The Red Belly Blockchain Experiments

Concurrent Systems Research Group, University of Sydney, Data61-CSIRO

Abstract

In this paper, we present the largest experiment of a blockchain system to date. To achieve scalability across 1000 servers in more than 10 countries located on four different continents, we drastically revisited Byzantine fault tolerant blockchains and verification of signatures.

The resulting blockchain, called the *Red Belly Blockchain (RBBC)*, commits more than a hundred thousand UTXO transactions issued by permissionless nodes, that are grouped into blocks within few seconds through a partially synchronous consensus run by permissioned nodes. It prevents double spending by guaranteeing that a unique block is decided at any given index of the chain in a deterministic way.

We evaluated the performance of RBBC on up to 1000 machines across 4 continents. In the same geo-distributed environment with low-end machines, we noticed two interesting comparisons: *(i)* the RBBC throughput scales to 220 nodes whereas the classic 3-step leader-based BFT state machine used by consortium blockchains cannot scale to 40 nodes; *(ii)* RBBC also achieves a latency of 3 seconds on 100 nodes where HoneyBadgerBFT relying on a constant expected time randomized consensus achieves a latency of more than 10 seconds.

1 Experimental Evaluation

In this section, we evaluate RBBC on up to 1000 machines on Amazon EC2 located in up to 14 separate regions. To this end, we compare the performance of (1) RBBC with its sharding and its DBFT consensus. (2) RBBC where we replaced DBFT by the Honey Badger of BFT protocol (HBBFT) [7], for which we reused the publicly available cryptographic operations implementation and (3) RBBC where we replaced DBFT by a classic 3-step leader-based BFT algorithm *CONS1* [2–4].

We run three types of experiments: *(i)* with up to 300 deciders all deciding and generating the workload, allowing new proposals to be made as soon as the previous one is committed (§1.1 and §1.3); *(ii)* with requesters running on nodes separated from the permissioned nodes to measure their impact on performance and finally (§1.4); *(iii)* with up to 1000 nodes all runnings as replicas, some requesting and some deciding, but all updating their copy of all account balances (§1.5).

Leader-based (CONS1) and randomized BFT (HBBFT). CONS1 is the classic 3-step leader-based Byzantine consensus implementation similar to PBFT [3], the Tendermint consensus [4], and including the concurrency optimizations of BFT-Smart [2]. To reduce network consumption CONS1 is implemented using digests in messages that follow the initial broadcast. Both CONS1 and HBBFT variants make use of a classic verification, as in traditional blockchain systems [9, 10], that takes place at every decider upon delivery of the decided block from consensus. Unless otherwise stated, all nodes behave correctly. Apart from the sharded verification of RBBC, all algorithms run the same code for the state-machine component implementing the blockchain. Note that there exist BFT algorithms that terminate in less message steps than CONS1, but require additional assumptions like non-faulty clients [1, 5] or $t < n/5$ [6]. HBBFT uses a randomized consensus [8] and reliable broadcast using erasure codes.

Machine specification. We run the blockchains on the 14 Amazon datacenters that we had at our disposal at the time of the experiment: North Virginia, Ohio, North California, Oregon, Canada, Ireland, Frankfurt, London, Tokyo, Seoul, Singapore, Sydney, Mumbai, São Paulo. We tested two different VMs: (1) *high-end* c4.8xlarge instances with an Intel Xeon E5-2666 v3 processor of 18 hyperthreaded cores, 60 GiB RAM and 10 Gbps network performance when run in the same datacenter where storage is backed by Amazon's Elastic Block Store (EBS) with 4 Gbps dedicated throughput; (2) *low-end* c4.xlarge instances with an Intel Xeon E5-2666 v3 processor of 4 vCPUs, 7.5 GiB RAM, and "moderate" network performance (as defined by Amazon). Storage is backed by EBS with 750 Mbps dedicated throughput. To limit the bottleneck effect on the leader of PBFT, we always place the leader in the most central (w.r.t. latency) region, Oregon. When not specified, proposals contain 10,000 transactions and t is set to the larger integer strictly lower than $\frac{n}{3}$.

