Scalable but Wasteful: Current State of Replication in the Cloud

Venkata Swaroop Matte
The Pennsylvania State University

Aleksey Charapko
University of New Hampshire

Abutalib Aghayev
The Pennsylvania State University

ABSTRACT

Consensus protocols are at the core of strongly consistent replication deployed in cloud-based storage systems. There have been many proposals to optimize these protocols, most of which work by identifying and shifting load from bottlenecked nodes to underutilized nodes.

We show that while these optimizations increase throughput, they sacrifice resource efficiency, which is paramount in a cloud setting. We propose a new metric to measure the efficiency of these protocols and show that using this metric, for example, the optimized EPaxos protocol is less efficient than the unoptimized Multi-Paxos protocol. We then demonstrate that Multi-Paxos can achieve 2× higher throughput than EPaxos in a fixed-budget resource setting that is typical of the cloud. Our work underlines the need for considering resource efficiency when optimizing consensus protocols, given that they are increasingly deployed in the cloud.

ACM Reference Format:
Venkata Swaroop Matte, Aleksey Charapko, and Abutalib Aghayev. 2021. Scalable but Wasteful: Current State of Replication in the Cloud. In *Proceedings of 13th ACM Workshop on Hot Topics in Storage and File Systems (HotStorage'21).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.475/1234.5678

1 INTRODUCTION

Strongly consistent replication is the backbone of modern cloud services, ranging from configuration management [13, 29] to cloud datastores [8, 22, 26, 28, 33, 39]. To ensure strong consistency, these services typically implement state machine replication using consensus protocols [4], **henceforth referred to as replication protocols**, which need to scale and provide high-throughput, low-latency services to the users. Recently, many new replication protocols have emerged, all improving throughput, latency, or both.

These optimized replication protocols follow a similar recipe: they identify one or more bottlenecked components in a traditional solution and try to alleviate the bottleneck by shifting the work elsewhere. For example, many traditional protocols [2, 17, 24, 32] rely on a dedicated leader to prescribe the operations and their order to the follower nodes. EPaxos [23] conjectures that a single leader is a bottleneck for both the throughput and latency and removes it by allowing any node to become an opportunistic leader; other recent systems [9, 31] further optimize EPaxos. PigPaxos [7] moves a significant portion of communication from the leader onto the followers. Compartmentalized Paxos [35] separates Paxos into multiple distinct roles or compartments; many of these roles, such as batching nodes or communication relays scale horizontally, allowing them to be deployed on separate VMs to offload the leader and improve throughput.

Although the aforementioned and similar solutions improve the throughput, they all share a common pitfall: **they increase the throughput at the cost of reducing efficiency**. This is because they assume dedicated clusters with identical nodes, where one of the nodes is bottlenecked (fully utilized) in a traditional baseline protocol while the remaining nodes are underutilized. As such, the optimized protocols try to spread the load from the bottlenecked node to the remaining nodes **with no regard for efficiency** because the resources in the underutilized nodes are idle anyway. Such load-shifting techniques, however, may not be ideal for sharded cloud-native services that operate in "resource-packed" cloud environments [5, 34] where every unit of resource costs money.

Consider an example of Multi-Paxos [32] (or Raft [24], or a primary-backup [2]) replication scheme deployed on dedicated nodes, as shown in Figure 1 (a). These protocols rely on a single leader, and it is expected for a leader to do more work and use more resources than the followers [1], leaving some idle resources at the follower nodes. EPaxos, and similar works that aim to scale consistent replication, assume such a setting with identical nodes; they spread the load evenly among the nodes at the expense of added complexity and, most importantly, reduced efficiency. As such EPaxos

HotStorage'21, July 27-28, 2021, virtual conference

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/1234.5678

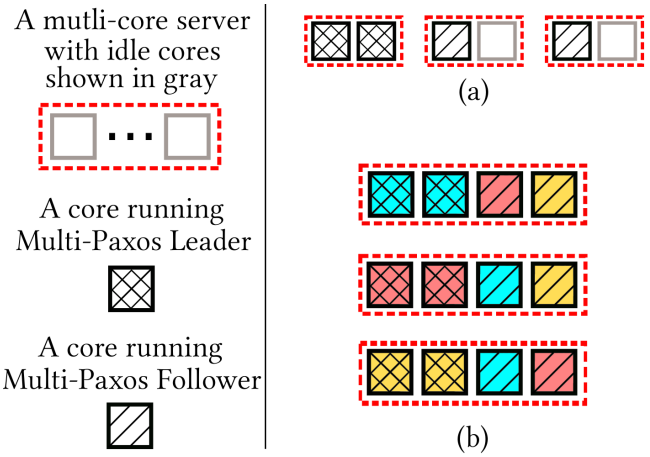

Figure 1: (a) A single instance of Multi-Paxos running on three dedicated nodes each with two cores. The Multi-Paxos leader uses both cores of a node since it is CPU-intensive, whereas Multi-Paxos followers use one core each. (b) Three instances of Multi-Paxos (each instance shown in a different color) running in a cloud environment on three 4-core nodes, where the leader and followers are spread out over the nodes.

can take advantage of resources left unused by leader-based replication and provide higher throughput.

