ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

TOWARDS EFFICIENT AND SAFE BLOCKCHAIN

MASTER SEMESTER PROJECT

Yanick Paulo-Amaro

Supervised by Gauthier Voron and Prof. Rachid Guerraoui

15th June 2022

Abstract

Blockchain is an innovative technology with promising uses in finance, decentralized computing and market platforms. These applications require high performance to be able to reach a truly global scale. Unfortunately, current blockchains do not reach the tens of thousands of transactions per second necessary for this goal. This project explores bottlenecks in blockchains by building a simple smart-contract system using the same components as Diem, a state of the art blockchain. We show that consensus is the best performing component of blockchain systems and that cryptography and virtual machines are bottlenecks. We also show that our simplistic system is able to sustain much higher throughput that Diem while using the same components, indicating that the architecture of blockchains have a large impact on performance, independently of the components used.

1 Introduction

1.1 Context

Nowadays, blockchain is a pervasive technology, with promising applications in finance (with cryptocurrencies), market platforms (with non fungible tokens), and decentralized computing (with smart contracts). To serve the increasing demand, blockchain systems must rely on efficient and safe building blocks, such as BFT consensus. State of the art consensus protocols such as HotStuff [1] now serve tens of thousands of requests per second. Surprisingly, blockchain systems based on these protocols never reach such performance, with an end to end throughput lower by two orders of magnitude. This gap in performance is a major limitation for the large scale use of such blockchain systems.

1.2 Goals

This project investigates the reason for this performance discrepancy. We build a minimal smart-contract system based on HotStuff, Diem's MoveVM [2] and ED25519 digital signatures. We evaluate each component of our system: standalone consensus, consensus with cryptographic signatures and full smart-contract based cryptocurrency and compare them to HotStuff and Diem [3].

1.3 Claims

In this report, we show that the core component of blockchain systems, consensus, is able to achieve very high throughputs on clusters of up to 48 nodes. With this performance as baseline, we show that the verification of signatures is an important bottleneck when running on machines with fewer cores. Despite this, we show that a simple system with consensus and cryptography can achieve decent performance given enough parallelism. We will then show that the smart-contract virtual machine is also a bottleneck compared to consensus and that adding a VM on top of cryptography reduces performance further. We will also show that its standalone performance is higher than cryptography, indicating that the two bottlenecks interfere with each-other. We also compare our simplistic system with Diem and show that we achieve one order of magnitude higher throughput despite using similar components. Finally, we show that DiemVM performs much worse than MoveVM which it is built on top of. As a side result, we find that contrary to what we expected, HotStuff performs better when deployed across multiple regions compared to only one. We also find that for clusters in a single region, using AWS EC2 (faster) private IPs results in slower and more erratic performance compared to using (slower) public IPs.

1.4 Outline

In the following sections, we describe our simple smart-contract system and how we measured its performance. We then discuss the results and compare them to Diem and HotStuff. We briefly discuss the performance of the Diem VM itself, comparing it to the Move VM. We will discuss strange behaviors of HotStuff when using high bandwidth connection and when running on multiple datacenters compared to only one. Finally we conclude by summarizing our findings.

2 Implementation

We built our system in Rust on top of Alberto Sonnino's implementation of HotStuff [4]. Clients send their signed transactions to the consensus nodes. The transactions contain the sender's address, the sender's public key, a nonce, a transaction script, arguments for the script and an ED25519 signature. Once consensus is reached on a block of transactions, it is sent to a signature verifier thread which checks the signatures in parallel and filters out invalid transactions. Finally, the resulting block is sent to an executor thread which checks the nonce of the transactions before executing them one by one on the MoveVM. Since transactions can be reordered when they appear in different consensus blocks, the executor thread uses a backlog to ensure first-in first-out execution for each client. Each thread is connected to the next one by a buffered channel. We make sure to record whenever the channel is full, signaling that a component is waiting on the next. This is helpful to know if a component is a bottleneck in the system.

This first configuration, dubbed HotMove, is a full smart-contract system using all three components: HotStuff consensus, ED25519 cryptographic signatures and Move virtual machine.

In a second configuration, dubbed HotCrypto, the executor thread does not use the VM and execute transactions directly. This results in a very basic cryptocurrency system using only consensus and signatures. Note that in this configuration the only kind of transaction is a transfer between two accounts. This was done to keep our system simple. Other types of transactions could easily be added by hardcoding them in the system.

Finally, in our third configuration the verifier thread does not check the signature and simply discard all transactions. This lets us run only HotStuff consensus, without any cryptography or virtual machine.