1.1 Comparing geodistributed blockchains

First, we report the performance when running 10 high-end VMs in each of the 14 regions for a total of 140 machines. Each region contains 10 high-end machines. As depicted on Table 1, we computed the variation of communication latencies and throughput between these Amazon EC2 datacenters as measured using c4.xlarge instances. The minimum latency is 11 ms between London and Ireland, whereas the maximum latency is 332 ms observed between Sydney and São Paulo. Bandwidth between Ohio and Singapore is measured at approximately 64.9 Mbits/s (with variance between 6.5 Mbits/s and 20.4 Mbits/s).

1.1.1 Impact of verification

To measure the impact of verification on performance, we varied the parameter t from the minimum to its maximum value ($46 < \frac{140}{3}$) with sharded verification as depicted in Figure 1 (left) and we compared all three blockchains with

	Tokyo	Seoul	Mumbai	Singa.	Sydney	Canada	Frankfurt	Ireland	London	São P.	N.Virg.	Ohio	N.Cal.	Oregon
Tokyo	0	551	129	240	161	106	74	66.4	59	55.4	90.1	96.2	129	132
Seoul	33	0	137	157	141	91.5	54	60.8	54.7	56.6	84.2	114	84.2	116
Mumbai	133	164	0	121	67	90.9	176	178	145	46.7	81.9	80.5	69.1	64.2
Singapore	69	100	67	0	90.9	83.2	90.7	86.1	90.4	40.8	59.5	64.9	80.5	77.3
Sydney	106	135	235	170	0	77.1	61.3	53.8	51.2	40.2	74.9	99.7	135	119
Canada	166	185	196	220	225	0	166	250	164	159	808	760	205	168
Frankfurt	244	275	112	178	292	102	0	477	823	92.9	222	220	144	85.7
Ireland	226	246	122	188	286	78	25	0	829	114	185	183	104	117
London	255	284	111	179	281	90	15	12	0	107	190	195	107	85.5
São Paulo	271	293	302	328	332	125	210	184	192	0	131	124	77.7	81.7
N. Virginia	162	209	182	238	205	15	89	85	76	122	0	827	232	186
Ohio	169	199	193	227	196	25	99	91	87	131	13	0	428	219
N. California	120	150	262	178	148	76	148	142	138	182	64	52	0	681
Oregon	105	135	235	163	162	66	164	141	158	183	76	71	22	0

Table 1. Heatmap of the bandwidth (Mbps) in the top right triangle and latency (ms) in the bottom left triangle between the 14 regions of Amazon Web Services, as used in our experiments

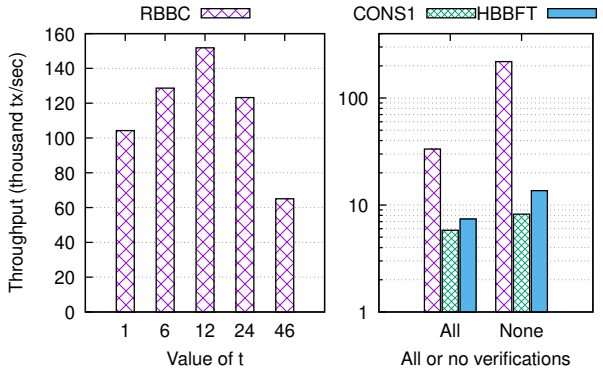

Figure 1. Impact of fault tolerance and verification on the RBBC throughput when $n = 140$ geodistributed machines