Modern cloud services, however, rarely operate on dedicated resources; on the contrary, physical resource sharing with some resource isolation is the norm. Moreover, most systems scale horizontally by running multiple protocol instances side-by-side to support different data partitions [8, 26, 28, 36]. In such shared environments, the resources are allocated and scheduled using some sort of task packing [11], leaving only limited free resources on each node.

We illustrate a hypothetical example of such task packing in Figure 1 (b), where three instances of Multi-Paxos share a set of 4-core servers and get packed in such a way that the leader of one instance is co-located with the followers from the other instances. In our example, the leader uses two cores, while the followers use one core each. Despite the resource usage disparity, packing multiple instances of the protocol achieves an equal load on all physical nodes. Additionally, such packed deployment needs fewer cores than three dedicated Multi-Paxos instances as in Figure 1 (a).

Given the same budget of resources, however, protocols such as EPaxos are at a disadvantage because they are less efficient. That is because resource balancing and bottleneck avoidance in these protocols often come at the price of added protocol complexity and increased overall resource usage. In this paper, we show that **while throughput-optimized protocols deliver better throughput than their simpler counterparts when given dedicated resources, these same protocols may significantly underperform in resource-shared environments.** Our findings show that we, as a community, have been neglecting the efficiency aspect of scaling consensus-based replication. We believe that efficiency, in addition to traditional performance metrics, is essential as these protocols are increasingly being deployed in the cloud. Indeed, in our survey, we were unable to find papers that propose optimized consensus-based state machine replication protocols to also include CPU or memory usage of the proposed optimizations.

We argue for the need for *efficiency* optimizations that allow processing more operations using the same pool of resources in addition to optimizations that take advantage of more added resources. To that order, we propose to evaluate the replication algorithms under more scrutiny and refrain from judging them only by the best or maximum performance metrics. **We suggest a new metric for inclusion to typical protocol evaluations: throughput-per-unit-of-constraining-resource-utilization.** This metric normalizes the performance of a protocol relative to the consumption of the constraining resource.

We illustrate our metric on Multi-Paxos and EPaxos protocols and show that it serves as a good proxy for the performance of the replication systems in sharded resource-shared environments. This is because a more efficient system can pack more copies of itself in the same pool of cloud resources than a less efficient protocol, despite the more efficient system being less capable in absolute terms when used with dedicated resources. For instance, in our experiments, we demonstrate that although EPaxos outperforms Multi-Paxos by nearly 20% on a set of dedicated VMs, when we deploy multiple instances of the protocols on a fixed set of shared resources, Multi-Paxos provides almost twice the throughput of EPaxos.

Our work applies the metric to CPU usage alone, but it can also be applied to other constraining resources, such as memory or network bandwidth. With the help of this new metric focused on resource efficiency instead of absolute performance, we hope that the community will start optimizing replication protocols for the modern cloud infrastructure.

2 REPLICTION IN THE WILD

State machine replication protocols are deployed in many systems and applications spanning a variety of environments. These environments range from dedicated hardware deployments in the on-premise data centers to fully managed public cloud solutions. Most state machines, however, gravitate away from fully dedicated clusters even in the on-premise

setting due to cost. When an on-premise cluster has lots of unused hardware, it is tempting to deploy all systems in their own isolated environments. Over time, though, this becomes more challenging as the number of systems grows without expanding the on-premise data center. As such, resource sharing (Figure 1 (b)) becomes increasingly important, especially considering the multi-core nature of modern servers. The need to deploy state machine replication protocol in the resource shared environments also grows when considering the migration from on-premise to the public cloud. The public cloud model further encourages resource sharing with its pay-as-you-go cost model that charges for resources allocated or used.

Many state-of-the-art storage systems take advantage of resource sharing. Systems like Spanner [8], CockroachDB [28], and YugabyteDB [36] all use small instances of Multi-Paxos or Raft [24], such that each physical machine or VM supports many replicas from different partitions. For example, CockroachDB relies on 64MB partitions, allowing for hundreds if not thousands of replicas on the same physical server. DocumentDB [27], now known as Azure Cosmos DB [22], also claims a load-balanced approach where primary-backup [2] replicas are strategically assigned to servers to balance the load in the federation.

