What Blocks My Blockchain's Throughput? Developing a Generalizable Approach for Identifying Bottlenecks in Permissioned Blockchains

Orestis Papageorgiou
SnT – University of Luxembourg
orestis.papageorgiou@uni.lu

Lasse Börtzler
Karlsruhe Institute of Technology
lasse.boertzler@student.kit.edu

Egor Ermolaev
SnT – University of Luxembourg
egor.ermolaev@uni.lu

Jyoti Kumari
SnT – University of Luxembourg
jyoti.kumari@uni.lu

Johannes Sedlmeir
SnT – University of Luxembourg
johannes.sedlmeir@uni.lu

Abstract

Permissioned blockchains have been proposed for various use cases where a certain degree of decentralization is necessary yet enterprise IT requirements must be met. However, their throughput remains considerably lower than that of established centralized systems. Previous studies that address permissioned blockchains' performance remain blockchain-specific, lacking a generalizable approach for locating and understanding bottlenecks. This paper presents a unified, graphical method for identifying bottlenecks in permissioned blockchains. We augment the DLPS – an open-source benchmarking tool – with graphical evaluation functionalities and use them to identify performance bottlenecks of Hyperledger Fabric and Quorum, two widely used permissioned blockchains with distinct architectural designs. Our work provides researchers and practitioners with a toolkit, guidelines on blockchain performance data analytics, and insights that assist with the bottleneck identification and improvement of permissioned blockchains.

Keywords: Blockchain, Distributed Ledger, Performance Evaluation, Hyperledger Fabric, Quorum

1. Introduction

Since its inception by Nakamoto (2008), blockchain technology has been explored across various industries far beyond its use in the cryptocurrency Bitcoin. Researchers and practitioners have analyzed its potential in a variety of applications where a neutral platform is desirable (Sedlmeir et al., 2022b), such as supply chain management (Queiroz et al., 2020) and the streamlining of cross-organizational workflows (Fridgen et al., 2018). Organizations looking to implement blockchain-based information systems often opt for permissioned blockchains as they restrict participation in consensus, reduce data visibility and latency, and allow for better throughput. However, transitioning projects based on permissioned blockchains from pilot stages to business applications still presents many technical challenges (Toufaily et al., 2021). One major obstacle is the technology's inferior performance compared to established centralized systems, stemming from the resource-intensive nature of replication and consensus (Sedlmeir et al., 2022a). As a result, a substantial part of research focuses on examining the performance characteristics of permissioned blockchains (Fan et al., 2020). Blockchain benchmarking research has primarily examined high-level performance indicators such as throughput, with less emphasis on identifying performance-limiting factors. While assessing throughput is useful for comparing different blockchains and assessing deployment parameters (Guggenberger et al., 2022), it provides limited insights into crucial aspects like node resource utilization, which are usually reported only as aggregated data (Fan et al., 2020), offering a limited understanding of the inner workings of blockchain performance.

This paper addresses this shortcoming by analyzing resource-related metrics of blockchain nodes and their impact on throughput in a systematic and graphical way, providing a general approach that can be used to detect bottlenecks. We survey related work to ground our method and then conduct *exploratory data analysis* (EDA) to determine the performance bottlenecks of Hyperledger Fabric (Fabric) and Quorum, using an extended version of the distributed ledger performance scan (DLPS) (Sedlmeir et al., 2021).

URI: https://hdl.handle.net/10125/109740
978-0-9981331-8-8

2. Background

2.1. Hyperledger Fabric

Fabric has become one of the industry's leading permissioned blockchains (Guggenberger et al., 2022), known for its unique architecture that provides ample opportunities for finetuning performance (Androulaki et al., 2018). In Fabric, nodes are grouped into organizations, and a node can take at least one of the roles of a peer node (*peer*) or an orderer node (*orderer*) (Androulaki et al., 2018). Peers maintain an append-only ledger and a corresponding running aggregate (state), whereas orderers create and broadcast blocks. Fabric relies on the *execute-order-validate* paradigm that involves three phases. In the *Execution Phase*, a client sends a signed transaction proposal to the peers. Peers *simulate* the transaction, i.e., they run the required smart contract ("chaincode") on their current version of the state without updating it. Peers then respond to the client with a signed *endorsement* containing the peer's digital certificate, the transaction's read-write set, and the simulation outcome (Androulaki et al., 2019). During the *Ordering Phase*, and once a client has collected sufficient endorsements according to a chaincode's policy, it packs them into a transaction and forwards it to the ordering service. Orderers use a consensus protocol, such as RAFT (Ongaro et al., 2014), to sort transactions, group them into a batch ("block"), and sign this block without evaluating the transactions' validities. Subsequently, they broadcast the block to a subset of peers (to one *anchor peer* per organization) for validation. In the *Validation Phase*, upon receiving a block – either directly from an orderer or via a fellow peer – a peer validates the included transactions in three steps (Thakkar et al., 2018). First, through parallel verification, *validation system chaincode* (VSCC) ensures transactions have the required endorsements and consistent execution results. Next, valid transactions undergo *multi version concurrency control* (MVCC) – a sequential check of whether the simulations were conducted on compatible ledger versions by comparing the respective read-write sets. Finally, each peer commits the transaction to their local ledger (including a flag for its validity) and – if both checks have passed – updates its state accordingly.