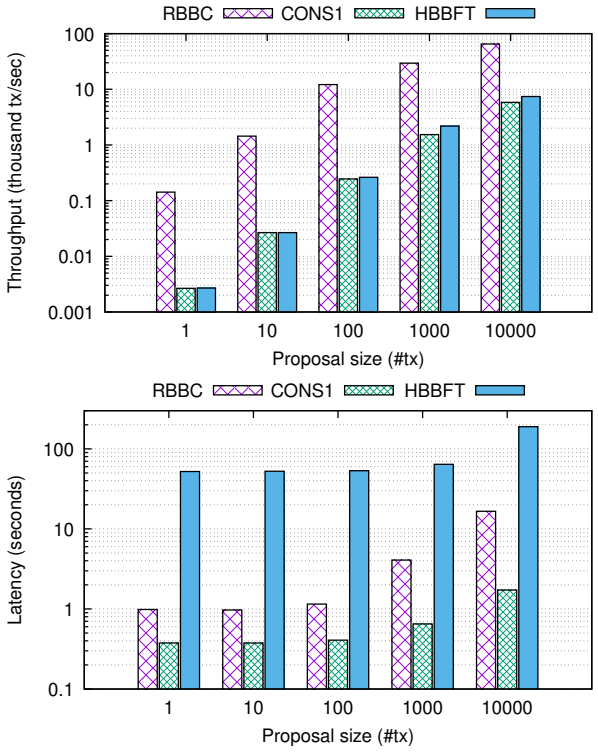

Figure 2. Throughput and latency comparison of the blockchain solutions with $n = 140$ and $t = 46$, and proposal sizes of 1, 10, 100, 1000 and 10000

all nodes verifying all transactions (all) and with no verification (no validation) as depicted in Figure 1 (right). The peak throughput of 151,000 *transactions per second* (tx/sec) is achieved with the fault-tolerance parameter $t = 12$. When $t \leq 6$, performance is limited by the $(t-1)^{th}$ slowest node as the consensus waits for a higher number of $n - t$ proposers. When $t \geq 24$, performance is then limited by the growing number of $t + 1$ necessary verifications. In Figure 1 (right), the performance of all algorithms is higher without verification than with full verification. RBBC is the most affected dropping from 219,000 tx/sec to 33,000 tx/sec while HBBFT and CONS1 throughputs drop less but from a lower peak. As we will show in §1.1.2 and §1.1.3, there are factors other than verification that have a larger impact on these algorithms.

1.1.2 Combining proposals

Figure 2 explored the effect of deciding the unions of proposals when running the blockchain. CONS1 has the lowest latency because in all executions the leader acts correctly, allowing it to terminate in only 3 message delays, where RBBC with DBFT requires 4 message delays. Probably due to its inherent concurrency, RBBC offers the best latency/throughput tradeoff: at 1000 ms latency, RBBC offers 12,100 tx/sec whereas at 1750 ms latency, CONS1 offers only 5800 tx/sec. Finally, the blockchain with HBBFT has the worst performance for several reasons: HBBFT relies on a consensus algorithm [8] whose termination is randomized and it uses erasure codes: the computation time needed for performing reliable broadcast using erasure codes on a single node with a proposal size of 1000 transactions takes over 200 ms. Each node then has to do this for each proposal (i.e., 140 times in this experiment) increasing significantly the latency.

1.1.3 Low-end machines and distributed proposals

We now experiment on up to 240 low-end VMs, whose CPU resource is closer to the one of cell phones, and evenly spread