3 CURRENT APPROACHES TO SCALING STATE MACHINE REPLICATION

Given the pervasiveness of replicated state machines in distributed systems and applications, improving the replication throughput has been an important problem in the past decade. Some solutions [9, 23] call for boosting throughput and reducing latency at the same time, while others [7, 35] argue for trading off some latency in exchange for better throughput. Both camps, however, take a similar high-level approach for improving their performance. They first identify a bottleneck in traditional systems and then work to eliminate the bottleneck by moving it elsewhere. Usually, the bottleneck is at the leader, as it is used to communicate with the clients and coordinate the replication [1].

Systems like EPaxos [23, 31] and Atlas [9] avoid having a one-node bottleneck by not having a single leader that centers all communication around it. Instead, these systems expand on Fast Paxos [17] ideas and try to use *fast quorums* to commit/replicate operations in one round-trip network latency from any node in the system. This approach, commonly known as leaderless consensus, allows any node to become an opportunistic leader for an operation in the hopes that the operation won't conflict with any other concurrent replication initiated by other nodes. When conflicts do occur, leaderless solutions fall back to a more traditional Paxos protocol to resolve the ordering. The added benefit of opportunistic protocols is reduced latency, as the system may immediately start replication and avoid routing all operations to the centralized leader.

Some solutions like SDPaxos [38], separate the replication from operation ordering when implementing replicated state machines. This allows the payload to be accepted and replicated by every node in the system. However, a lightweight designated leader or sequencer is still used to establish the order among all replicated operations.

Other protocols keep a centralized leader to avoid the complexity of determining and resolving the conflicts and instead offload as much work from the leader as possible. Both Compartmentalized Paxos [35] and PigPaxos [7] fall into this category; they use some form of intermediate nodes to handle most of the leader's communication and processing responsibilities, shift the bottleneck away *from the leader and onto other Paxos components*. Compartmentalized Paxos goes back to the roots of Paxos [16] protocol and implements all the standard Paxos roles (proposers, acceptors, learners, etc.) as stand-alone components, called compartments. It adds a few additional roles, such as leader proxies for relaying leader communication and batching nodes to aggregate operations together. Such role separation coupled with the additional roles allows serial compartments, like proposers, to run on their own dedicated VMs, leaving them more spare resources. PigPaxos, on the other hand, retains the combined replica roles popular in the practical implementations of Multi-Paxos and Raft, and instead delegates some followers to act as a communication proxy on behalf of a leader. Another similar approach, Linearizable Quorum Reads [6], shifts processing away *from Paxos components to the clients*; more specifically, it allows the clients to read directly from the quorum of followers, bypassing the leader entirely.

The general approach of moving the processing away from bottlenecked component allows the protocols to reach higher throughput by utilizing the resources previously left unused due to the bottleneck. To confirm this, we run EPaxos and Multi-Paxos on five AWS EC2 *m5a.large* instances with 2 vCPUs and 8 GiB of RAM. We use a 50% write workload targeting an in-memory key-value store where items are selected uniformly randomly. Our experiment uses up to 90 concurrent, closed-loop clients to saturate the cluster and observe its maximum throughput. For EPaxos, we use a workload with up to a 10% conflict ratio. As Figure 2 (a) shows, EPaxos outperforms Multi-Paxos by about 20%—an observation consistent with the original EPaxos evaluation [23].

Similar to EPaxos, the original evaluations of all improved protocols focus solely on the absolute best performance that was unlocked by avoiding the bottleneck and utilizing idle resources of dedicated nodes. The systems today, however, are

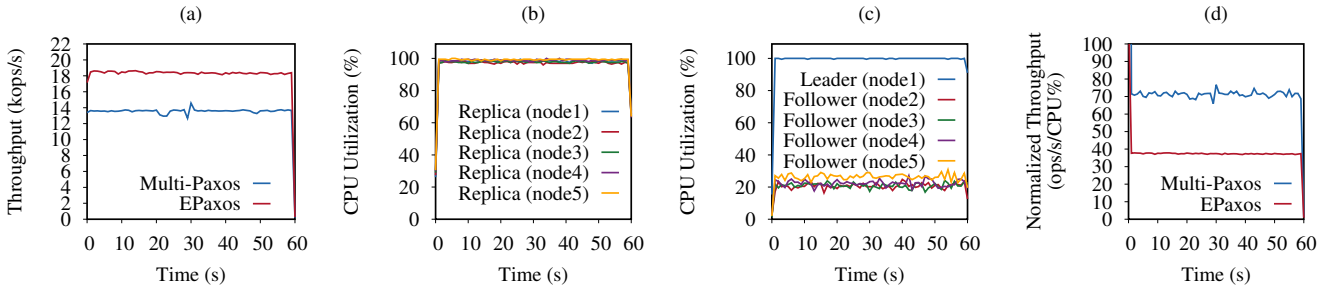

Figure 2: (a) Throughput of EPaxos and Multi-Paxos on five AWS EC2 m5a.large VMs. CPU utilization of each (b) EPaxos and (c) Multi-Paxos node. (d) Throughput of EPaxos and Multi-Paxos per unit of aggregate CPU utilization.