2.2. Quorum

Quorum is another blockchain that has emerged as a significant player among the industry's permissioned blockchains because of its similarity to the public Ethereum blockchain. Unlike Fabric, Quorum relies on the *order-execute* paradigm. It supports four different consensus mechanisms, including RAFT. In the *Ordering Phase*, clients send signed transactions to the nodes, which perform preliminary validations (e.g., correct nonce, syntax, and signatures). Verified transactions are shared with other nodes via a gossip protocol and added to their unconfirmed transaction pools ("mempool"). The RAFT leader sorts and batches valid transactions from its mempool, compiles them into a block, and disseminates the block to follower nodes. Followers attach the received block to their ledger and send acceptance messages to the leader. After receiving acceptance messages from the majority of nodes, the block becomes the new head of the blockchain ("2-phase commit"). During the *Execution Phase*, each node then updates its state deterministically according to each included transaction.

2.3. Distributed Ledger Performance Scan

The DLPS is an open-source, end-to-end benchmarking framework where users can define specifications for various blockchain and client network configurations using a single configuration file (Sedlmeir et al., 2021). We simplified its complex deployment process by dockerizing both the deployment and experiment handlers and extended its graphical capabilities for analyzing performance and resource utilization data.[1] The benchmarking follows a recursive localization of maximum throughput by gradually increasing the request rate to determine the network's maximum throughput. The first run starts by sending (asynchronous) requests at a base rate. The DLPS measures each component's (e.g., nodes, clients) resource utilization, as well as the average request frequency (f_{req}) and response frequency (f_{resp}), of successful transactions by collecting request and response timestamps from all clients. To achieve this, the DLPS leverages system monitoring tools available by the operating system, such as `vmstat`, `mpstat`, `sar`, and `ping`, to gather comprehensive performance metrics on CPU usage, memory, network activity, and I/O operations. If f_{resp} does not deviate from f_{req} by more than a given threshold, the next run increases f_{req}. Otherwise, a certain number of retries is performed. If f_{resp} fails to get close to f_{req} also in the retries, the ramp-up sequence is terminated and the maximum throughput is set to the highest average f_{resp} in any of the runs of the previous ramp-up sequence. After the experiment, the DLPS stores fine-granular data in CSV format for further analysis and generates summary figures for illustrative purposes.

[1] The source code and higher resolution figures are available at: `https://github.com/orepapas/What_Blocks_My_Blockchains_Throughput_Data`.

3. Related work

To gain an overview of the academic literature on bottleneck identification in permissioned blockchains, we conducted a systematic literature review. We used the broad search string *(blockchain OR "distributed ledger technology") AND (performance OR throughput OR latency) AND (benchmarking OR measurement OR evaluation OR analysis)* on Google Scholar, ACM Digital Library, IEEE Xplore, and arXiv. Our search yielded 4,248 results. After manually reviewing titles and abstracts and removing duplicates, 57 publications remained. For these, we performed a full-text screening and excluded articles that did not provide empirical results on blockchain performance. Additionally, we excluded papers that lacked insights into potential bottlenecks, such as papers focusing on comparisons of different configurations or comparisons between different blockchains. For Fabric, we excluded research on v0.6 since it still used the order-execute architecture. We ended up with six relevant publications.

In Fabric v1.0, Androulaki et al. (2018) identify the validation phase and, in particular, the VSCC as a major bottleneck. Thakkar et al. (2018) find three major bottlenecks of v1.0, which are related to the validation phase and were addressed in subsequent Fabric versions. Ruan et al. (2020), using Fabric v1.3, also point to the validation phase as the bottleneck, especially when many unserializable transactions are included in the ledger. Wang et al. (2020) find that in v1.4, the VSCC remains the bottleneck due to limited parallelization. For the same version, Chacko et al. (2021) trace transaction failures to systemic issues, with the validation phase being the bottleneck. Specifically, MVCC read conflicts result in transaction failure, necessitating a return to the execution phase for a new round of endorsements and endorsement policy failures, which significantly slow down the VSCC and transaction processing. We could not find research focusing on identifying bottlenecks in Fabric v2.0 or higher. For Quorum, Mazzoni et al. (2021) posit that a potential bottleneck lies with the node's remote procedure call (RPC) server buffers being capped at 128KB. While this limitation suggests a maximum transaction size of 128KB, it is improbable to be the main bottleneck, as average transaction sizes are only a few hundred bytes.

4. Method and Results

We developed our approach by analyzing the experiments of the papers mentioned in Section 3, using the DLPS to collect data, and EDA (Chatfield, 1986) to identify bottlenecks. We collected extensive data on factors influencing node performance, ensuring a comprehensive overview of resource utilization. After cleaning and validating the data, we used EDA to detect performance irregularities. We then examined the relevant metrics to determine potential root causes. We divided our analysis into two major parts. The first part identifies potential bottlenecks by analyzing the different node resources that can impact blockchain performance. In the second part, we examine the relationship between these candidates and the blockchain's throughput in varying degrees of resolution, such as different time windows and component selections.

Since many enterprise applications have focused on Fabric and Quorum, and Section 3 indicates that their bottleneck analysis is intricate, we detail our bottleneck identification method for both blockchains. We selected an experiment for Fabric v2.0, based on the findings of (Guggenberger et al., 2022) to determine a configuration that seems robust under modifications and extended it to Quorum v23.4. For Fabric, the network configuration comprised 16 clients, 8 peers, and 4 orderers, with four organizations comprised of four clients, two peers and one orderer each. The Quorum configuration consisted of 16 clients and 8 nodes. In both configurations, we selected RAFT as consensus mechanism due to its minimal overhead, as previous publications suggest that consensus is not the bottleneck in this case (Guggenberger et al., 2022; Mazzoni et al., 2021). We conducted the experiment on the AWS cloud platform (Amazon EC2), where each node was configured in an independent EC2 instance allocated with 16 vCPUs, 64 GB of RAM, and 1 Gbps of bandwidth running Ubuntu Server 18.04 LTS (HVM) – a typical configuration for enterprise blockchain nodes.

