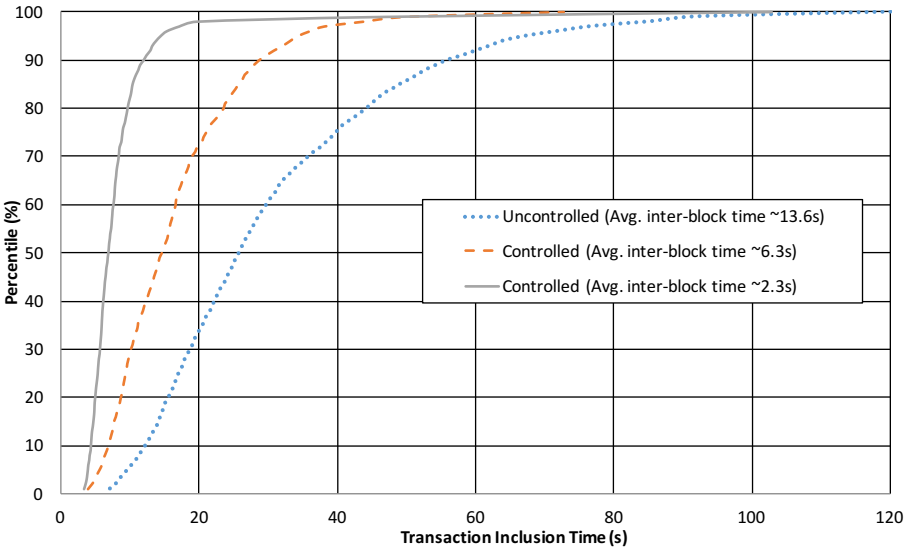


Figure 3.1: Transaction inclusion time measured on Ethereum (cumulative)

confidence that the transaction has been successfully included in the blockchain. We benchmark this in a representative deployment of the blockchain to be used by the client application, as described here. We start the clock on submission of a transaction, and stop the clock when the broadcasting node receives a sufficient number of confirmation blocks after receiving a block which includes the transaction. If 1 block is enough as confirmation, we call this *transaction inclusion time* instead. The total time will depend on the transaction propagation time, inter-block time, transaction inclusion probability, block propagation time, and the number of confirmation blocks. Our benchmark measurement abstracts from these details to create a transaction inclusion time distribution for our architectural performance model. Our benchmark measurements also include latency overhead for our trigger code and the communication between the trigger and the Ethereum node. However, this overhead is in milliseconds range, compared to the seconds inter-block time, so is not significant; and in any event, client applications using the blockchain encounter similar delays.

As discussed previously, the number of confirmation blocks is a design choice for client applications using a blockchain. Although twelve confirmation blocks are often recommended for the public Ethereum blockchain, the "right" number depends on the business risk involved in the transaction, and on other trade-offs with latency.

To demonstrate the approach, we ran benchmarks on a private Ethereum blockchain. We used a private deployment to prevent flooding the public Ethereum blockchain, to reduce our cost, and to be able to vary inter-block time. We used one virtual machine to deploy the trigger and a go-Ethereum (Geth) full node with mining disabled. The mining node was deployed on a different virtual machine. This situation would mimic practical deployment to some degree: each organisation would deploy their own full node and trigger in a virtual machine controlled by them, whereas miner node is operated on separate machines. Both

virtual machines run on one Intel(R) Xeon(R) CPU E5-2697 v3 @ 2.60GHz core each. The virtual machines were located in the same data center and had a LAN connection. The trigger was implemented in Node.js version 4.2.6 using Ethereum Javascript library (web3) version 0.15.3. Geth version 1.5.4-stable was used, and the trigger was configured to use Remote Procedure Calls (RPC) communication to interact with the Geth node.

For benchmarking latency, we submitted many transactions as follows. A script invoked the trigger API, which submitted the transaction. The trigger then listened to the blockchain for the announcement of a sufficient number of confirmation blocks after observing a block including the transaction and forwarded the result of successful inclusion back to the script. The script initiated the next transaction directly afterward.

As a baseline, we report here the observations of transaction inclusion time (i.e., where sufficient confidence of inclusion is judged to have occurred on seeing the transaction in a block, as defined above). We ran the experiment on a private blockchain, where we varied inter-block time, by either controlling the complexity mechanism or leaving the default implementation (uncontrolled). The mean *inter-block time* of the uncontrolled blockchain was 13.6s, in two settings of controlled private blockchain settings, we measured mean inter-block times of 2.3s and 6.3s.

For each of the three settings, we measured *transaction inclusion time* across 1000 transactions. The results are shown as cumulative distributions in Fig. 3.1. While median transaction inclusion time was 25.8s for uncontrolled private blockchain, it was 6.91s and 14.65s respectively for the two controlled private blockchains with 2.3s and 6.3s mean inter-block time. It should be noted that median transaction inclusion time would be higher on public blockchains, because of additional network delays and strategic transaction inclusion by miners.

3.2 Blockchain-Based System Performance Modelling

This section describes our approach for architectural performance modelling of blockchain-based systems. We illustrate our approach using architectural models for Weber et al.'s [25] method for business process execution on the blockchain.

In our approach, we model the blockchain from the perspective of the client application, as a component. So, we do not model the details of the blockchain mining network, node inter-communication, or consensus algorithm. All of these factors are aggregated in our abstract model and measurements. The client application interacts with the blockchain through a local blockchain node, and we model the resource and performance characteristics of this local blockchain node running as a component. In the architecture of a scalable client application, one may need to operate multiple blockchain nodes, each independently participating in the blockchain system; in such cases, we would model those as multiple deployed instances of the blockchain client. Note that these blockchain clients do not need to be resource-intensive mining nodes attempting to create new blocks on the blockchain. Instead, it is enough for these nodes to be just submitting and observing transactions and blocks on the blockchain network.