2.1 Limitations

Unlike Diem, our system does not have a mechanism for charging clients for executing their transactions and only executes transactions sequentially. A real system like Diem would optimistically execute transactions in parallel and default to sequential execution only in case of conflicts. We chose to execute them sequentially because we use very few clients for our benchmarks. This means executing transactions concurrently would often lead to conflicts and force us into the sequential case anyway. In the HotMove configuration, the VM is setup to use in-memory storage instead of putting data to disk which should present an idealized scenario for our tests.

2.2 Difficulties

One of the main difficulties in implementing this system was due to our lack of experience in Rust. For consistency, we wanted to use the same code to benchmark all configurations as well as the Move and Diem VMs. For the distributed systems, we could just adapt Alberto Sonnino's benchmarking client to send the correct type of transactions and augment the nodes so that they process the committed transactions. For the VMs however, since we do not use consensus, we replace the nodes by a local thread dedicated to VM execution on the client itself. The problem was that the wrapper around DiemVM does not satisfy

the required trait bounds to be sent across threads. This means that it cannot survive an **await** when running on a worker thread and prevented us from reusing our benchmark code. Since it *does work* when running on the main thread, it took us quite some time to understand this error. We managed to avoid it by swapping the role of the main and worker threads for the DiemVM benchmark.

Another difficulty was understanding compilation for DiemVM and MoveVM. The Diem and Move repositories use different versions of the Move language, with different syntaxes and multiple ways of compiling Move scripts and modules and add them to their respective VMs. This was quite confusing, especially since we imported both of them and they are not compatible with each other when imported this way.

Finally, the last difficulty was related to the benchmarking framework. We had to adapt the benchmark from HotStuff to be able to run and parse logs for the other systems automatically. We also had issues where longer benchmarks would crash midway because too many connections were kept open, which was easy to fix once we figured out the cause.

2.3 Availability

Our system and the components it is based on are available here:

- Our system: [5]
- HotStuff: [4]
- Diem: [3]
- Move language: [2]

3 Evaluation

3.1 Setup

We adapted Alberto Sonnino's benchmarking setup to run our different configurations. Detailed instructions on how to install, run benchmarks, plot and interpret results can be found on our repo [5]. What follows is a short description of the benchmarking setup.

We benchmark the performance of each component of our implementation at different input rates and cluster sizes. We deployed our testbed on AWS EC2 using c5-xlarge instances with 4 vCPU cores and 8 GiB of memory. Each instance runs a single node as well as a client which generates transactions at a fixed rate. Each client sends an equal share of the total input rate to the node on the same instance. All instances are deployed in the eu-west-3 region and use private 10 Gbps IPs to communicate. We did not add failed nodes. When benchmarking DiemVM and MoveVM on their own, we only deploy a single instance running both the client and the VM.

To run tests yourself, first install our system by following the instructions on github [5]. You can then run tests locally with `fab local --mode=<MODE>` where `<MODE>` is:

- **hotstuff** - Consensus standalone
- **hotcrypto** - Consensus + cryptography
- **hotmove** - Consensus + cryptography + MoveVM
- **movevm** - MoveVM standalone
- **diemvm** - DiemVM standalone

To run benchmarks on a cluster, you will need to use the commands `fab create --nodes=<NODES>`, `fab install` and `fab remote --mode=<MODE>`. Once you have some results, you can plot them using `fab plot --mode=<MODE>`. The will produce three graphs: a robustness graph, a latency-throughput graph and a tps graph. In both cases, the dots represent the mean of the measurements and the bars represent one standard deviation.

3.2 Distributed Systems

Consensus

To evaluate whether HotStuff can reach throughputs of tens of thousands of transactions per second, we measured its performance with 6, 12, 24 and 48 nodes and input rates ranging from 25'000 to 300'000 transactions per seconds. Since HotStuff does not use the content of the transactions, we simply used arrays of 197 bytes. We chose 197 to match the size that HotCrypto requires for its transactions. The benchmark was run three times per configuration and lasted 120 seconds per run. We expect HotStuff to achieve throughputs of hundreds of thousands of transactions per second as this is what was demonstrated in the original paper [1].

As we can see from figure 1.1, HotStuff is able to sustain a throughput of 200'000 tx/s with up to 48 nodes. We also see that past this rate, the variance of our results increases drastically. For example, with 24 nodes with 275'000 tx/s as input, we obtained throughputs of 274'443 tx/s, 49'759 tx/s and 218'224 tx/s. Even worse, with 12 nodes and an input rate of 300'000 tx/s, we got 299'284 tx/s, 118'500 tx/s and 92'379 tx/s. We believe this might be caused by the use of private IPs, as we will discuss in a later section. This confirms that consensus is really fast and is already a good starting point for building large scale blockchains.