4.1. Fabric: Resource utilization

The initial analysis focuses on the impact of incremental increases in request rate on the resource utilization of each node. We plot data points for each node every second during a 14-second period within a 20-second experiment, excluding the first and last three seconds to avoid distortions related to the discontinuity of f_{req} (Figure 1). This approach aims to uncover trends and correlations between resource usage and increased f_{req}. We directly see that Fabric peers' central processing unit (CPU) and network utilization exhibit clear plateaus, indicating they could be bottlenecks.

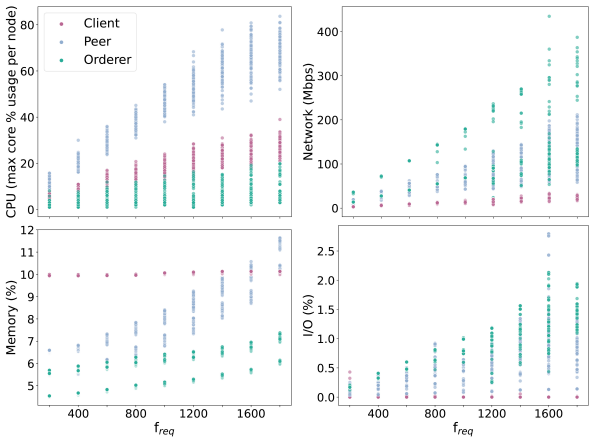


Figure 1: Fabric – Nodes' key resource utilizations.

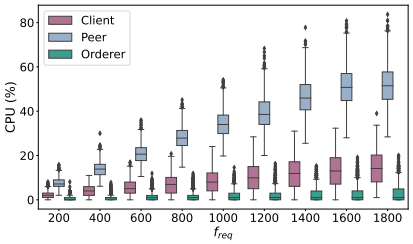
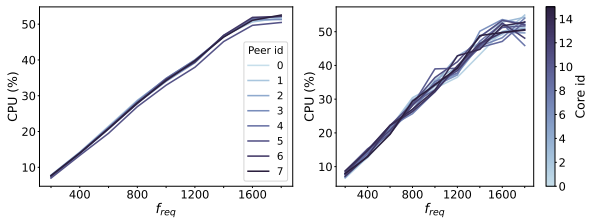


Figure 2: Fabric – CPU utilization of all cores.

4.1.1. CPU The analysis begins by examining the average CPU utilization across a 14-second period. We observe that the linear relationship between the request rate and CPU utilization of peers breaks down at $f_{req} \approx 1600\,s^{-1}$, indicating a potential saturation point or limitation in CPU capacity. Looking into the CPU usage across individual cores (Figure 2) reveals a limited utilization for orderers and clients that is not plateauing at higher request rates. Focusing on peers, we observe a significant variance in CPU utilization, ranging from 25 % to 80 % at higher request rates. This variability suggests an uneven distribution of resources, with some peers bearing a heavier workload than others or an imbalanced allocation of tasks within some peers' cores. First, we investigate the CPU utilization per peer in Figure 3a, which indicates an equitable distribution of computational resources among the peers, with peer 5 falling behind slightly. Analyzing the mean CPU utilization of individual cores for a single peer (peer 0 in this case) in Figure 3b reveals that this also is not the cause of the high fluctuations in CPU usage since all cores show similar utilization. Because of the inconclusive results of both possible explanations, our analysis progresses to evaluate the temporal evolution of CPU usage across individual cores at the highest



(a) Mean utilization per peer. (b) Mean utilization of peer 0 per core.

Figure 3: Fabric – peer CPU utilizations.

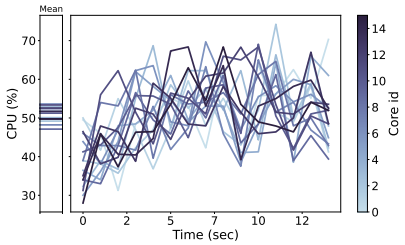
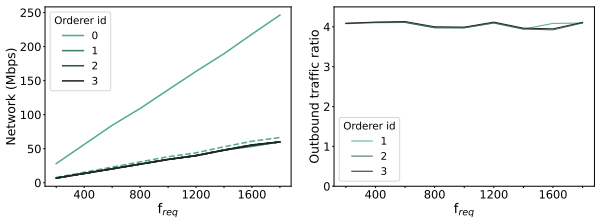


Figure 4: Fabric – Peer 0 CPU utilization for f_{req}=1600 s^{-1}.