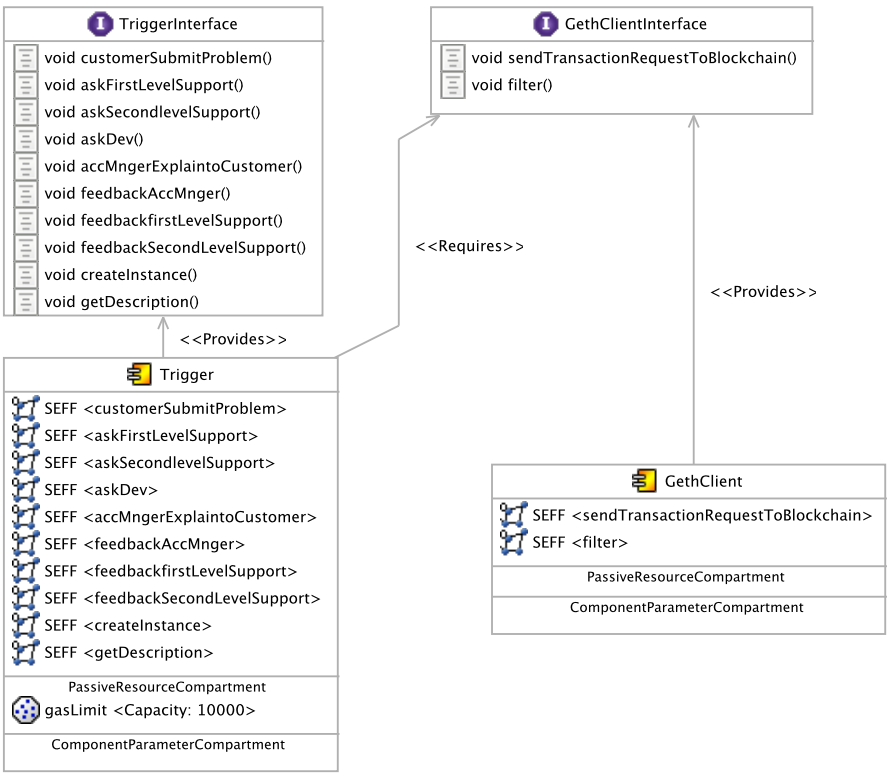


Figure 3.2: PCM Repository diagram

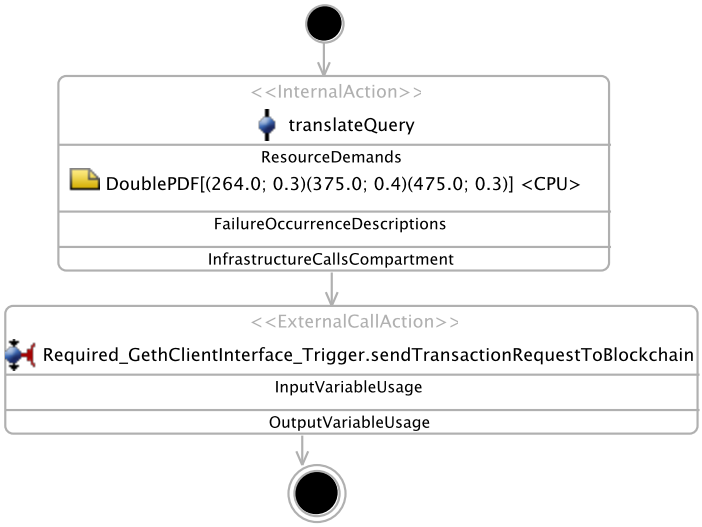


Figure 3.3: RDSEFF diagram of operation transaction

Component Repository Model

In Weber et al.'s [25] method, off-chain business systems interact with the blockchain through trigger components. Fig. 3.2 shows an example model of a trigger component interacting with an Ethereum blockchain client node (using the Ethereum geth client), modelled as two components each exposing a relevant interface. In a Palladio Component Model (PCM), operations provided by a component are specified in an interface. The trigger interface provides operations for each action in an example process. We here use an incident management process as an example, described in Section 4.1. The trigger interface also provides a *createInstance* operation, which creates an instance of a process monitor by invoking the factory smart contract for the business process, pre-configured on the blockchain. The trigger translates API calls into corresponding blockchain transactions and submits them for execution on the blockchain through the locally-deployed Ethereum client.

Resource Demanding Service Effect Specifications (RDSEFF)

After modelling the components, interfaces, and their relationships, we then model the non-functional behaviour of component operations. In PCM, the component operation behaviour is specified in a RDSEFF. Each operation translates an API call to a blockchain transaction and uses an external action to forward the transaction to the blockchain node, as illustrated in Fig. 3.3. The resource utilisation of each component is configured as a probability distribution function (PDF) constructed using benchmarks as described in Section 3.1. Each operation is benchmarked and modelled separately to account for variation in the operations and to demonstrate the capability of modelling their different behaviours. For manual steps in the process, operator resolution time must be separately benchmarked for inclusion into the model, but this is not dealt with

in this paper.

Usage Model

To simulate the execution of the system, we specify a usage model that captures representative use of the system at points of variation. Our illustrative usage model in Fig. 3.4 reflects process flow in our example business process for incident management, where the points of variation are optional branches of the process. For the purpose of our laboratory experiments, we assumed that at each stage of incident response (except for the final developer stage), 75% of issues received were resolved in that stage. The final developer support stage resolves every request. Fig. 3.4 shows the branching probabilities used to represent this behaviour.

Here we show a usage model as a single scenario with branching probabilities. Variation in the possible resolution times, e.g., due to randomness in the path taken, is explored through multiple simulation runs. However, it would also possible to examine multiple usage scenarios separately, each using different probabilities or execution/resolution times. This could be done to drill down onto specific issues or opportunities regarding the design of the business process.

4 Evaluating System-Level Latency Predictions

In this section, we evaluate the prediction accuracy of our performance model. We use an exemplar business process system and compare predictions from simulation with macro-level measurements. To this end, we used the same private Ethereum environment as for the micro-level benchmarking (Section 3.1). However, rather than measuring latency at the micro level for transaction inclusion time, we measure latency over entire business process execution instances and compare those to predictions from our PCM model.