Consensus + Crypto

To evaluate whether the verification of signatures is a bottleneck in blockchain systems, we measured the performance of HotCrypto, the configuration of our system with consensus and cryptographic signatures. We used the same setup as for HotStuff, same number of nodes, same duration and same number of runs. The only difference is that we used proper transactions that include sender address, sender public key, destination address, transfer amount, nonce and signature. A serialized signed transaction is 197 bytes long. We expect the performance of HotCrypto to be much lower than HotStuff as signature verification is generally slow and we use instances with only 4 cores, limiting the advantage of doing verification in parallel.

Figure 1.2 shows our results with input rates from 10'000 to 80'000 tx/s. As we can see, HotCrypto achieves a maximum throughput of almost 50'000 tx/s with 6 nodes. This throughput goes down to 40'000, 30'000 and 20'000 tx/s as we increase the number of nodes to 12, 24 and 48 respectively. This dependency on the number of nodes is due to the signature verifier receiving batches of different sizes in each configuration. As we increase the number of nodes, the rate of each individual client goes down, causing its node to create smaller batches. As we can see in figure 1.11, with an input rate of 70'000 tx/s the average batch size with 6 nodes is 1201 transactions whereas with 48 nodes the average is 146. In the end, this reduction in batch size increases the overhead of running signature verification in parallel, leading to lower throughput. These results show that the addition of cryptography drastically reduces the maximum throughput compared to HotSuff and confirms that signatures are a major bottleneck in blockchain systems using commodity hardware. As we will see later, increasing the number of available cores can give a non-negligible boost in performance, although it is not enough to remove the bottleneck.

CONSENSUS + CRYPTO + VM

To show that a smart-contract virtual machine is a bottleneck in blockchain systems, we evaluate the performance of HotMove with the same setup as HotStuff and HotCrypto. The only difference with HotCrypto is that the transactions contain a compiled Move script in addition to the other fields. The script executes a transfer of `BasicCoin` from the Move tutorial [6]. The addition of the script brings the size of serialized signed transactions to 286 bytes. We expect the performance to be at most the performance of HotCrypto as we are adding a new component on top of it.

Figure 1.3 shows our results from 10'000 to 80'000 tx/s. As we can see, HotMove reaches 30-35'000 tx/s with 6 to 24 nodes and 15-20'000 tx/s with 48 nodes. For small number of nodes, this is fairly close to HotCrypto's 30-50'000 tx/s and for 48 nodes this is well within margin of error. This shows that adding a virtual machine on top of consensus and signatures reduces performance slightly. This performance reduction is most likely caused by the signature verifier since it is very dependent on the available parallelism, as we will see in the next section. As the VM is running alongside the verifier, it is taking away some parallelism and could be reducing the performance of the verifier. Therefore we cannot conclude that the VM is a bottleneck just yet. To find out whether it is, we will have to measure its standalone performance, which we do in the next section.

DIEM

To evaluate how Diem compares to our simplistic system, we measured its performance using a similar setup as for HotMove. We deployed testnets of 6 and 48 nodes in a single region and using private IPs with clients sending transaction at a fixed rate. Because Diem has a limit on the number of transactions in mempool per account, we had to use more accounts than for HotMove. We expect Diem to reach no more than 1000 transactions per seconds as it is the throughput it advertised.

Figure 1.4 shows our results with 6 and 48 nodes, both using 1000 accounts in the VM. As we can see, Diem is able to handle an input rate of 1000 transactions per second with no issue. However, with an input rate of 1500 tx/s its performance collapses, reaching only around 200 tx/s with 6 nodes and around 300 tx/s with 48 nodes. It is possible that Diem was fine tuned for this specific throughput and is not meant to handle more than 1000 tx/s, resulting in this hard ceiling. This shows that even when using the same components, two blockchains can have very different performance depending on their architecture.

3.3 CPU SCALING AND VM PERFORMANCE

CONSENSUS + CRYPTO

Since signatures can be verified in parallel, we believe that a system with consensus and cryptography could achieve decent performance given enough parallelism. To test this hypothesis, we evaluated HotCrypto's performance on clusters of four xlarge, 2xlarge, 4xlarge and 9xlarge instances with 4, 8, 16 and 36 vCPU cores respectively.

As we can see in figure 1.5, HotCrypto achieves 40 to 50'000 tx/s on 4 nodes with 4 cores. This matches what we found for 6 nodes in the previous section. Increasing the number of cores to 8 almost doubles the performance of HotCrypto which achieves around 70'000 tx/s on 4 nodes. Increasing the number of cores further yields a much lower improvement in performance with 16 cores achieving 80'000 tx/s and 36 cores reaching almost 90'000 tx/s. This confirms that given enough parallelism, a system with consensus and cryptography can achieve throughputs close to 100'000 tx/s with a small number of nodes. This result also implies that attempting to improve blockchain performance by running the VM in parallel will have adverse effects as it would take resources away from signature verification and would likely reduce throughput.