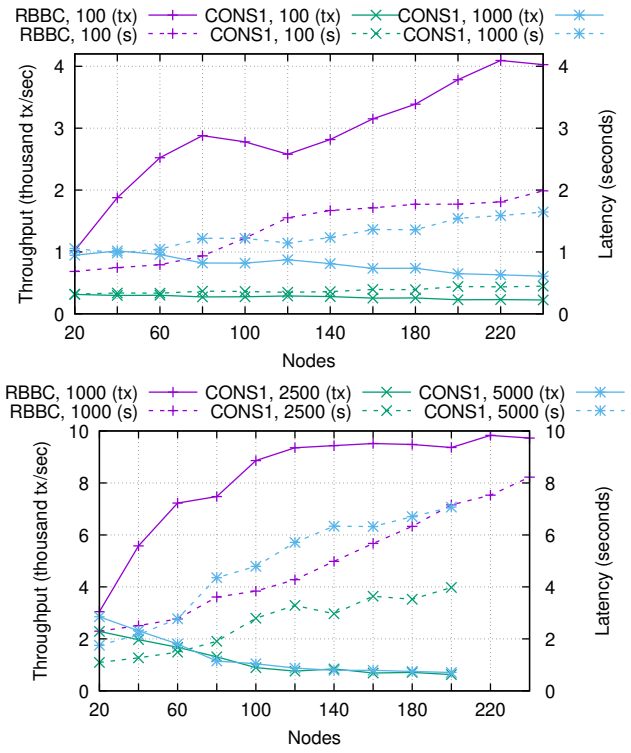

Figure 3. The performance of CONS1 and RBBC with $t + 1$ proposer nodes; the number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency

on 5 datacenters in the United States (Oregon, Northern California, and Ohio) and Europe (Ireland and Frankfurt). We examine the impact of having $t + 1$ vs. n proposers. Dedicating the 4 vCPUs of these low-end instances led to verify about 7800 serialized transactions per second with 97% of CPU time spent verifying signatures and with 3% spent deserializing and updating the UTXO table.

$t+1$ proposer nodes. Figure 3 shows the throughput and latency of RBBC with $t+1$ proposers and CONS1 with different sizes of proposals. As CONS1 is limited to a single proposer (its leader) while RBBC supports multiple proposers, we tested whether CONS1 performance would be better with more transactions per proposal than RBBC.

With proposal size of 100, RBBC throughput increases from 1000 to 4000 tx/sec while its latency increases from 750 ms to 2 seconds. The throughput increase stems from increasing CPU and bandwidth resources with more proposers. With larger proposal size (1000), performance increases faster (from 3000 tx/sec to 9000 tx/sec) with the number of nodes and flattens out earlier around 10,000 tx/sec while latency increases from 2 to 8 seconds.

With proposal size of 100, CONS1 throughput decreases from 310 tx/sec to 220 tx/sec while latency increases from 320 ms to 460 ms. Unfortunately, this low latency does not

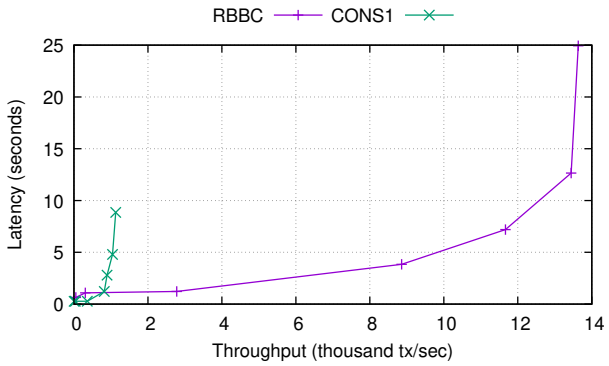

Figure 4. Comparing throughput and latency of CONS1 and RBBC with $t + 1$ proposer nodes on 100 geodistributed nodes; each point represents the number of transactions in the proposals, either 10, 100, 1000, 2500, 5000 or 10000 (HBBFT does not appear due to lower performance)

help to increase throughput by increasing proposal size after a certain number of nodes. In particular, with proposal size of 5000 the throughput drops by 4 times (from 2800 tx/sec to 700 tx/sec). While CONS1 can broadcast message authentication codes (MACs) through UDP in local area networks, no such broadcast primitive is available in this wide area testnet.