For this work, only business process's end-to-end latency was considered. In order to evaluate the accuracy of scenario latency, the simulated and measured results are described using mean, median, quartiles and interquartile ranges, and the variance and standard deviation were excluded due to the skewness of the underlying distribution. The distributions are illustrated as cumulative distributions diagrams. Medians and quantiles are illustrated using boxplot diagrams. Additionally, relative errors are provided for each statistical values.

We reuse the incident management system from our previous work. To make this paper self-contained, a brief overview of the implementation is given in Section 4.1. The same implementation is used to evaluate the accuracy of performance models. The generalisability of this approach is further discussed in Section 5 for different business applications.

4.1 Incident Management Business Process Implementation

Throughout this paper, a case study process of Incident Management was used as an example, as shown in Fig. 4.1. Further details are discussed below and under experiment set up in Section 4.2.

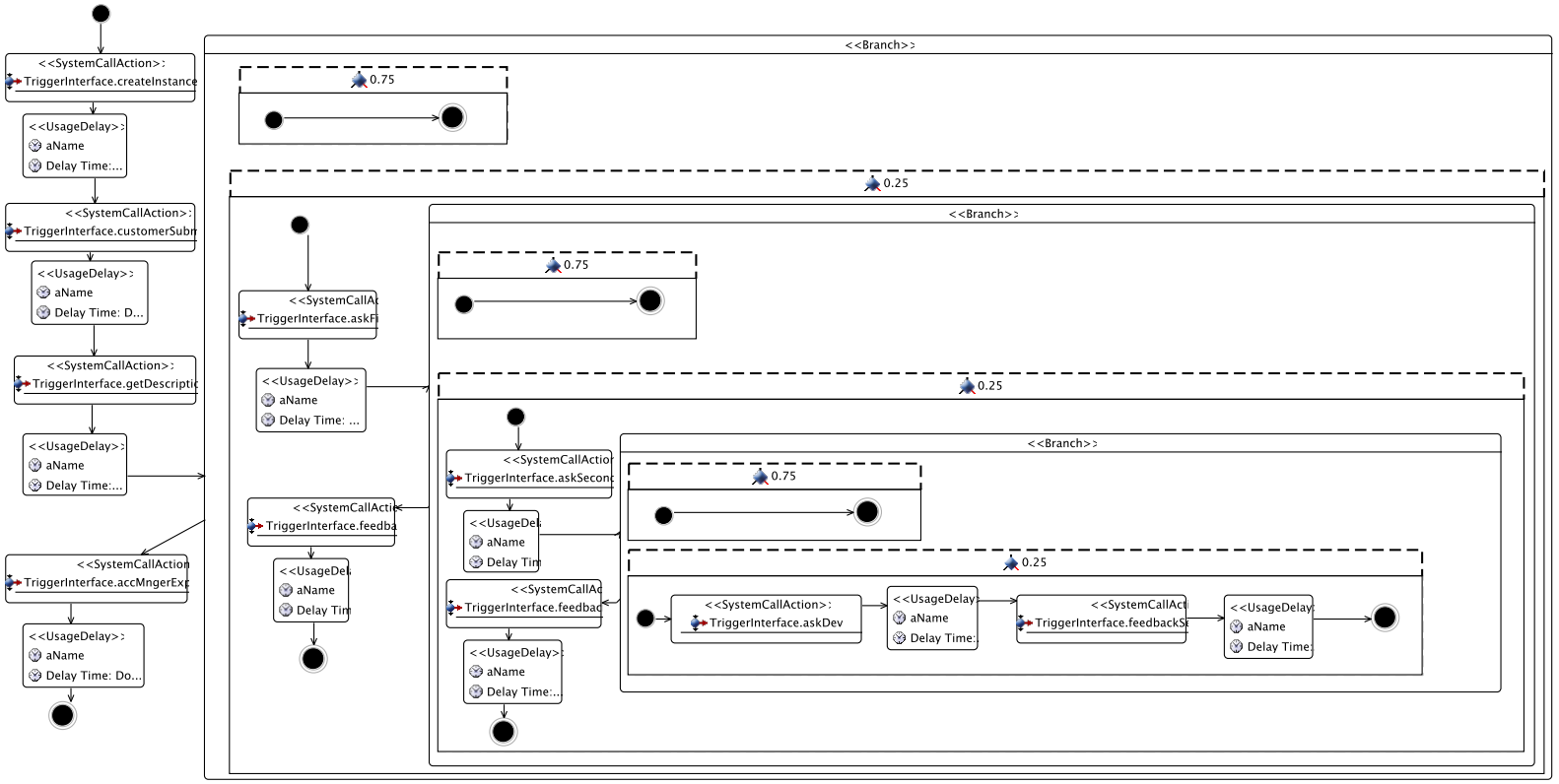


Figure 3.4: PCM usage model diagram of incident management business process

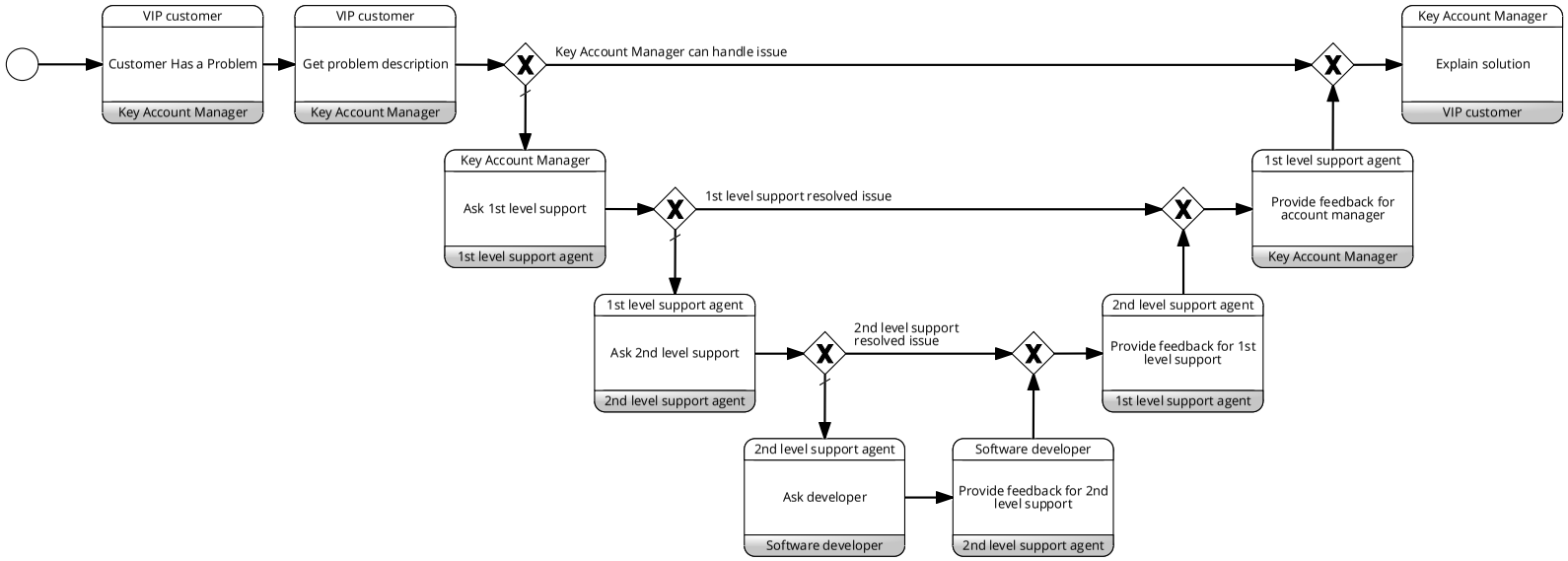


Figure 4.1: Incident management business process. Adapted from [18, p.18]

The process model is shown in Fig. 4.1. There are four issue resolution stages: account manager, first level support, second level support, and developer support. When a customer report an issue first, account manager requests problem description and attempt to solve the issue. If the issue is solved directly feedback the customer. Otherwise, account manager asks first level support and if first level support can not solve the issue ask second level support so on. At each stage, if someone finds the solution feedback to the upper level and finally account manager explains the solution to the customer.