increasingly deployed on the cloud, making such evaluations inadequate for resource-shared environments.

4 EFFICIENCY METRIC

The general scalability approach taken by the new protocols that we described earlier has a major disadvantage. The elimination of the bottleneck and shifting processing elsewhere is not free and requires more complex and nuanced algorithms to make these systems work. As a result, while these systems can unlock the unused resources of dedicated nodes, they do so at the expense of using more resources.

We confirm this by measuring the CPU usage of EPaxos from the previous experiment. As Figure 2 (b) shows, all nodes running EPaxos replicas are fully utilized. (Since all nodes in EPaxos have uniform roles, we call them replicas and not leaders or followers.) To our surprise, none of the evaluations in EPaxos, its derivatives, and similar work provides measurements of resource usage (CPU, memory, network bandwidth) during the experiments.

Having information about the protocol's resource consumption can help determine the added cost of avoiding the bottleneck and rate its suitability for a resource-packed cloud setting. For example, unlike EPaxos, Multi-Paxos consumes a lot less CPU, as Figure 2 (c) shows. More specifically, Multi-Paxos fully utilizes only the leader node, while each follower uses only about a quarter of its node's CPU, effectively leaving $0.75 * 4 = 3$ nodes worth of CPU unused in the cluster. While not ideal for the protocol's own load balancing, these unused resources also mean that Multi-Paxos can achieve roughly 80% of EPaxos' throughput using just 40% of CPU.

To understand the efficiency of replication protocols, we propose a new performance metric: *throughput-per-unit-of-constraining-resource-utilization.* We can apply the metric to any cluster resource used by the protocol, such as combined memory or CPU usage across all nodes. However, the most useful resource to study is the limiting or constraining resource that causes a bottleneck on one or more machines. In the context of replicating small payloads (Figure 2), such limiting resource is likely to be CPU, as we see that both protocols reach nearly 100% utilization on at least one node. In other settings and under different workloads the constraining resource may change.

A handful of cloud and database systems [20, 25] consider using price/cost as a proxy for the system's efficiency. The cost metric normalizes the performance per dollar spent. This, however, serves as an indirect measure of the true resource efficiency of protocols, as cost per unit of resource may vary between vendors and over time. Moreover, the cost measure often encompasses the price of multiple resources bundled together, hiding the impacts a single resource may have on limiting the scalability of the protocol. For instance, if a cost metric incorporates multiple distinct resources, such as RAM, storage IOPS, and CPU, it becomes unclear which of these bundled resources is a constraining one and must be optimized to improve performance at the fixed cost. Furthermore, for some systems or deployments, the cost estimates are based on the allocated resources, and not resources actually consumed, which again favors protocols that can use all allocated resources and not more efficient ones. For these reasons, we chose to focus on a per-resource efficiency instead of the cost measurement, especially considering that the efficiency can be translated to cost in the cloud if the resource rates are known. The resource efficiency metric can also help to improve on-premise deployments, as estimating the monetary cost in these environments may be more difficult.

We now revisit the efficiency of EPaxos and Multi-Paxos protocols in light of the proposed metric. To this end, we measure the CPU utilization percentage on each node once every second and sum up these utilizations to get an aggregate CPU utilization value in the cluster. We then align the rolling throughput measurements with this aggregate value to arrive at the CPU-normalized performance expressed as throughput-per-unit-of-aggregate-CPU-utilization (ops/s/CPU%). As Figure 2 (d) shows, this new efficiency metric paints a drastically different picture than the