Figure 4 further examines the performance of CONS1 and RBBC with 100 nodes and proposal sizes of 1, 10, 100, 1000, 2500, and 5000. Here we see that the throughput of CONS1 reaches a limit of about 1100 tx/sec while RBBC approaches 14,000 tx/sec. CONS1 has a better minimum latency of 270 ms compared to 640 ms for RBBC for proposals of size 1.

n proposer nodes. Figure 5 depicts the performance of RBBC and HBBFT with n proposers, with proposal sizes of 100 and 1000 transactions. Unsurprisingly, with n proposers the throughput of RBBC increases faster than with $t + 1$ proposers. With a proposal size of 100, the throughput reaches 6000 tx/sec at 80 nodes and slowly degrades, while latency starts at 740 ms with 20 nodes and reaches 5160 ms with 240 nodes. With a proposal size of 1000, the throughput reaches 10,000 tx/sec at 40 nodes and remains mostly flat, latency starts at 2670 ms with 20 nodes and reaches 25,100 ms with 240 nodes. With larger node counts (around 200), the configurations with $t + 1$ proposals achieve similar throughput, but with much lower latency. Thus when using nodes similar to the low-end instances, having n proposers seems better suited for configurations of less than 100 nodes.

For HBBFT we observe that latencies increase superlinearly and throughput degrades as we increase the number of nodes. As mentioned before, this is primarily due to the computation needed for the erasure codes. Note that we only run HBBFT up to 100 nodes as afterwards we start seeing latencies approaching minutes.

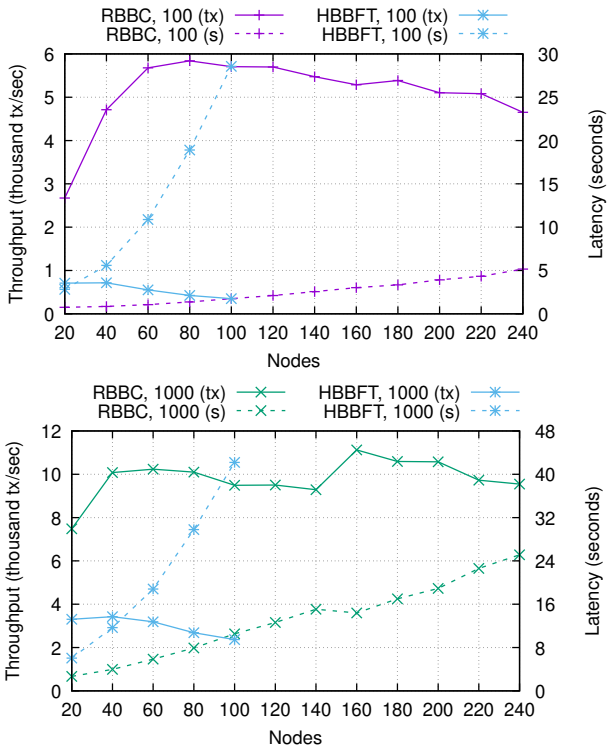

Figure 5. The performance of HBBFT and RBBC with n proposer nodes. The number following the algorithm name represents the number of transactions in the proposals; solid lines represent throughput, dashed lines represent latency

1.1.4 Transaction verification count

In the previous experiments we also recorded the average number of times a transaction is verified to examine the state of sharded verification, the results are shown in Figure 9. The best case is $t + 1$ verifications while the $2t + 1$ is the worst case. We observe that with $t + 1$ proposers the number of verifications stays close to the optimal, while with n proposers the number of verifications remains around the middle of $t + 1$ and $2t + 1$. This is likely due to the increased load on the system causing verifications to occur in different orders at different nodes.

1.2 Experiment under Byzantine attacks

We evaluate RBBC performance under 2 Byzantine attacks:

Byz1 The payload of the reliable broadcast messages altered so that no proposal is delivered for reliable broadcast instances led by faulty nodes. The binary payloads of the binary consensus messages are flipped. The goal of this behavior is to reduce throughput and increase latency.

Byz2 The Byzantine nodes form a coalition in order to maximize the bandwidth cost of the reliable broadcast using the digests described in §??. As a result, for any reliable broadcast initiated by a Byzantine node, $t + 1$ correct nodes will deliver the full message while the remaining t will only deliver the digest of the message, meaning they will have to request the full message from $t + 1$ different nodes from whom they receive echo messages.