In the blockchain, a trigger exposes APIs for each action. When a customer submits an issue, the trigger creates a smart contract instance. Other actors can interact with this smart contract instance via the trigger. The trigger updates the status of the process by sending transactions via a full blockchain node to that instance and keeps track of the process. For this experiment, it was assumed that all actors interact with the same trigger. In general, for each action, the trigger will invoke a transaction in a smart contract instance. When the transaction is included in a block and that block is confirmed by a predecided number of blocks, the system considers the action to have successfully completed.

The factory smart contract, which acts as a process monitor as discussed in Section 2.2, was compiled using Solidity compiler version 0.4.7 without enabling optimisations. The factory smart contract was manually deployed on the private blockchain and the trigger was configured to interact with the factory contract before starting the benchmarking.

4.2 Experiment Setup

Macro-level Measurements A synthetic workload was generated which follows the same 75% resolution rate at each stage as in Section 3.1. Trigger operations were invoked by HTTP requests using an external python script deployed separately was measured the time delay for the complete scenario. One second delay was injected in between two case initiations. The experiment was run for 1000 times (created 1000 process monitor instances) which was executed for approximately 20 hours.

Simulation SimuCom simulation engine was used for executing PCM model and executed for the same number of scenario executions. Simucom is the standard simulation engine for PCM model simulation which uses model-2-text transformations to translate PCM models into Java Code. Eclipse version 4.6.0 (Neon release) and Palladio Component model version 4.0.1 were used for modelling and SimuCom version 4.0.1 was used for simulations. The simulation was configured for the same number of measurements as the experiment (1000 measurements).

4.3 Comparing Measurement and Simulated Results

The measured and predicted latency distributions are illustrated as a cumulative density diagram in Fig. 4.2. The cumulative distribution of the latency is informative as it shows the percentile of process executions under specific time. As shown in Fig. 4.2 the cumulative distribution highly coincides with the results from the benchmark.