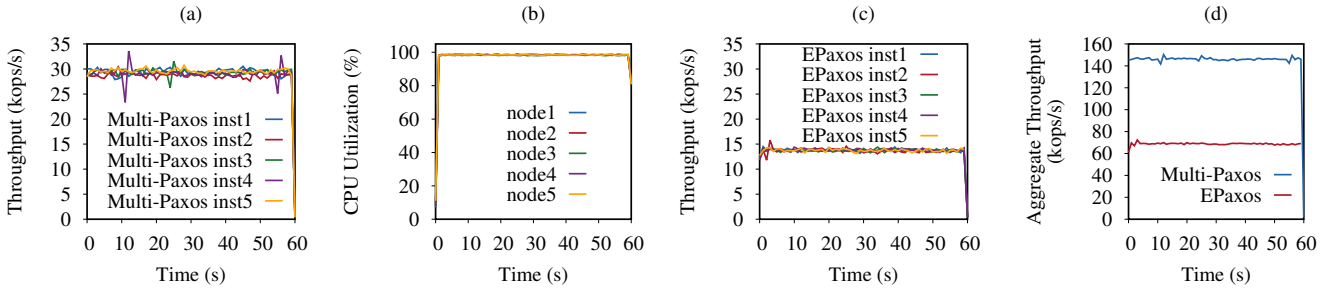

Figure 3: (a) Throughput of five Multi-Paxos instances on five AWS EC2 m5a.2xlarge VMs. Each instance is spread over all VMs, as shown in Figure 1. (b) CPU utilization of each VM when running five Multi-Paxos instances. (c) Throughput of five EPaxos instances running on five AWS EC2 m5a.2xlarge VMs. (d) Aggregated throughput of five Multi-Paxos instances and five EPaxos instances running on five m5a.2xlarge VMs.

raw performance numbers shown in Figure 2 (a). The latter shows EPaxos having higher throughput than Multi-Paxos in a dedicated cluster, essentially illustrating that EPaxos is more efficient given the *resources allocated*. The same experimental run expressed with our new efficiency metric in Figure 2 (d) shows that Multi-Paxos is nearly twice as efficient than EPaxos when considering the *resources consumed*. This confirms our intuition that the extra processing requirements of EPaxos, as it needs to check each operation against the dependency graph for possible conflicts, drastically impact its efficiency. Multi-Paxos, on the other hand, is a less complicated protocol that requires only a few simple comparisons to determine the validity of each operation.

While we focused on just two protocols and one resource, the metric applies to any replication protocol and a variety of resources. For instance, some implementations of primary backup [2] may be more CPU-efficient at the leader than the consensus protocols due to slightly different messaging requirements. Similarly, having a disk-bound state machine instead of a memory-bound one may put more strain on the storage subsystem. The efficiency metric can also be useful for other data-driven protocols and systems that have replication at their core, such as distributed queues or logs [3], pub-sub systems [15], and transaction processing protocols [30, 37].

5 INEFFECTIVENESS OF CURRENT APPROACHES PER NEW METRIC

Dedicated server or VM deployments of state machine replication waste allocated but unused resources, making better-balanced protocols like EPaxos or PigPaxos more desirable. Many cloud-native systems, however, rely on shared environments for better utilization of available resources. More importantly, these shared environments are often constrained by a budget, making efficiency paramount to extracting as much work from a fixed pool of resources. In this section, we demonstrate how more efficient protocols, such as Multi-Paxos, achieve higher aggregated throughput despite having worse protocol-level load balancing characteristics.

In a modern data center, large storage systems scale horizontally in a data-parallel manner by providing strong consistency in different data partitions using independent instances of replicated state machines. This allows a more comprehensive resource balancing that is not confined within the bounds of a single replicated state machine instance. For example, resources left unused by one instance can be picked up by another one, given proper resource allocation and scheduling strategies. Luckily, modern containerized infrastructure facilitates such resource scheduling [5, 12, 34].

As a concrete example, let's consider a task-packing setup on a five-node cluster for simplicity. This cluster of machines may represent a fixed budget that we may be willing to spend on replication and storage infrastructure. In this setup, we can "task-pack" five Multi-Paxos instances such that in a fault-free case each node hosts one Multi-Paxos leader and four followers each belonging to a different Multi-Paxos instance (similar to Figure 1 but with five Multi-Paxos instances instead of four). This simple deployment, despite relying on a protocol with disproportionate resource usage between the leader and the follower nodes, achieves good resource usage by pairing resource-heavy roles with resource-light ones.

We demonstrate this by emulating such a "resource-shared cloud" scenario on an AWS EC2 cluster using five *m5a.2xlarge* VMs with 8 vCPUs and 32 GiB of RAM. We deploy five instances of Multi-Paxos using the aforementioned task-packing setup and measure the throughput of each Multi-Paxos instance and CPU utilization across the VMs while repeating the previous experiment (§ 3). We target each of the five instances separately with up to 100 concurrent closed-loop clients and aggregate the throughput data across all protocol instances. Figure 3 (a) shows that each Multi-Paxos

instance achieves roughly 30 kops/s of throughput and Figure 3 (b) confirms that CPUs on all nodes are fully utilized. (In this case, the throughput of a single Multi-Paxos instance is higher than the throughput in Figure 2 (a) due to a more powerful VM.)