MoveVm

To see if a smart-contract virtual machine is a bottleneck compared to consensus, we evaluate the performance of the Move virtual machine in isolation. The setup is similar to what we used for the distributed systems but on a single machine. We use a client and an executor thread on a single machine, either an xlarge, 2xlarge or 9xlarge instance, with 4, 8 and 16 cores respectively. The client thread creates simple transactions containing source address, destination address, amount and the same script as for HotMove but no signature. The transactions exchange balance back and forth between two accounts already registered in the VM. The executor is responsible for running the VM and executing the transactions one by one. Overall, the setup is just like in HotMove but without consensus and signature verification. We expect the VM to perform very well regardless of the number of cores since the transactions are executed sequentially.

Figure 1.6 shows our results from 10'000 to 100'000 transactions per second. As we can see, the MoveVM is able to achieve a maximum throughput of 80'000 tx/s whether there are 4, 8, or 16 cores available. This shows that the MoveVM is a bottleneck compared to HotStuff but that it is able to achieve decent performance when compared to HotCrypto. The fact that this limit of 80'000 tx/s matches what we found for HotCrypto with 16 cores indicates that a system combining consensus, cryptography and the MoveVM can achieve decent performance given enough parallelism.

DiemVM

To find out whether Diem's lower performance is caused by its VM, we evaluated the Diem virtual machine using the same setup as for the MoveVM. The client thread produces batches of transactions at a fixed rate while the second thread executes them on the DiemVM. We disabled signatures in DiemVM and forced the transactions to be executed one by one. This makes for a better comparison with the MoveVM as it doesn't have signatures nor optimistic execution of transactions. We expect DiemVM to be slower than MoveVM since it has more work to do outside of transaction execution (e.g. running a prologue and an epilogue before and after each transaction, deducting gas, etc).

As we can see in figure 1.7, the DiemVM is able to reach 8000 transactions per seconds with extremely low latency. When the load goes past 10'000 tx/s, the latency increases quickly and performance hits a ceiling of about 10'000 transactions per second. This shows that DiemVM is much slower than the MoveVM. This is probably because DiemVm has to execute a prologue and epilogue for each transaction. Further work is need to see exactly where the DiemVM spends this additional time.

3.4 Multiple regions

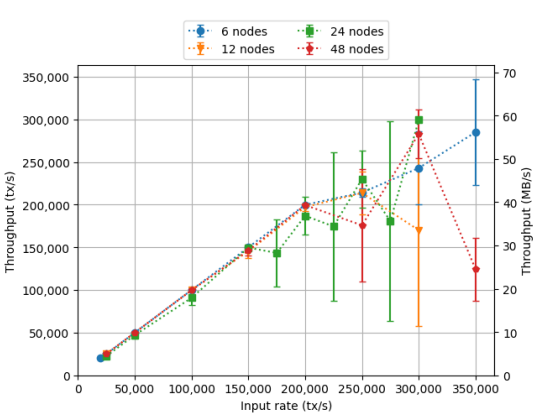
In order to get more real world results, we wanted to evaluate HotStuff running across three regions: Paris (eu-west-3), Tokio (ap-northeast-1) and Ohio (us-east-2). Since the nodes were in different datacenter, we used the public IPs of the instances for communication across nodes. We were expecting the results to be lower than what we measured in a single region since communication between nodes suffer a much larger latency. However, as you can see by comparing figures 1.8 and 1.9, we found that running HotStuff on multiple regions was actually faster than in a single region! Our hypothesis is that having nodes so close to each-other causes them to generate much more traffic than normal. This could mean that nodes create more work for each others, leading to slightly lower performance and more noise. While we were unable to verify this hypothesis directly, we did run the same test using private IPs. This should exacerbate the issue since they have more bandwidth than public IPs. Indeed, as you can see in figure 1.10, we found that with the private IPs the performance was lower and the variance slightly higher. Given these results, we decided to focus on single region tests for the time being. It could be interesting to investigate this unexpected behavior in future work.