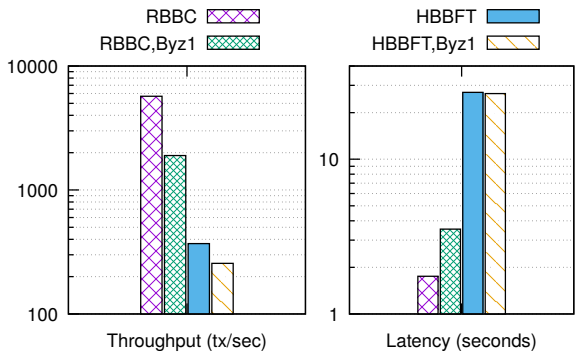

Figure 6. Comparing throughput and latency of RBBC and HBBFT, with normal and Byzantine behavior on 100 geodistributed nodes; all n nodes are making proposals of 100 transactions

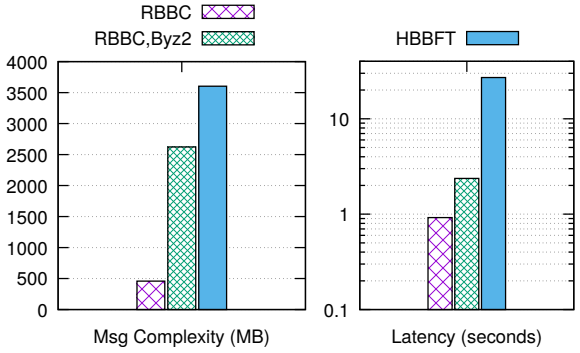

Figure 7. Comparing message complexity and latency of RBBFT and HBBFT with normal and Byzantine behaviors on 100 geodistributed nodes

As in §1.1.3, experiments are run with 100 low-end machines using the same 5 datacenters from US and Europe and with n proposers. Figure 6 shows the impact of Byz1 on performance with n proposers and proposal sizes of 100. For RBBC, throughput drops from 5700 tx/sec to 1900 tx/sec, and latency increases from 920 ms to 1750 ms. The drop in throughput is partially due to having t less proposals being accepted (the proposals sent by Byzantine nodes are invalid), and to the increase in latency. The increase in latency is due to the extra rounds needed to be executed by the binary consensus to terminate with 0. The throughput of HBBFT drops from 350 to 256 tx/sec due to the decrease in proposals, but interestingly the latency also decreases. This is due to the fact that since there are less proposals, less computation is needed for the erasure codes.

Byz2 is a behavior designed against the digest based reliable broadcast implementation described in §??, with the

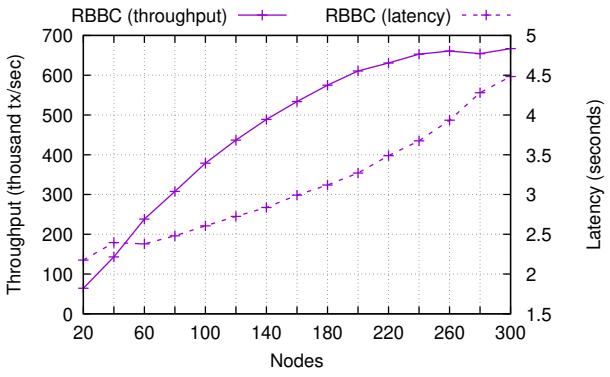

Figure 8. The performance (latency and throughput) of RBBC in a single datacenter

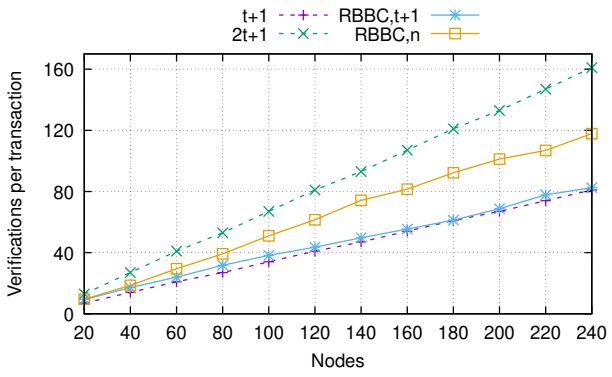

Figure 9. The number of times a transaction is verified in RBBC with proposal size of 100 transactions, with either $t + 1$ or n proposer nodes; the dashed lines $t + 1$ and $2t + 1$ represent the minimum and maximum number of possible verifications.