For comparison, we also deploy five instances of EPaxos on the same cluster and repeat the experiment. This simulates two systems with the same number of shards running on identical clusters for a fair comparison. In theory, due to its ability to saturate the node's resource, we could have kept EPaxos as a single partition running on the cluster of larger machines. However, in practice scaling systems to work efficiently on high-core count machines is challenging, and rarely produces perfect speedup. Partitioned deployments are better in this regard, as different partitions running in separate processes have fewer contention points over shared software resources, such as locks, queues, and atomic counters. Figure 3 (c) shows that each instance of EPaxos achieves only 15 kops/s of throughput with CPUs on all nodes fully utilized (the graph omitted to save space). Per-instance throughput of EPaxos in such sharded and packed environment has dropped compared to our dedicated setup. This is because each of the five EPaxos instances had $\frac{8}{5}$ vCPUs available at each server (assuming the OS scheduling was fair), which is slightly under 2 vCPUs available in the dedicated experiment.

To summarize, Figure 3 (d) shows that the five instances of Multi-Paxos boast almost twice the aggregate throughput of that of five instances of EPaxos. Hence, compared to the dedicated resource experiment (Figure 2), a more efficient protocol has a significant advantage in a shared environment with proper task-packing.

6 CONCLUSION AND FUTURE DIRECTIONS

The emergence of cloud computing has dramatically transitioned how we build systems, but this transition is far from over. Many storage systems rely on strongly consistent replication protocols. These protocols, however, predate the cloud by decades, and many proposed optimizations on them continue to ignore the different reality put in front of us by cloud computing. In this work, we make a case for redesigning replication protocols for the cloud.

The key insight of our work is that while the optimizations proposed so far significantly improve the transaction throughput, they ignore resource efficiency, which is critical in a pay-as-you-go utility model of cloud computing. More specifically, these optimizations assume as a baseline a Multi-Paxos instance running on a dedicated cluster with identical nodes, where the leader node is the bottleneck and the follower nodes are underutilized. They then invent a more complex and *less resource efficient* protocol that spreads the load to the underutilized follower nodes. We show that when deployed in a typical cloud environment with a fixed budget of resources, these "optimized protocols" significantly underperform compared to "unoptimized protocols".

This begs the question of designing the protocols and systems that are more appropriate for resource-shared environments and specifically the cloud. To this end, we have introduced a new metric that takes into account resource efficiency by normalizing the throughput over the resource consumption. The metric serves as a good proxy for protocols' relative performance in sharded, task-packed environments.

Using this metric, we can identify bottlenecks in replication protocols and optimize them for modern environments. One potential source of optimizations is the emerging hardware and software technologies. These technologies may provide opportunities for further efficiency optimizations or efficiency trade-offs when efficiency over one resource improves at the expense of other, more abundant resources. For instance, Odyssey [10] argues for designing replication protocols with modern hardware in mind—taking into account the multi-core nature of servers and the availability of new communication technologies that bypass CPU, such as RDMA. Other modern implementation-driven methods, such as kernel bypass (i.e. DPDK [14]), may be used to reduce the CPU footprint of the protocols.

The optimizations don't have to stop at using new implementation technologies, and we can still do a lot to curb resource usage at the protocol level. For example, some of the older tricks, such as Cheap Paxos [18] may get handy to limit unnecessary communication. Use-case-specific optimizations are possible as well. For instance, in a system that frequently overwrites some small subset of data objects, a leader may forgo replicating an operation right away in anticipation that some shadowing operation may arrive shortly.

More radical or questionable ideas are worth exploring as well. For example, using ever-improving time synchronization [19] in the datacenters as an always-on implicit source of communication and synchronization may result in the protocol that avoids some of the explicit synchronizations and spare some resources.

These and other potential solutions require an understanding of the resource efficiency of protocols and systems. The importance of scaling without sacrificing efficiency has been brought up before in the context of big data platforms [21]. However, the efficiency problem is largely ignored in replication protocols and especially consensus-based replication. We think studying the efficiency is even more important in the context of protocols running on the cloud and we hope to see the emergence of a new generation of resource-efficient replication protocols designed for the cloud.

REFERENCES

[1] Ailidani Ailijiang, Aleksey Charapko, and Murat Demirbas. 2019. Dissecting the Performance of Strongly-Consistent Replication Protocols. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD 2019)*. 1696–1710.