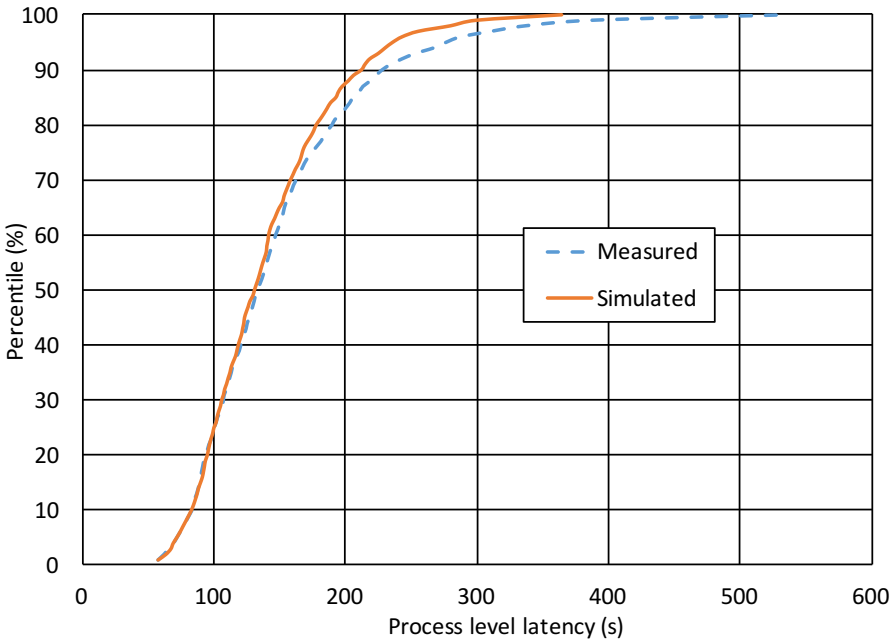


Figure 4.2: Scenario latency - Cumulative distribution

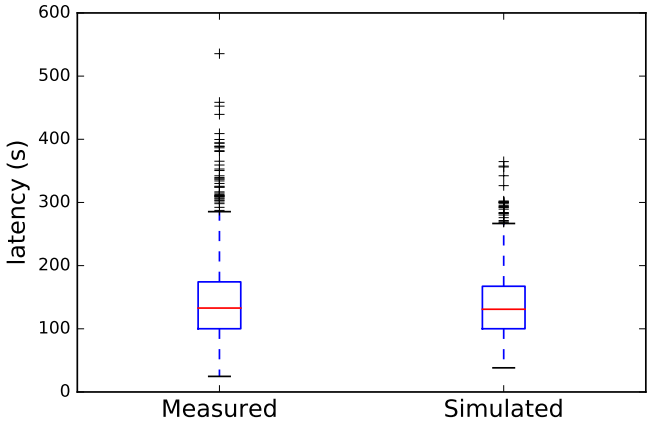


Figure 4.3: Boxplot diagrams of measured and simulated scenario latency - Measured median is 132.83s, simulated median is 130.93s, relative error of median is 1.42%, and relative error of 95^{th} percentile is 14.6%

The simulation predicted the mean latency of the process scenario with a relative error of 1.6%. The measured mean latency of the process was 136.29s and the simulation predicts the mean latency as 134.08s where the standard error of mean(SEM) is 1.27 and 1.07 respectively.

Further statistical measures are illustrated as boxplot diagram in Fig. 5.1a which consists of a box whose bounds denote the first quartile (Q_1) and third quartile (Q_3) of the underlying data sample. Medians are denoted by a horizontal line within the box. Maximums and minimums indicate by vertical lines outside the box (whiskers). For many applications, 95% and 99% percentiles are significant measures when considering the latency and the skewness of the distribution. The PCM model predicted the 95% and 99% percentiles with a relative error of 9.4% and 11.5% accuracy. Error in predicted maximum and minimum are respectively 7.62% and 16.89%.

5 Architectural Decision Making

Design alternatives can be evaluated by predicting latency in example scenarios. This lets us explore *what-if* questions in architectural decision making. Xu et al. [29] have discussed blockchain system design alternatives, and the impact of design decisions on quality attributes. Here we focus on latency.

5.1 Choice of Inter-Block Time

In a public blockchain, the target inter-block time is fixed. However, in private blockchains, it can be varied as a design choice. This reduces transaction inclusion time, which can reduce system-level latency. When evaluating inter-block time alternatives, we use the same models, but modify the transaction inclusion time parameter.

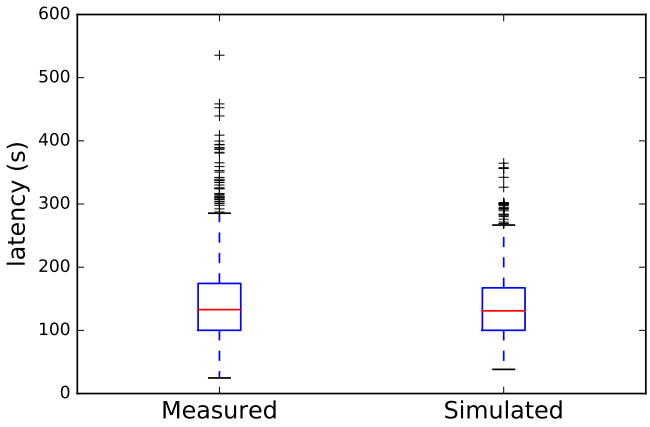
We conducted an experimental evaluation of the accuracy of our simulation for various transaction inclusion times, on a private blockchain. The results are shown as boxplots in Fig. 5.1.

For the default inter-block time of 13.6s, the measured and simulated median process latency was 132s and 130.93s respectively. For inter-block times of 6.3s and 2.3s, the median measured latencies were 64.7s and 28.1s, and the median simulated latencies were 71.1s and 30.7s respectively. The relative errors of median were 1.4%, 9.6%, and 9.4%, while the relative error of 95^{th} percentiles were 14.6%, 8.5%, and 0.7% respectively.

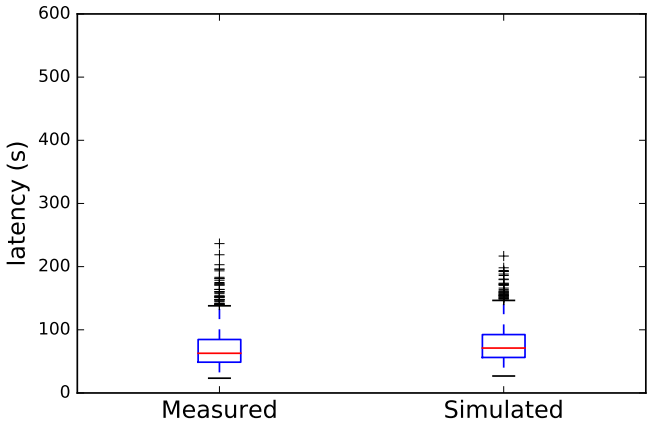
5.2 Choice of Number of Confirmation Blocks

The vulnerability of blockchain-based systems to double-spending attacks can be reduced by increasing the number of confirmation blocks [23]. This introduces additional latency to the system. We measured the transaction commit time with 6 and 12 blocks separately and populated the model as mentioned above. We ran a separate experiment for benchmarking the latency of the BP with 1, 6, and 12 confirmation blocks. The results are illustrated as boxplot diagrams in Fig. 5.2.