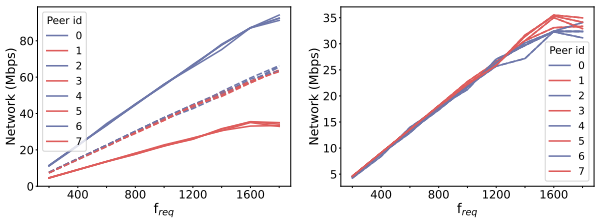
f_{req} before the utilization plateaus (Figure 4). This f_{req} represents the peak stress on the network before any performance degradation. Here, we observe that individual cores' CPU utilization can fluctuate as much as 30 % within a single run. We are unable to deduce the reasons behind these fluctuations from CPU utilization data alone. However, as these fluctuations balance out over time according to the narrowly distributed mean CPU utilization, they are likely not the reason for the plateau. It is worth noting that Figure 3 indicates that average CPU usage plateaus at around 50 % across all cores on all peers, an improvement over previous Fabric versions. However, in scenarios utilizing a higher number of vCPUs (16 in our case), Fabric v2.0 still demonstrates a mediocre mean utilization, leaving room for further improvement (see also Thakkar et al. (2018)).

4.1.2. Network The network-related analysis begins by looking into the mean network utilization, distinguishing between inbound and outbound traffic for the different types of nodes. Orderers' traffic does not plateau at high request rates, suggesting the ordering service is not the bottleneck (Figure 5a). Notably, orderer 0 broadcasts a disproportionate amount of traffic compared to the rest, which indicates that orderer 0 is the RAFT leader and, as a result, has the additional task of broadcasting new blocks to each following orderer. Detailed traffic analysis allows its decomposition into individual components, such as the traffic generated by block propagation. According to



(a) Mean utilization per orderer . (b) Ratio of leader outbound traffic over follower outbound traffic.

Figure 5: Fabric – orderer mean network utilization (outbound traffic as continuous and inbound traffic as dashed lines).



(a) Mean utilization per peer. (b) Outbound traffic per peer .

Figure 6: Fabric – peer mean network utilization (outbound traffic as continuous and inbound traffic as dashed lines).

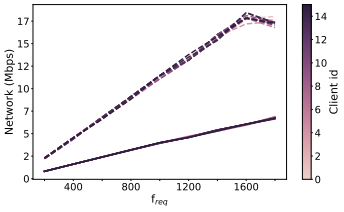


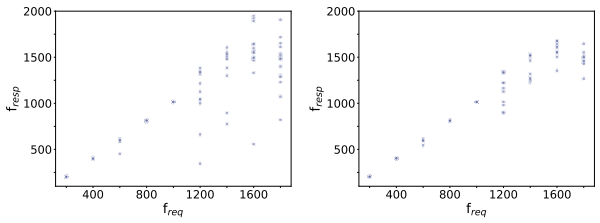
Figure 7: Fabric – Client mean network utilization (outbound traffic as continuous and inbound traffic as dashed lines).

Fabric's architecture, the outbound traffic of follower orderers predominantly consists of sending blocks to the peers. From Figure 5a, and in line with expectations, consistency of inbound and outbound traffic of the followers is observed: followers receive each block once from the leader, and each orderer forwards a received block only once to an anchor peer. On the other hand, when comparing followers' outbound traffic with the leader's outbound traffic (Figure 5b), the ratio consistently stands around four, as the RAFT leader sends each block to each follower (three in our case) and to one peer. Our analysis excludes traffic generated by consensus-related messages, such as appended entries and heartbeat, because it is difficult to distinguish them from block propagation traffic. Nonetheless, the consensus-related messages generate significantly less traffic compared to block dissemination, making the outbound traffic of follower orderers a viable approximation for traffic related to block propagation. Regarding peers (Figure 6), we see that inbound traffic scales linearly with f_{req} for all of them, indicating it is not a bottleneck. Concerning outbound traffic, we observe that it plateaus for some peers while remaining unaffected for others (Figure 6a). Thus, we classify the peers into two main clusters, color-coded as blue and orange. We infer that blue peers act as the gossip leaders of their respective organizations, each with one follower.

The primary distinction in outbound traffic among the two types of peers stems from block propagation between them. To confirm that blue peers are gossip leaders, we deduct the traffic associated with block propagation (as obtained from the ordering service analysis) from the outbound traffic of blue peers (Figure 6b). The resulting traffic is relatively uniform, which is in line with our hypothesis that the blue peers are the gossip leaders. Furthermore, we observe a significant overlap in outbound traffic among peers, particularly at lower request frequencies, with some discrepancies at higher rates. This is expected, as traffic from consensus-related messages increases at higher request rates, making our outbound traffic approximation less accurate. Peer outbound traffic consists of sending requested endorsements and transaction confirmations back to clients. Since the peer outbound traffic plateaus at high f_{req}, it suggests that these components are potential bottlenecks.

We focus on client traffic to pinpoint the specific bottleneck (Figure 7). The inbound traffic of clients, which drops at higher request rates, is generated by the same components as peers' outbound traffic, leaving us with the same potential bottlenecks. Client outbound traffic includes transaction proposals submitted to peers and endorsed transactions sent to the orderers. Since the traffic does not plateau, it suggests that these are not the bottleneck. Overall, the traffic from endorsed transactions sent to the ordering service and blocks sent to peers never plateaus, indicating that the execution and ordering phases are not the bottleneck. Since the endorsements that peers send back to the clients come in between the execution and ordering phases, they are also not the bottleneck. This leaves transaction confirmations from peers to clients, sent after the validation phase, as the likely candidate for a bottleneck. In other words, the validation phase seems to remain the bottleneck even in Fabric v2.0.

4.1.3. Memory & Hard Drive Starting with memory usage, from Figure 1, we observe that all node types exhibit non-plateauing usage levels, suggesting



(a) 1s window (b) 3s window

Figure 8: Fabric – Rolling mean of throughput with different window sizes.