4 Conclusion

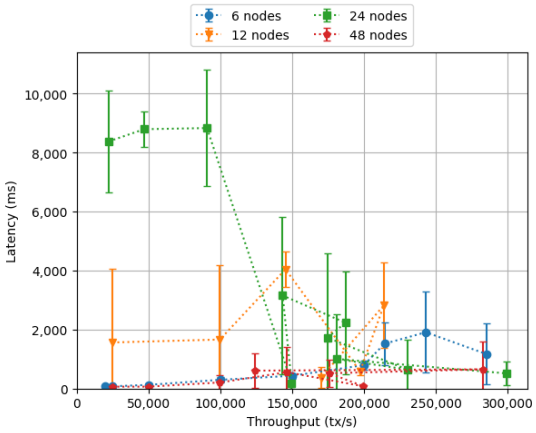
In this report, we have confirmed that state of the art consensus can achieve hundreds of thousands of transactions per seconds in test environments. We have built a simple smart-contract blockchain system called HotMove based on HotStuff, ED25519 signatures and Diem's MoveVM. Using our system, we have shown that the verification of cryptographic signatures drastically slows down blockchain systems but that increasing parallelism can alleviate some of this bottleneck. We have also shown that modern smart-contract virtual machines like MoveVM are also bottlenecks compared to consensus but that they can achieve decent performance when evaluated on their own. We have shown that cryptography and virtual machine interfere with each-other, indicating that simply running the VM in parallel will not improve the overall performance of a blockchain. We compared our system to Diem and shown that despite using the same building blocks, our simplistic system performs much better. We compared DiemVM and MoveVM and shown that the former is quite a bit slower than the latter. These last two results suggest that even when using state of the art implementations of consensus, cryptography and virtual machines, the performance of modern blockchain can vary significantly based on their design alone.

5 Acknowledgements

We would like to thank Prof. Rachid Guerraoui for his supervision and for giving us the opportunity to work on this project. We would like to thank Gauthier Voron for his guidance and support throughout the semester and for providing the Diem results.