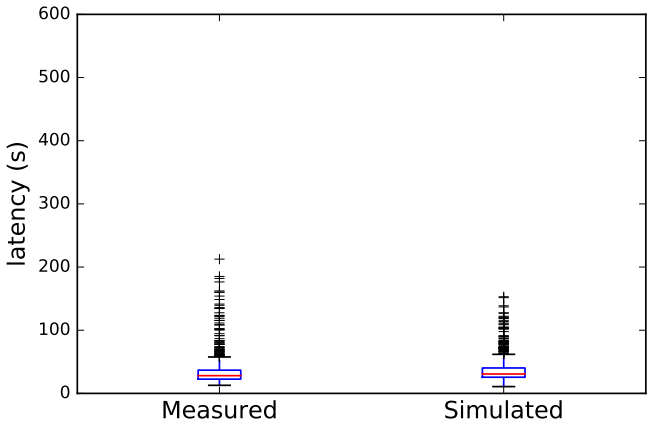
We used a controlled blockchain for this experiment with mean inter-block time of approx. 2.3s. The measured median process latencies with 1, 6, and 12



(a) For 13.6s average inter-block time: median time (measured 132s, simulated 130s), relative error (median 1.4%, 95$^{\text{th}}$ percentile 14.6%)

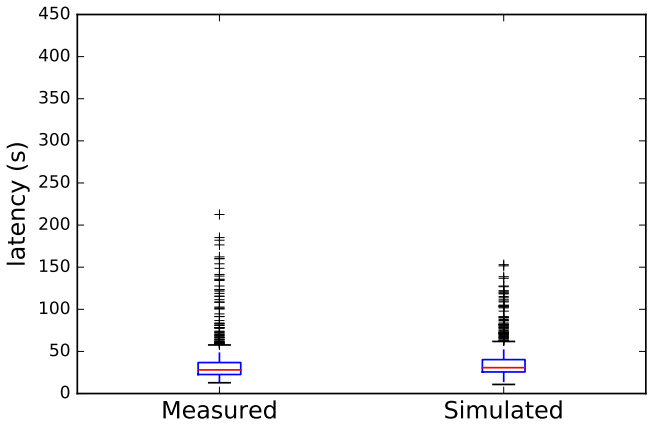


(b) For 6.3s average inter-block time: median time (measured 64.7s, simulated 71.1s), relative error (median 9.6%, 95$^{\text{th}}$ percentile 0.7%)

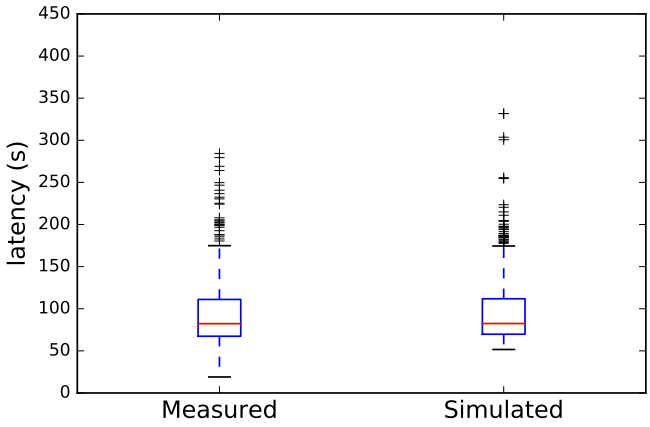


(c) For 2.3s average inter-block time: median time (measured 28.1s, simulated 30.7s), relative error (median 9.4%, 95$^{\text{th}}$ percentile 8.5%)

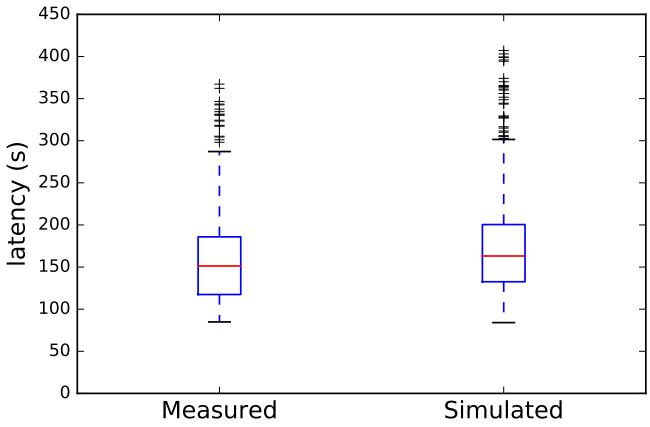
Figure 5.1: Boxplot diagrams of measurement and simulated results for transaction inclusion time under various inter-block times.



(a) 1 confirmation block: median time (measured 28.1s, simulated 30.7s), relative error (median 9.4%, 95$^{\text{th}}$ percentile 8.5%)



(b) 6 confirmation blocks: median time (measured 81.5s, simulated 81.7s), relative error (median 0.2%, 95$^{\text{th}}$ percentile 6.7%)



(c) 12 confirmation blocks: median time (measured 152s, simulated 164s), relative error (median 0.2%, 95$^{\text{th}}$ percentile 12.3%)

Figure 5.2: Boxplot diagrams of measured and simulated confirmation time for varying numbers of confirmation blocks. Average inter-block time was 2.3s.

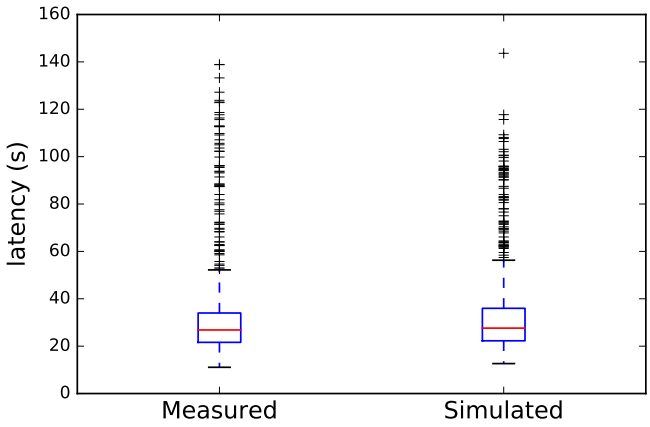


Figure 5.3: Measured and simulated latency of modified BP. Median time (meas. 26.8s, sim. 27.6s), relative error (median 2.9%, 95$^{\text{th}}$ percentile 0.3%).

blocks confirmation were 28.1s, 81.5s, and 152s respectively while the simulation predicted 30.7s, 81.7s, and 164s. The relative errors of median predictions were 9.4%, 0.2%, and 0.2% and the relative errors of 95^{th} percentiles were 8.5%, 6.7%, and 12.3% respectively.