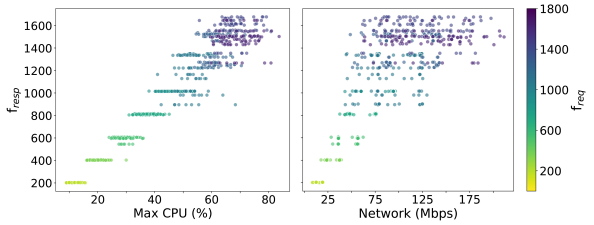


Figure 9: Fabric – Throughput against key resources.

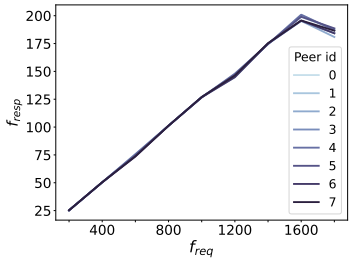


Figure 10: Fabric – Throughput per peer.

that memory constraints are not the bottleneck. For hard drive utilization, only peers' usage appears to plateau. However, peers' I/O operations (such as ledger updates or block processing) occur during or after the validation phase, which does not provide new insights. Additionally, with the peak utilization at approximately 2 %, it is evident that hard drive usage is far from reaching capacity. This indicates that constraints lie within a different component, which in turn limits hard drive utilization. As mentioned in Section 2, the validation phase is comprised of three steps: VSCC, MVCC, and each peer updating its state. As the peers' hard drive utilization is minimal, this leaves only VSCC and MVCC as the potential bottlenecks.

4.2. Fabric: Throughput

We start by gaining an overview of how the request rate affects throughput. This is achieved by plotting the throughput as a rolling mean across two window sizes (Figure 8). Using a one-second window, each data point is plotted individually, revealing significant fluctuations in throughput beyond, f_{req}=1200 s^{-1}, with fluctuations in f_{resp} reaching up to 1000 s^{-1}. We increase the window size to three seconds to observe the overall network performance trend. We selected this interval as it matches the average time it takes for a transaction to be committed to the blockchain under high request rates (see also Guggenberger et al. (2022)). This is significant because queuing effects become prominent at elevated f_{req}, and opting for a shorter time window could underestimate throughput. We see that, on average, the system keeps up with the request rate until reaching approximately f_{req}=1600 s^{-1}.

Next, we explore the correlation between Fabric's throughput and the components and resources highlighted as potential bottlenecks in the first part of the analysis, namely peer CPU utilization and peer network traffic. Figure 9 plots the two resources against the network's throughput using the three-second window. For CPU usage, we observe an initial linear increase with throughput until the characteristic plateau. Additionally, we observe that the instability in CPU utilization starts at around f_{resp}=1200 s^{-1}, which is also the point at which the correlation between f_{req} and f_{resp} starts breaking down (Figure 8a). Consequently, CPU usage correlates more closely with f_{resp} than f_{req}. Similarly, although network traffic increases with throughput, it does so at a lower rate and begins to exhibit instability at f_{resp}=600 s^{-1} already, where the throughput still manages to keep up with f_{req}.

Next, we examine the throughput of individual peers (Figure 10). We observe that every peer contributes similarly to throughput, with minor variations at high f_{req}. Given the significant differences in network traffic between gossip leaders and followers, network traffic is likely not a significant factor in determining throughput. If it were, we would expect noticeable differences in throughput between gossip leaders and followers. Thus, peer CPU utilization appears to be the primary factor behind the leveling off of throughput. The crucial role of peer CPU utilization is in line with our conclusion in Section 4.1 that VSCC and MVCC are the only candidates for bottlenecks in Fabric as both depend on peer CPU. However, given that the checks in MVCC are sequential and Figure 4 shows similar mean core utilization for peers, it is unlikely that MVCC is the bottleneck. This leaves VSCC as the only bottleneck candidate. This hypothesis is further supported by the fact that the validations executed in VSCC are parallelized, and we have noted that the parallelization capacity of Fabric is limited.

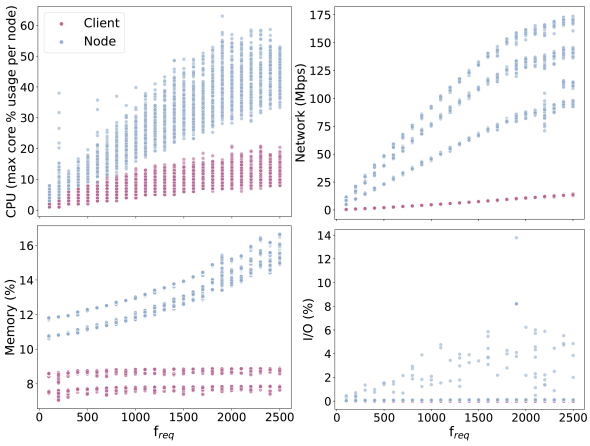


Figure 11: Quorum – Key resource utilization for different request rates f_{req}.

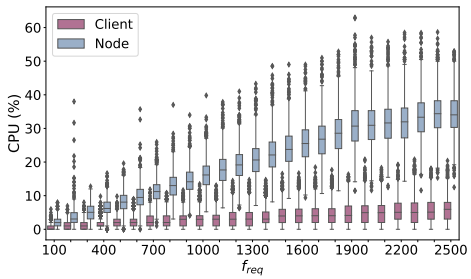


Figure 12: Quorum – CPU utilization of all cores.