goal of delaying the delivery of the message to t of the correct nodes, and increasing the bandwidth used. HBBFT avoids this problem by using erasure codes, but has a higher bandwidth usage is the non-faulty case. Figure 7 shows the impact of this behavior on bandwidth usage and latency for RBBC and HBBFT with n proposers and proposal sizes of 100. The bandwidth usage of RBBC increases from 538 MB per multivalued consensus instance to 2622 MB per multivalued consensus instance compared to HBBFT which uses 3600 MB in all cases. Furthermore, the latency of RBBC increases from 920 ms to 2300 ms. Note that the bandwidth usage can further increase if additional delays are added to the network, in such cases the use of Erasure codes becomes beneficial.

1.3 Single availability zone experiment

To really stress test RBBC, we tested the performance on 300 high-end VMs in the Oregon datacenter. We fixed t to the largest fault tolerance parameter we can tolerate with $n = 20$ nodes and increase the number of nodes from 20 to 300 permissioned nodes. While the setting is not realistic, it helps identifying potential performance bottlenecks. Note that Fig. 1 depicts the impact of varying t on performance. The results, shown in Figure 8, indicates that the throughput scales up to $n = 260$ nodes to reach 660,000 tx/sec while the latency remains lower than 4 seconds. At $n = 280$ throughput drops slightly. Other experiments not shown here indicated about 8 verifications per transaction converging towards $7 = t + 1$ as n increases. The performance is thus explained by the fact that the system is CPU-bound up to $n = 260$, so that increasing n adds CPU resources needed for the sharded verification and improves performance, after what the system becomes network-bound due to the consensus and performance flattens out.

1.4 Impact of remote requesters

For the following experiments we run the blockchain with requesters defined as follows. At the start of the benchmark each requester is assigned a random private key and a single UTXO contained within the genesis block with value 100,000 coins. The requester then loops over the following two steps until the benchmark completes: *(i)* For each UTXO currently assigned to the requester a new transaction is created using that UTXO as input. For the transaction's output a UTXO is created using a randomly chosen account as the receiver with a value of 10 coins. Any change is included in a second UTXO sent back to the requester. Each transaction is then broadcast to the requester's assigned proposers. *(ii)* The requester then repeatably performs the request_utxos(*account*) operation until it receives at lest one new UTXO and then returns to step (i). Each requester is run in its own thread and maintains connections to $2t + 1$ of the blockchain nodes, including the requester's $t + 1$ proposers (all CONS1 requesters have the same primary proposer).

For this experiment we ran RBBC and CONS1 using 100 c4.4xLarge server instances 25 c4.4xLarge requester instances. Both types of nodes are evenly distributed across the 5 datacenters from US and Europe. The c4.4xLarge instances use Intel Xeon E5-2666 v3 processors with 16 vCPUs, and 30 GiB RAM. The number of requesters vary from 1,000 to 50,000 and are evenly distributed across the requester nodes. For the proposal size β, we choose 1000 for RBBFT as it gave the best throughput. For CONS1 we chose a proposal size of 2500 as we found that larger sizes increased latency without increasing throughput and smaller sizes decreased throughput while only having a minor impact on latency. The experiments were run for 45 sec with a 15 sec warmup.

Table 2 shows the results. *Valid-tx/sec* is average number of valid transactions committed per second, *Read/sec* is the average number of request_utxos(*account*) operations performed per second, *R/W ratio* is the ratio of the previous two values, *Latency* is the average amount of time between committed blocks, *Valid-tx/block* is the average number of valid transactions per block, and *Invalid-tx/block* is the average number of invalid transactions per block.