5.3 Process Level Changes

It is straightforward to use architectural performance models to evaluate process-level changes. In our approach, the process is defined by the Palladio usage model. Performance models can also be useful for estimating the impact on latency of process redesign [22] such as task elimination, process integration, or task composition. These are modelled by changing the workflow. Most of the BP control flow patterns [24] can be directly translated to Palladio component model patterns.

We experimented with a changed process model, where the account manager assigns issues directly to second-level support (skipping the first level) in 5% of cases. We used a private Ethereum blockchain with a mean inter-block time of 2.3s. The results are shown in Fig. 5.3. The median process latency was measured as 26.8s, vs. simulation as 27.6s. The relative error of median was 2.9% and the relative error of 95^{th} percentile was 0.3%.

6 Discussion and Future Work

In this section, we discuss some of the limitations of our approach and evaluation, and propose possible extensions and other uses for our performance models.

6.1 Limitations

The transaction inclusion-time benchmarks we reported in Section 3 are not intended to be generalisable. Instead, they illustrate our approach to benchmarking transaction inclusion time in order to configure an architectural performance model. In particular, our laboratory experiments were performed on a private deployment of Ethereum with only one mining node. This means that there are no significant network delays for transaction or block propagation among peers, and there is no occurrence of *uncles* (short-lived alternate competing recent histories). The occurrence of uncles can affect transaction inclusion time. We recommend benchmarking end-to-end latency in the target blockchain platform in order to account for all sources of delay and variation in transaction inclusion.

Similarly, our experiments on Ethereum use Nakamoto consensus and proof-of-work. We expect our modelling approach would be usable for proof-of-stake consensus, after benchmarking transaction inclusion times in those systems. Our general approach would be applicable in blockchains using classical distributed consensus algorithms, but the stronger transaction commit semantics supported by those algorithms means that confirmation blocks would not be required.

Our focus in this paper is on latency, not throughput or scalability. We have therefore benchmarked latency and evaluated predictions under low demand. In our experiments, we observed low CPU load, so assume that CPU utilisation did not impact latency. In real-world situations, latency is affected by high demand, resource bottlenecks, and architectural mechanisms (e.g., load balancing) used for scalability. We expect that for a particular use case, if a representative load can be used on a representative deployment of a blockchain, then latency benchmarking could be performed as we have described in this paper. A full treatment is left for future work.

In a blockchain-based system, in addition to CPU, network, and disk there are other resources. We have not modelled smart contract gas consumption, gas limits, and public blockchain transaction fees, although these may able to be modelled as passive resources, as discussed below in Section 6.3.

We have used only one example system for this initial evaluation of our approach. However, the architectural performance and simulation approach we have used is largely consistent with the previous body of work in this field which has been applied to a variety of application systems [3, 20]. Our approach should similarly apply to other systems.

6.2 Evaluating Other Kinds of Design Alternatives

In Section 5 we showed that the architectural performance modelling approach can be used to simulate design alternatives for blockchain configuration, and provided empirical evaluation about the accuracy of those simulation results. Palladio can be used to model and predict resource utilisation and to compare different deployment architectures. Frameworks such as PerOpteryx [14] can be used to automatically search for and define architectures that maximise predicted system performance. We do not illustrate automatic optimisation in this paper. However, we do discuss below how the simulation model may be used to explore some other design issues.

Using Blockchain as a Component

using blockchain as a software component is an architectural decision [29]. As discussed earlier, a challenge is that latency can be significantly higher for blockchains that for traditional databases. Another concern will be uncertainty about the upper bound for latency. Acceptable latency levels depend on the system requirements and should be considered before using a blockchain.

Performance models can help us compare design alternatives. The off-chain portion of a system can be modelled conventionally. We have used standard functionality for modelling, so off-chain and on-chain components can appear in the same model. This can aid visualisation and improve system understanding [8].

The accuracy of performance models of traditional cloud and database based systems has been previously studied [6, 8]. The off-chain portion of a blockchain-based system can be modelled using these approaches. The granularity of the system model involves a trade-off between model complexity, simulation time, and the accuracy of the predictions.

One approach to assess the impact of introducing blockchain to a traditional system is to construct the performance models for the traditional systems using conventional methods, and then injecting the transaction commit delays in the places where blockchain interactions will be added. This facilitates the evaluation of the overall impact to the system, if the blockchain is added as a component, but with a slightly higher prediction error. If further precision is needed, the models can be constructed following the methodology in Section 3.

6.3 Execution Cost Modelling

Estimating the cost of software systems using performance modelling and simulation has been previously studied [8, 15]. Component cost and hardware cost can be modelled for blockchain-based systems in the same way. However, blockchain-based systems have other costs including blockchain gas cost for invocation and deployment (*Gas* is the internal currency for transaction or contract usage fees in Ethereum [27]). The cost model for blockchains is different to conventional systems, and so simulation-based approaches for cost modelling are expected to be useful design tools.

The gas consumption of a transaction or smart contract creation depends on the size of the payload (≈200 gas per byte), the gas price for address allocations if it involves contract creation (32,000 gas), the complexity of the smart contract, and the base transaction gas price (21,000 gas). The details of the cost associated with contract creation are provided by Wood [27]. *Ether* is the crpyto-currency for Ethereum, analogous to Bitcoins. Gas translates to Ether with a *gas price*, and Ether to USD or other currencies through exchange rates – thus, any operation on the public Ethereum blockchain can be understood to cost real money. Generally, the use of smart contracts is more expensive than regular function invocation.