[2] Peter A. Alsberg and John D. Day. 1976. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the 2nd International Conference on Software Engineering* (San Francisco, California, USA) *(ICSE '76)*. IEEE Computer Society Press, Washington, DC, USA, 562–570.

[3] Mahesh Balakrishnan, Dahlia Malkhi, Vijayan Prabhakaran, Ted Wobbler, Michael Wei, and John D. Davis. 2012. CORFU: A Shared Log Design for Flash Clusters. In *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*. USENIX Association, San Jose, CA, 1–14. https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/balakrishnan

[4] William J. Bolosky, Dexter Bradshaw, Randolph B. Haagens, Norbert P. Kusters, and Peng Li. 2011. Paxos Replicated State Machines as the Basis of a High-Performance Data Store. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/nsdi11/paxos-replicated-state-machines-basis-high-performance-data-store

[5] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. 2016. Borg, Omega, and Kubernetes: Lessons Learned from Three Container-Management Systems over a Decade. *Queue* 14, 1 (Jan. 2016), 70–93. https://doi.org/10.1145/2898442.2898444

[6] Aleksey Charapko, Ailidani Ailijiang, and Murat Demirbas. 2019. Linearizable quorum reads in Paxos. In *11th {USENIX} Workshop on Hot Topics in Storage and File Systems (HotStorage 19)*.

[7] Aleksey Charapko, Ailidani Ailijiang, and Murat Demirbas. 2021. PigPaxos: Devouring the Communication Bottlenecks in Distributed Consensus. In *Proceedings of the 2021 ACM SIGMOD International Conference on Management of Data (SIGMOD 2021) (SIGMOD '21)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3448016.3452834

[8] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems (TOCS)* 31, 3 (2013), 8.

[9] Vitor Enes, Carlos Baquero, Tuanir França Rezende, Alexey Gotsman, Matthieu Perrin, and Pierre Sutra. 2020. State-machine replication for planet-scale systems. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–15.

[10] Vasilis Gavrielatos, Antonios Katsarakis, and Vijay Nagarajan. 2021. Odyssey: The Impact of Modern Hardware on Strongly-Consistent Replication Protocols. In *Proceedings of the Sixteenth European Conference on Computer Systems* (Online Event, United Kingdom) *(EuroSys '21)*. Association for Computing Machinery, New York, NY, USA, 245–260. https://doi.org/10.1145/3447786.3456240

[11] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-Resource Packing for Cluster Schedulers. *SIGCOMM Comput. Commun. Rev.* 44, 4 (Aug. 2014), 455–466. https://doi.org/10.1145/2740070.2626334

[12] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*. USENIX Association, Boston, MA. https://www.usenix.org/conference/nsdi11/mesos-platform-fine-grained-resource-sharing-data-center

[13] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. 2010. ZooKeeper: wait-free coordination for internet-scale systems. In *Proceedings of the 2010 USENIX annual technical conference (ATC 2010)* (Boston, MA). USENIX Association, 11–11.

[14] Intel. 2014. DPDK: Data Plane Development Kit. https://www.dpdk.org/.

[15] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, Vol. 11. 1–7.

[16] Leslie Lamport. 1998. The Part-Time Parliament. *ACM Trans. Comput. Syst.* 16, 2 (May 1998), 133–169. https://doi.org/10.1145/279227.279229

[17] Leslie Lamport. 2006. Fast Paxos. *Distributed Computing* 19 (October 2006), 79–103. https://www.microsoft.com/en-us/research/publication/fast-paxos/

[18] L. Lamport and M. Massa. 2004. Cheap Paxos. In *International Conference on Dependable Systems and Networks, 2004.* 307–314. https://doi.org/10.1109/DSN.2004.1311900

[19] Yuliang Li, Gautam Kumar, Hema Hariharan, Hassan Wassel, Peter Hochschild, Dave Platt, Simon Sabato, Minlan Yu, Nandita Dukkipati, Prashant Chandra, and Amin Vahdat. 2020. Sundial: Fault-tolerant Clock Synchronization for Datacenters. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, 1171–1186. https://www.usenix.org/conference/osdi20/presentation/li-yuliang

[20] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. 2020. OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, 189–203. https://www.usenix.org/conference/atc20/presentation/mahgoub

[21] Frank McSherry, Michael Isard, and Derek G. Murray. 2015. Scalability! But at what COST?. In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*. USENIX Association, Kartause Ittingen, Switzerland. https://www.usenix.org/conference/hotos15/workshop-program/presentation/mcsherry

[22] Microsoft. 2021. Global data distribution with Azure Cosmos DB - under the hood. https://azure.microsoft.com/en-us/services/cosmos-db/.