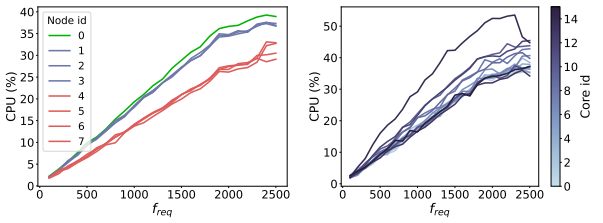
4.3. Quorum: Resource utilization

Our benchmarking experiment for Quorum increases the request rate in increments of 100 requests per second. The Quorum bottleneck analysis based on this series of measurements also starts by gaining an overview of the four resources in relation to f_{req} (Figure 11). Similar to the Fabric case, it is apparent that CPU and network utilization are closely correlated with the request rate. Memory and hard drive usage are limited, with I/O operations exhibiting high fluctuations but generally showing less than 1 % usage. Across all resources, client utilization appears to be minimal, and it is either unaffected or grows linearly with the request rate. Therefore, we again focus on the resource utilization of nodes.

4.3.1. CPU We begin the node CPU analysis by examining the utilization across all cores, where – as for the case of Fabric – we note significant fluctuations among the cores of the nodes as request rates increase (Figure 12). Examining the mean node CPU utilization (Figure 13a), we see that nodes can be grouped into three distinct categories. Node 0 (green) exhibits the highest utilization, indicating it is the RAFT leader, which is responsible for additional operations in consensus, such as transaction ordering and block composition. Nodes 1, 2, and 3 (blue) display slightly lower utilization levels, as their role in consensus is receiving and pre-validating transactions. The remaining nodes (orange) exhibit significantly lower CPU usage as they do not have additional responsibilities beyond appending blocks to their local ledgers and applying them to their state.

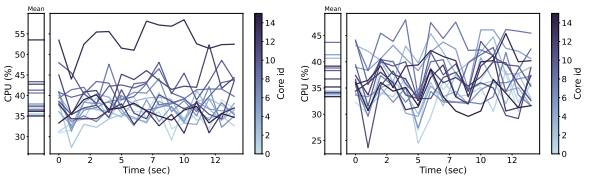
Examining the utilization per core of node 0 (Figure 13b), we observe that one core (core 15) bears a higher workload across all request rates. This is because, except pre-validation, all other tasks of the leader are executed sequentially, leading to a disproportionate strain on one core. Following this observation, we examine the temporal evolution of CPU usage across individual cores at f_{req}=2300 s^{-1} (Figure 14). Here, we see fluctuations by as much as 20 % for the RAFT leader and followers, with disparities of up to 10 % between individual cores when excluding core 15 for node 0. The main difference between the mean utilization of node 0 and nodes 1, 2, and 3 comes only from core 15. These results suggest that parallel processing in Quorum is even more limited than in Fabric, with average core usage not exceeding 40 % and specific leader tasks overburdening one core. The CPU utilization of nodes 0, 1, 2, and 3 reaches a plateau at f_{req}=2400 s^{-1}. Additionally, we notice the first break in the linear relationship between CPU usage and the request rate at f_{req}=1800 s^{-1}. While this does not necessarily indicate a bottleneck, it could provide clues for identifying factors behind the decline in CPU utilization. From Figure 13a, we infer that node 0 and nodes 1, 2, and 3 have similar behavior after reaching a plateau. Therefore, transaction ordering and block building, the main unique operations performed by the leader, are likely not the bottlenecks. Examining Figure 13b, we see a rapid decline in the utilization of core 15 at high request rates. Considering that the remaining sequential operations, such as transaction ordering and block propagation, require minimal CPU resources, this sharp drop cannot be justified, and it appears that another component limits CPU utilization.

4.3.2. Network Focusing on mean network utilization (Figure 15), we can categorize the nodes into three groups. Due to the complexity of Quorum network traffic, we cannot accurately decompose it into individual components, so we rely on the architectural design to identify bottlenecks. Starting with the blue nodes, outbound traffic levels off at f_{req}=2400 s^{-1} while



(a) Mean utilization per Node. (b) Mean utilization of Node 0 per core.

Figure 13: Quorum – Mean CPU utilization.



(a) Node 0. (b) Node 1.

Figure 14: Quorum – CPU utilization for f_{req}=2300 s^{-1}.

signs of plateauing in their inbound traffic appear at f_{req}=1800 s^{-1}. The inbound traffic primarily consists of transactions received either directly from clients or through gossip and blocks from the leader, while outbound traffic originates from the dissemination of pre-validated transactions. This suggests that these operations are the limiting factors at their respective request rates. While we see similar patterns for the orange nodes, the patterns of the leader are essentially the opposite of those of other nodes. The inbound traffic is comprised of all the transactions that are broadcasted to the network and reach the leader through gossip or directly from the clients, and plateaus at f_{req}=2400 s^{-1}. The outbound traffic involves mainly block dissemination to the other nodes and plateaus at f_{req}=1800 s^{-1}. Since their inbound traffic plateaus at f_{req}=2400 s^{-1}, the leader likely keeps receiving the transactions from the blue nodes normally up until that point, leaving us only with block dissemination as the main bottleneck at f_{req}=1800 s^{-1} and transaction propagation as the main issue for f_{req}=2400 s^{-1}.