One way to model gas consumption may be as a passive resource in Palladio. A large amount of gas could be allocated at the beginning and be consumed in each operation (which should be estimated analytically or by running a benchmark). Simulation results can show the state of each passive resource over the time, which here would reflect gas consumption. Currently, PCM does not sup-

port bulk acquisition of passive resources. This could be modelled using looping and by passing the gas amount as a parameter for each operation.

6.4 Modelling Block Gas Limits

In Ethereum, the block gas limit is a parameter defined by miners, which can be a limiting factor to throughput and thus latency of the system, particularly if the transaction frequency and gas consumption per transaction are high. Modelling gas limit is nontrivial for public blockchains: the gas limit is dynamic and the gas amount that remains available in a block depends on the interactions of other parties.

7 Conclusion

In this paper, we have proposed and evaluated an approach for predicting the latency of blockchain-based systems using architectural performance modelling and simulation. For an illustrative experimental system in a laboratory environment, our predictions had a relative error of mostly under 10%. We further demonstrated the capability of using these performance models to support evaluation of design alternatives that would be encountered in architectural design. Some of these decisions are about blockchain-specific issues, such as inter-block time or the number of confirmation blocks. Some decisions about a blockchain-based system may be about system-level design options but are impacted by latency arising from the blockchain-related factors. The proposed architectural models also provide a basis for future research into optimal system configuration, cost, and other non-functional properties. This technical report serves as long version of a conference paper [30].

Bibliography

[1] Ether stats, November 2016.

[2] Palladio component model addons, October 2016.

[3] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[4] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S Trivedi. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons, 2006.

[5] Andreas Brunnert, André van Hoorn, Felix Willnecker, Alexandru Danciu, Wilhelm Hasselbring, Christoph Heger, Nikolas Herbst, Pooyan Jamshidi, Reiner Jung, Joakim von Kistowski, et al. Performance-oriented devops: A research agenda. *arXiv preprint arXiv:1508.04752*, 2015.

[6] Andreas Brunnert, Christian Vögele, and Helmut Krcmar. *Automatic Performance Model Generation for Java Enterprise Edition (EE) Applications*, pages 74–88. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[7] Vitalik Buterin. Ethereum white paper: a next generation smart contract & decentralized application platform. *www3.ethereum.org*, 2013.

[8] Thijmen De Gooijer, Anton Jansen, Heiko Koziolek, and Anne Koziolek. An industrial case study of performance and cost design space exploration. In *Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering*, pages 205–216. ACM, 2012.

[9] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International Conference on Financial Cryptography and Data Security*, pages 436–454. Springer, 2014.

[10] Greg Franks, Tariq Al-Omari, Murray Woodside, Olivia Das, and Salem Derisavi. Enhanced modeling and solution of layered queueing networks. *IEEE Transactions on Software Engineering*, 35(2):148–161, 2009.

[11] Luciano García-Bañuelos, Alexander Ponomarev, Marlon Dumas, and Ingo Weber. Optimized execution of business processes on blockchain. *arXiv preprint*, 2016.

[12] Vincent Gramoli. On the danger of private blockchains. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers (DCCL'16)*, 2016.

[13] Samuel Kounev, Fabian Brosig, and Nikolaus Huber. The Descartes modeling language. *Dept. of Computer Science, University of Wuerzburg, Tech. Rep*, 2014.

[14] Anne Koziolek, Heiko Koziolek, and Ralf Reussner. PerOpteryx: automated application of tactics in multi-objective software architecture optimization. In *Proceedings of the joint ACM SIGSOFT conference–QoSA and ISARCS*, pages 33–42. ACM, 2011.

[15] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proc. of Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 105–116. ACM, 2010.

[16] Michael K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on computers*, 100(9):913–917, 1982.

[17] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[18] Object Management Group. BPMN 2.0 by Example. www.omg.org/spec/BPMN/20100601/10-06-02.pdf, June 2010. Version 1.0. Accessed 10/3/2016.

[19] Steve Omohundro. Cryptocurrencies, smart contracts, and artificial intelligence. *AI matters*, 1(2):19–21, 2014.

[20] Juan F Pérez and Giuliano Casale. Assessing SLA compliance from Palladio component models. In *Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), 2013 15th International Symposium on*, pages 409–416. IEEE, 2013.

[21] Marc Pilkington. Blockchain technology: principles and applications. *Research Handbook on Digital Transformations, edited by F. Xavier Olleros and Majlinda Zhegu. Edward Elgar*, 2016.

[22] Hajo A. Reijers and S. Liman Mansar. Best practices in business process redesign: an overview and qualitative evaluation of successful redesign heuristics. *Omega*, 33(4):283–306, 2005.

[23] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint*, 2014.

[24] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[25] Ingo Weber, Xiwei Xu, Rgis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. Untrusted business process monitoring and execution using blockchain. In *Intl. Conf. Business Process Mgmt.(BPM), Rio de Janeiro, Brazil*, 2016.

[26] Felix Willnecker, Andreas Brunnert, and Helmut Krcmar. Predicting energy consumption by extending the Palladio component model. In *SOSP14 Symposium on Software Performance: Joint Descartes/Kieker/Palladio Days 2014*, page 177, 2014.

[27] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 2014.

[28] Jing Xu, Murray Woodside, and Dorina Petriu. Performance analysis of a software design using the UML profile for schedulability, performance, and time. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 291–307. Springer, 2003.

[29] Xiwei Xu, Cesare Pautasso, Liming Zhu, Vincent Gramoli, Alexander Ponomarev, An Binh Tran, and Shiping Chen. The blockchain as a software connector. In *Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2016.

[30] Rajitha Yasaweerasinghelage, Mark Staples, and Ingo Weber. Predicting latency of blockchain-based systems using architectural modelling and simulation. In *IEEE International Conference on Software Architecture (ICSA)*. IEEE, 2017.