	#Requesters	Valid-tx/sec	Read/sec	R/W ratio	Latency(ms)	Valid-tx/block	Invalid-tx/block
RBBFT	1,000	5,359	2,143	0.4	870	4,648	0
	10,000	13,870	33,288	2.4	2,475	34,132	877
	20,000	12,664	31,660	2.5	5,022	63,607	3,033
	50,000	14,450	47,685	3.3	4,303	62,193	5,455
CONS1	1,000	3,759	1,127	0.3	401	1,513	0
	10,000	3,309	6,278	1.9	359	1,172	0
	20,000	4,064	10,566	2.6	488	1,981	0
	50,000	4,035	12,509	3.1	625	2,500	0

Table 2. Performance of RBBFT and CONS1 with varying number of requesters.

#Replicas	#Requesters	Valid-tx/sec	Async write latency(ms)	Latency(ms)	Valid-tx/block	Invalid-tx/block
1000	8400	30684	238	3103	95407	378

Table 3. Performance of RBBFT with 1000 replicas spread in 14 data centers.

Similar to the previous experiments we see that RBBFT has the highest maximum throughput of 14, 450 tx/sec compared to 4, 064 with CONS1. RBBFT has the highest maximum latency between blocks of 5, 022 milliseconds compared to a maximum of 625 milliseconds for CONS1. The higher throughput and latency is explained by the higher utilization of resources by the sharded proposers and reduced computation needed for sharded verification. In RBBFT increasing the number of requesters past 10,000 has little impact on the throughput as the system resources are already saturated by this point, as a result we see an increase in the R/W ratio as it takes longer for each individual node's transaction to complete. A similar pattern is shown by CONS1, though this starts at 1,000 requesters as they are limited by the single primary proposer. Furthermore in RBBFT, increasing the number of requesters also increases the number of duplicate transactions occurring in blocks. This is due to the increased load in they system causing slower nodes to miss their proposals resulting in transactions being committed by secondary proposers.

1.5 Evaluation with 1000 VMs

To confirm that our blockchain scales to a large number of machines, we spawned 1000 VMs. To avoid wasting bandwidth, we segregated the roles: all 1000 VMs act as servers, keeping a local copy of the balances of all accounts. On these replicas, 10 requesters per 840 c4.large machines (60 VMs in each of 14 datacenters) send transactions and 160 c4.8xlarge machines (40 machines in each of the Ireland, London, Ohio and Oregon datacenters) decide upon each block. Each of the 8400 requesters start with 100 UTXOs and each proposal contains up to 1000 transactions. Performance are depicted in Table 3: throughput is only around 30,000 tx/sec due to the difficulty of generating the workload: the replicas are located in 14 different datacenters and have to wait for a UTXO to request a transaction that consumes it (cf. §1.4). The asynchronous write latency measures the time a proposer acknowledges a transaction reception. The transaction commit time (latency) remains about 3 seconds despite the large traffic.

References

[1] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. *ACM Trans. Comput. Syst.*, 32(4):12:1–12:45, January 2015.

[2] A. Bessani, J. Sousa, and E. E. P. Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, June 2014.

[3] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, November 2002.

[4] Kwon J. Tendermint: Consensus without mining, 2014.

[5] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: Speculative byzantine fault tolerance. In *Proceedings of Twenty-first ACM SIGOPS Symposium on Operating Systems Principles*, SOSP '07, pages 45–58, New York, NY, USA, 2007. ACM.

[6] Jean-Phillippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, July 2006.

[7] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42, New York, NY, USA, 2016. ACM.

[8] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous byzantine consensus with t > n/3 and o(n2) messages. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 2–9, New York, NY, USA, 2014. ACM.

[9] Satoshi Nakamoto. Bitcoin: a peer-to-peer electronic cash system, 2008. http://www.bitcoin.org.

[10] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger, 2015. Yellow paper.