Considering our CPU utilization findings, we posit that the rapid drop in CPU utilization for core 15 at f_{req}=2400 s^{-1} (Figure 13b) is caused by the leader not receiving enough transactions from the other nodes. At f_{req}=1800 s^{-1}, the decline could be attributed to either the block propagation or one of the processes preceding it, such as transaction pre-validation and adding the block to the chain. Considering that appending the block to a node's local ledger is not CPU intensive,

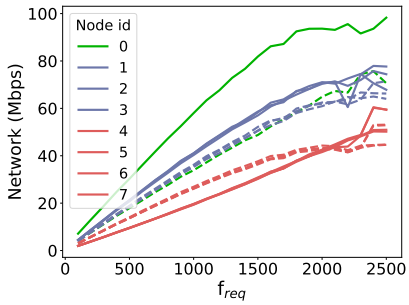


Figure 15: Quorum – Node mean network utilization (outbound traffic as continuous and inbound traffic as dashed lines).

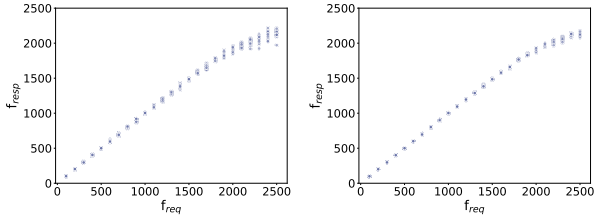
the primary issues likely lie with block propagation or pre-validation. Using the same line of reasoning for f_{req}=2400 s^{-1}, the bottleneck appears to be either transaction propagation or pre-validation, as it is the only operation that precedes propagation.

4.3.3. Memory & Hard Drive Starting with memory utilization, we see consistent behavior across all nodes, with no signs of plateauing (Figure 11). As such, memory does not seem to contribute to performance degradation. Despite observing significant peaks in hard drive utilization (Figure 11), the mean utilization remains mostly below 1 %, indicating it is not a constraining factor. The large fluctuations are probably related to the writing of the block into each node's database, but since it is improbable that it leads to a bottleneck, we do not examine it further.

4.4. Quorum: Throughput

Analyzing the correlation between f_{resp} and f_{req}, we see that even for the one-second window size, throughput remains stable but starts to exhibit higher fluctuations at f_{req}=1800 s^{-1}. Examining the three-second window, throughput plateaus at around f_{resp}=2100 s^{-1}, significantly lower than the maximum request rate. This suggests that the overall performance of the blockchain started to decline before reaching the highest request rate. This indicates that the performance degradation noted at f_{req}=1800 s^{-1} for both CPU and network utilization may be more critical in identifying the bottleneck.

Examining throughput against the CPU and network utilization (Figure 17), we see similar patterns. Both resources keep up with throughput initially and experience higher fluctuations around f_{resp}=1800 s^{-1}. Beyond this point, differences in throughput between request rates become less pronounced (even close



(a) 1s window (b) 3s window

Figure 16: Quorum – Rolling mean of throughput with different window sizes.

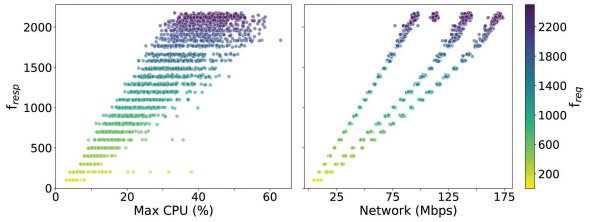
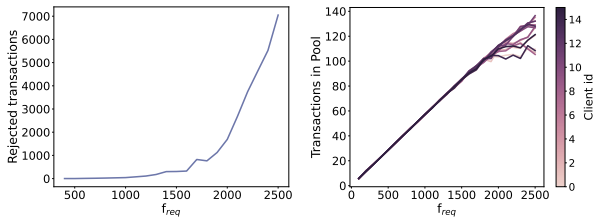


Figure 17: Quorum – Throughput against key resources.

to indistinguishable near f_{resp}=2100 s^{-1}). Given these similar patterns, we search for a potential common source behind the blockchain's performance degradation. Since block and validated transaction propagation are network-related, if they were the bottleneck, they would mainly impact network traffic, and, as a result, they are less likely to be the bottleneck.

This observation leaves only transaction pre-validation as the possible constraining factor, which impacts both CPU and network traffic when nodes are not receiving enough transactions. According to Figure 11, network utilization of clients increases linearly with the request rate, which suggests they send the proper number of transactions to the nodes. Examining further the interaction of nodes with the incoming transactions, we look into the number of rejected transactions (Figure 18a). Rejected transactions are those that nodes decline to propagate, leading to clients receiving nearly immediate (within 50 ms) notifications of transaction failure. Initially, the count of rejected transactions is minimal but begins to surge at f_{req}=1800 s^{-1}, culminating in approximately 7000 rejections by f_{req}=2500 s^{-1}. This corresponds to a rejection rate of 20 %, prompting further investigation into the underlying causes.

In Quorum, transactions are categorized as either executable or non-executable. Executable transactions can be immediately included in a block, while non-executable transactions are out of nonce order and must wait for preceding transactions with lower nonce to execute first. Clients are limited to keeping



(a) Number of rejected transactions. (b) Number of transactions in the pool per client.

Figure 18: Quorum – Transaction metrics.

16 executable and 500 non-executable transactions in the pool at any given time. Figure 18b illustrates that at f_{req}=1800 s^{-1}, the number of transactions in the pool from six clients begins to exhibit instability, diverging from the previous linear relationship with the request rate, and most clients diverge at f_{req}=2400 s^{-1}. This pattern, along with the fact that the number of client transactions in the pool never exceeds 500, suggests that clients are reaching the limit of executable transactions in the pool, leading to more rejections. This hypothesis is further supported by the number of rejected transactions, which increases sharply at f_{req}=1800,s^{-1}, coinciding with the initial reduction in CPU and network utilization and the subsequent plateau at f_{req}=2400,s^{-1}. This indicates that the main bottleneck for Quorum is the clients' limit of executable transactions in the pool.