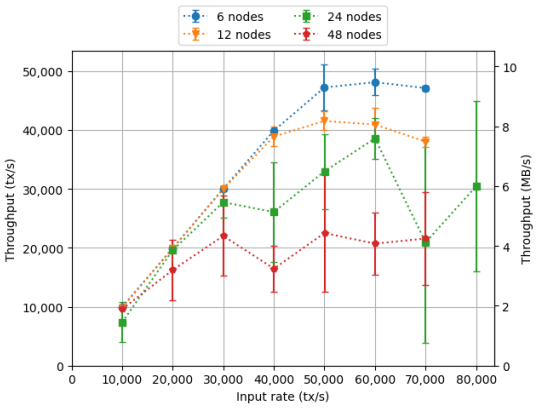


(A) Robustness

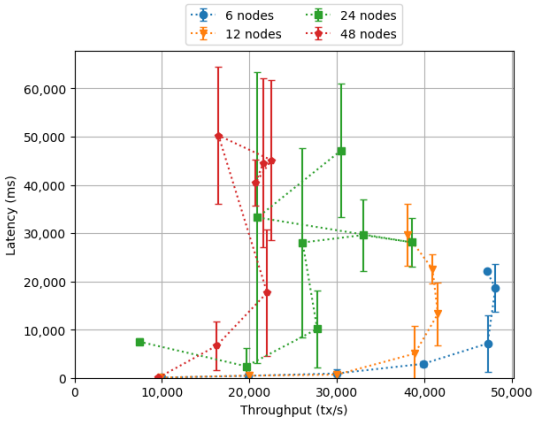


(B) Latency-throughput graph

FIGURE 1.1
Standalone consensus ↑

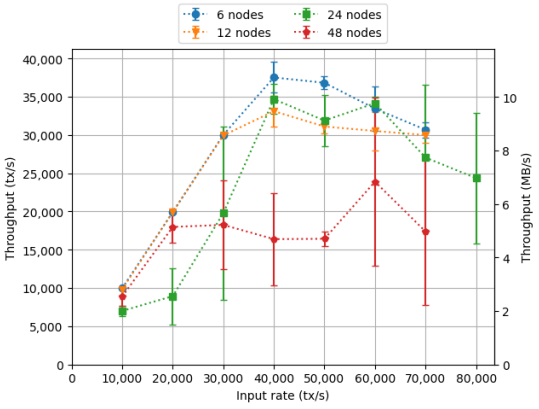


(A) Robustness

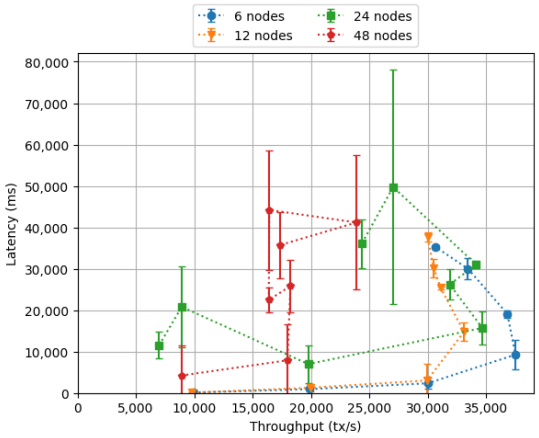


(B) Latency-throughput graph

FIGURE 1.2
Consensus + Crypto ↑

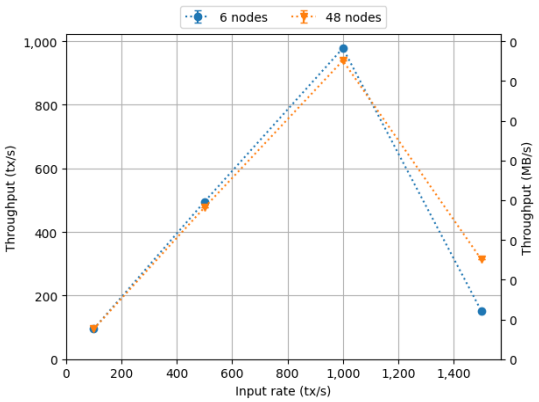


(A) Robustness

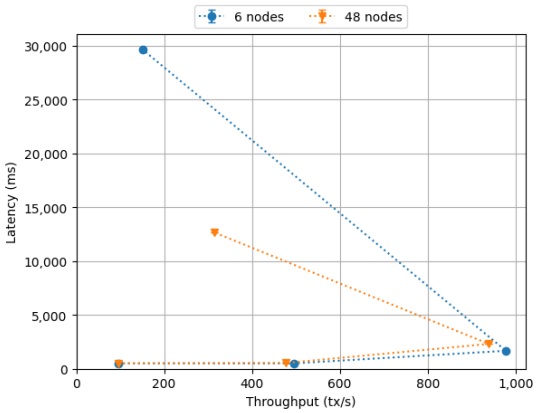


(B) Latency-throughput graph

FIGURE 1.3
Consensus + Crypto + MoveVM ↑

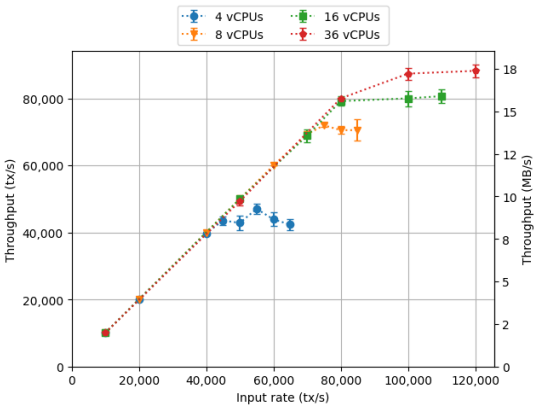


(A) Robustness

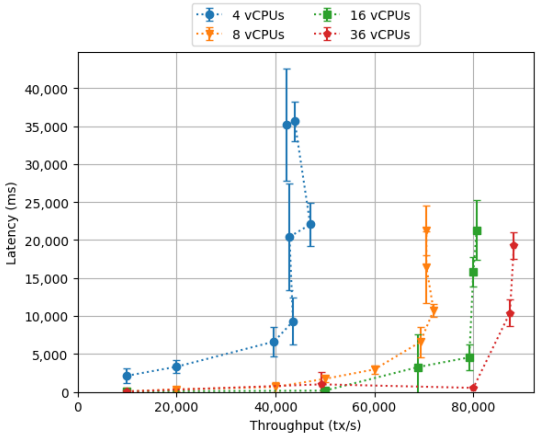


(B) Latency-throughput graph

FIGURE 1.4
Diem, 1000 accounts ↑

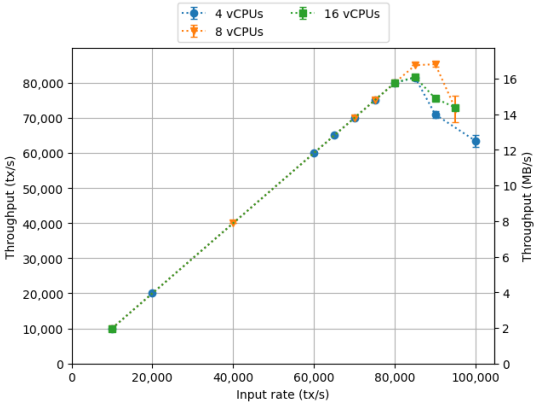


(A) Robustness

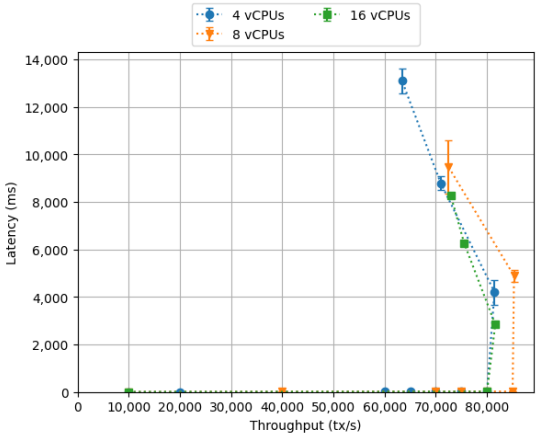


(B) Latency-throughput graph

FIGURE 1.5
CPU scaling: Consensus + Crypto ↑

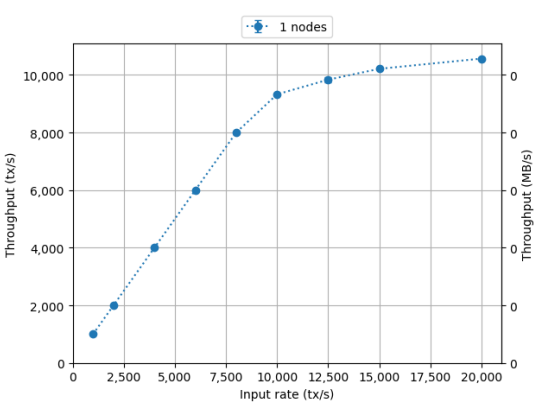