[23] Iulian Moraru, David G Andersen, and Michael Kaminsky. 2013. There is more consensus in egalitarian parliaments. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 358–372.

[24] Diego Ongaro and John Ousterhout. 2014. In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX} ATC 14)*. 305–319.

[25] Qifan Pu, Shivaram Venkataraman, and Ion Stoica. 2019. Shuffling, Fast and Slow: Scalable Analytics on Serverless Infrastructure. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, Boston, MA, 193–206. https://www.usenix.org/conference/nsdi19/presentation/pu

[26] William Schultz, Tess Avitabile, and Alyson Cabral. 2019. Tunable Consistency in MongoDB. *Proc. VLDB Endow.* 12, 12 (Aug. 2019), 2071–2081. https://doi.org/10.14778/3352063.3352125

[27] Dharma Shukla, Shireesh Thota, Karthik Raman, Madhan Gajendran, Ankur Shah, Sergii Ziuzin, Krishnan Sundaram, Miguel Gonzalez Guajardo, Anna Wawrzyniak, Samer Boshra, Renato Ferreira, Mohamed Nassar, Michael Koltachev, Ji Huang, Sudipta Sengupta, Justin Levandoski, and David Lomet. 2015. Schema-Agnostic Indexing with Azure DocumentDB. *Proc. VLDB Endow.* 8, 12 (Aug. 2015), 1668–1679. https://doi.org/10.14778/2824032.2824065

[28] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin,

Raphael Poss, Paul Bardea, Amruta Ranade, Ben Darnell, Bram Gruneir, Justin Jaffray, Lucy Zhang, and Peter Mattis. 2020. CockroachDB: The Resilient Geo-Distributed SQL Database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD 2020)* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1493–1509. https://doi.org/10.1145/3318464.3386134

[29] Chunqiang Tang, Thawan Kooburat, Pradeep Venkatachalam, Akshay Chander, Zhe Wen, Aravind Narayanan, Patrick Dowell, and Robert Karl. 2015. Holistic Configuration Management at Facebook. In *Proceedings of the 25th Symposium on Operating Systems Principles (SOSP 2015)*. ACM, 328–343. https://doi.org/10.1145/2815400.2815401

[30] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. 2012. Calvin: Fast Distributed Transactions for Partitioned Database Systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data* (Scottsdale, Arizona, USA) *(SIGMOD '12)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/2213836.2213838

[31] Sarah Tollman, Seo Jin Park, and John Ousterhout. 2021. EPaxos Revisited. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association. https://www.usenix.org/conference/nsdi21/presentation/tollman

[32] Robbert van Renesse and Deniz Altinbuken. 2015. Paxos Made Moderately Complex. *ACM Comput. Surv.* 47, 3, Article 42 (Feb. 2015), 36 pages. https://doi.org/10.1145/2673577

[33] Alexandre Verbitski, Anurag Gupta, Debanjan Saha, Murali Brahmadesam, Kamal Gupta, Raman Mittal, Sailesh Krishnamurthy, Sandor Maurice, Tengiz Kharatishvili, and Xiaofeng Bao. 2017. Amazon aurora: Design considerations for high throughput cloud-native relational databases. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1041–1052.

[34] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*. Bordeaux, France.

[35] Michael Whittaker, Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas, Neil Giridharan, Joseph M. Hellerstein, Heidi Howard, Ion Stoica, and Adriana Szekeres. 2021. Scaling Replicated State Machines with Compartmentalization [Technical Report]. arXiv:cs.DC/2012.15762

[36] Yugabyte, Inc. 2021. YugabyteDB. https://www.yugabyte.com/.

[37] Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurthy, and Dan R. K. Ports. 2018. Building Consistent Transactions with Inconsistent Replication. *ACM Trans. Comput. Syst.* 35, 4, Article 12 (Dec. 2018), 37 pages. https://doi.org/10.1145/3269981

[38] Hanyu Zhao, Quanlu Zhang, Zhi Yang, Ming Wu, and Yafei Dai. 2018. SDPaxos: Building Efficient Semi-Decentralized Geo-Replicated State Machines. In *Proceedings of the ACM Symposium on Cloud Computing* (Carlsbad, CA, USA) *(SoCC '18)*. Association for Computing Machinery, New York, NY, USA, 68–81. https://doi.org/10.1145/3267809.3267837

[39] Jianjun Zheng, Qian Lin, Jiatao Xu, Cheng Wei, Chuwei Zeng, Pingan Yang, and Yunfan Zhang. 2017. PaxosStore: high-availability storage made practical in WeChat. *Proceedings of the VLDB Endowment* 10, 12 (2017), 1730–1741.