5. Conclusion

This paper introduces a general illustrative method for blockchain bottleneck identification, demonstrated through an analysis of a 12-node Fabric network and an 8-node Quorum network. Our method leverages EDA to analyze blockchain performance metrics, highlighting their specific characteristics and bottlenecks. By employing a combination of proportional analysis and the study of plateau-shaped trends in resource utilization versus transaction metrics, we uncover performance anomalies. This approach allows us to narrow down the reasons for bottlenecks by comparing the correlation between data trends, the request rates (f_{req}), and response rates (f_{resp}).

For Fabric, we identify the validation phase as the main bottleneck, even for v2.0, with VSCC being the most likely component behind the bottleneck. We were also able to showcase the average moderate degree of parallelization within Fabric, which leaves ample room for improvement. In this sense, our findings align with previous studies (see Section 3). For Quorum, we posit that the bottleneck stems from the restriction

on the number of executable transactions a client can have in the transaction pool, leading to many rejections. Additionally, our findings illustrate Quorum's relatively limited capacity for parallel processing.

Acknowledgements

Funded by the Luxembourg National Research Fund (FNR), grant reference 16326754 and NCER22/IS/16570468/NCER-FT, and by PayPal, PEARL grant reference 13342933/Gilbert Fridgen. To meet grant obligations, the author has applied a Creative Commons Attribution 4.0 International (CC BY 4.0) license to any Author Accepted Manuscript version arising from this submission.

References

Androulaki, Elli et al. (2018). "Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains". In: *Proceedings of the 13th EuroSys Conference*. ACM. DOI: 10.1145/3190508.3190538.

Androulaki, Elli et al. (2019). "Endorsement in Hyperledger Fabric". In: *Proceedings of the International Conference on Blockchain*. IEEE. DOI: 10.1109/Blockchain.2019.00077.

Chacko, Jeeta Ann et al. (2021). "Why Do My Blockchain Transactions Fail? A Study of Hyperledger Fabric". In: *Proceedings of the International Conference on Management of Data*. ACM. DOI: 10.1145/3448016.3452823.

Chatfield, Chris (1986). "Exploratory Data Analysis". In: *European Journal of Operational Research*. DOI: 10.1016/0377-2217(86)90209-2.

Fan, Caixiang et al. (2020). "Performance Evaluation of Blockchain Systems: A Systematic Survey". In: *IEEE Access* 8. DOI: 10.1109/ACCESS.2020.3006078.

Fridgen, Gilbert et al. (2018). "Cross-Organizational Workflow Management Using Blockchain Technology – Towards Applicability, Auditability, and Automation". In: *Proceedings of the 51st Hawaii International Conference on System Sciences*. DOI: 10.24251/hicss.2018.444.

Guggenberger, Tobias et al. (2022). "An In-Depth Investigation of the Performance Characteristics of Hyperledger Fabric". In: *Computers & Industrial Engineering* 173. DOI: 10.1016/j.cie.2022.108716.

Mazzoni, Marco et al. (2021). "Performance Evaluation of Permissioned Blockchains for Financial Applications: The ConsenSys Quorum Case Study". en. In: *Blockchain: Research and Applications*. DOI: 10.1016/j.bcra.2021.100026.

Nakamoto, Satoshi (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. URL: bitcoin.org/bitcoin.pdf.

Ongaro, Diego et al. (2014). "In Search of an Understandable Consensus Algorithm". In: *USENIX Annual Technical Conference*. URL: raft.github.io/raft.pdf.

Queiroz, Maciel M et al. (2020). "Blockchain and Supply Chain Management Integration: A Systematic Review of the Literature". In: *Supply Chain Management: An International Journal* 25. DOI: 10.1108/SCM-03-2018-0143.

Ruan, Pingcheng et al. (2020). "A Transactional Perspective on Execute-Order-Validate Blockchains". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. DOI: 10.1145/3318464.3389693.

Sedlmeir, Johannes et al. (2021). "The DLPS: A New Framework for Benchmarking Blockchains". In: *54th Hawaii International Conference on System Sciences*. URL: hdl.handle.net/10993/45620.

Sedlmeir, Johannes et al. (2022a). "A Serverless Distributed Ledger for Enterprises". In: *55th Hawaii International Conference on System Sciences*. URL: http://hdl.handle.net/10125/80228.

Sedlmeir, Johannes et al. (2022b). "The Transparency Challenge of Blockchain in Organizations". In: *Electronic Markets* 32 (3). DOI: 10.1007/s12525-022-00536-0.

Thakkar, Parth et al. (2018). "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform". In: *Proceedings of the 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*. IEEE. DOI: 10.1109/MASCOTS.2018.00034.

Toufaily, Elissar et al. (2021). "A Framework of Blockchain Technology Adoption: An Investigation of Challenges and Expected Value". In: *Information & Management* 58.3. DOI: 10.1016/j.im.2021.103444.

Wang, Canhui et al. (2020). "Performance Characterization and Bottleneck Analysis of Hyperledger Fabric". en. In: *Proceedings of the 40th International Conference on Distributed Computing Systems*. DOI: 10.1109/ICDCS47774.2020.00165.