(A) Robustness

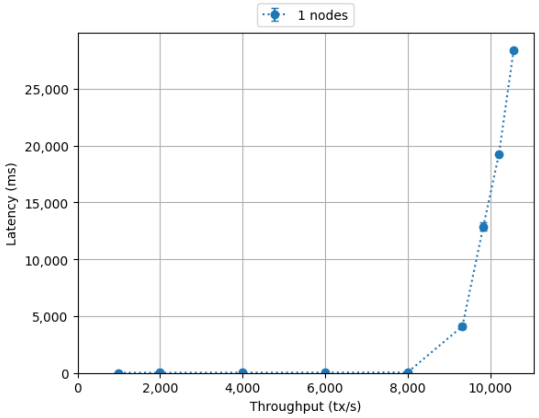


(B) Latency-throughput graph

FIGURE 1.6
CPU scaling: MoveVM ↑

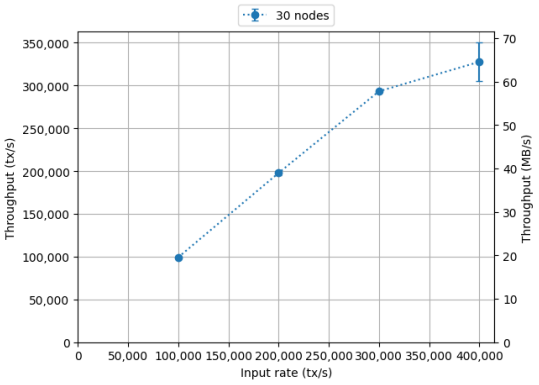


(A) Robustness

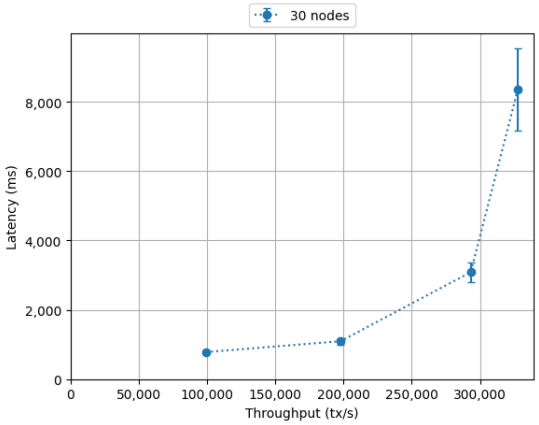


(B) Latency-throughput graph

FIGURE 1.7
DiemVM ↑

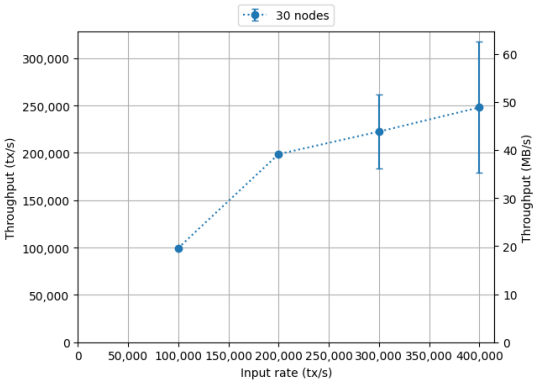


(A) Robustness

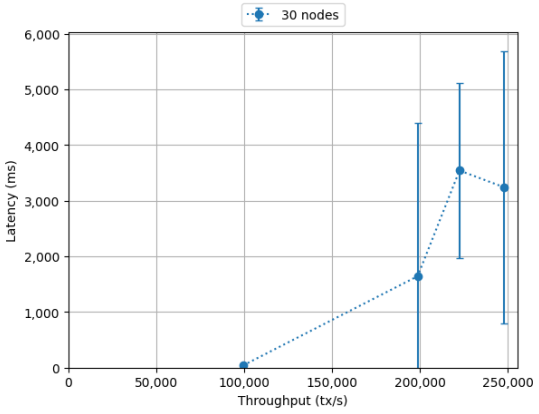


(B) Latency-throughput graph

FIGURE 1.8
Consensus: multiple regions (Paris, Tokio, Ohio) ↑

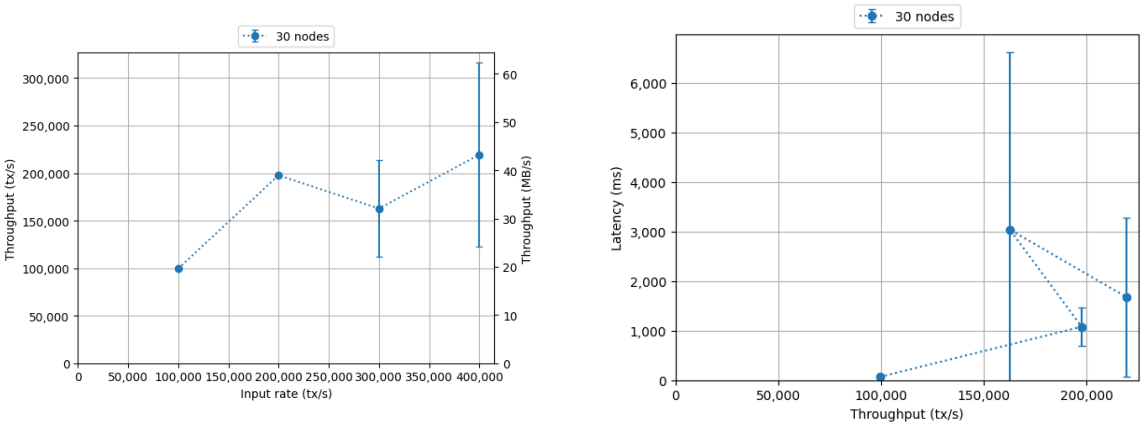


(A) Robustness



(B) Latency-throughput graph

FIGURE 1.9
Consensus: single region (public IPs) ↑



(A) Robustness

(B) Latency-throughput graph

FIGURE 1.10
Consensus: single region (private IPs) ↑

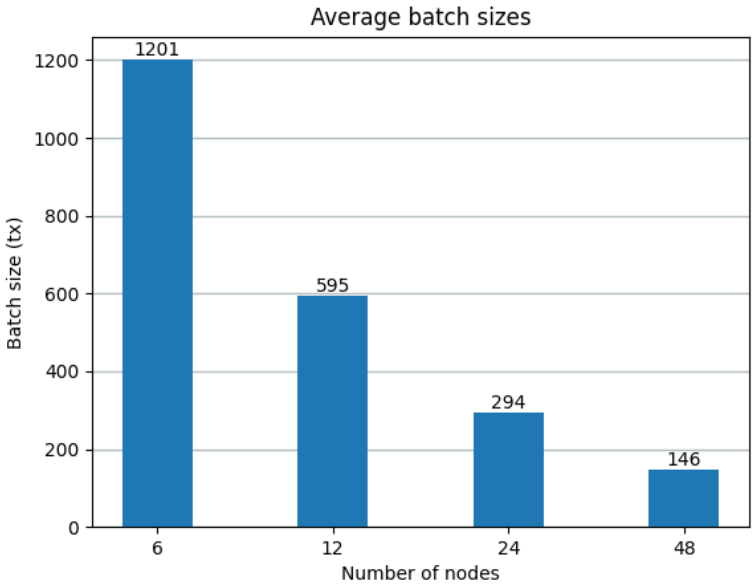


FIGURE 1.11
Consensus + Crypto: batch sizes at 70'000 tx/s ↑

BIBLIOGRAPHY

[1] Ittai Abraham, Guy Gueta and Dahlia Malkhi. 'Hot-Stuff the Linear, Optimal-Resilience, One-Message BFT Devil'. In: *CoRR* abs/1803.05069 (2018). arXiv: 1803.05069. URL: http://arxiv.org/abs/1803.05069.

[2] *Home of the Move programming language*. Diem Foundation. URL: https://github.com/diem/move (visited on 8th June 2022).

[3] *Diem github repository*. Diem Foundation. URL: https://github.com/diem/diem (visited on 8th June 2022).

[4] Alberto Sonnino. *Implementation of the HotStuff consensus protocol*. URL: https://github.com/asonnino/hotstuff (visited on 8th June 2022).

[5] Yanick Paulo-Amaro. *Simple smart-contract system to identify bottlenecks in blockchains*. URL: https://github.com/yanickpauloamaro/hotcrypto (visited on 8th June 2022).

[6] *Move tutorial: BasicCoin step 4*. Diem Foundation. URL: https://github.com/diem/move/blob/main/language/documentation/tutorial/step_4_sol/BasicCoin/sources/BasicCoin.move (visited on 8th June 2022).