AFRL-AFOSR-VA-TR-2020-0094

Alvin: A Strongly Consistent and Highly Scalable Geo-Distributed Transactional Software System

Binoy Ravindran
VIRGINIA POLYTECHNIC INST AND STATE UNIVERSITY

12/19/2019
Final Report

Air Force Research Laboratory
AF Office Of Scientific Research (AFOSR)/ RTA2
Arlington, Virginia 22203
Air Force Materiel Command

DISTRIBUTION A: Distribution approved for public release.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
14-07-2020	Final Performance	31 Mar 2015 to 30 Sep 2019

4. TITLE AND SUBTITLE
Alvin: A Strongly Consistent and Highly Scalable Geo-Distributed Transactional Software System

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-15-1-0098

5c. PROGRAM ELEMENT NUMBER
61102F

6. AUTHOR(S)
Binoy Ravindran

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
VIRGINIA POLYTECHNIC INST AND STATE UNIVERSITY
300 TURNER ST NW, SUITE 4200
BLACKSBURG, VA 24061-0001 US

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AF Office of Scientific Research
875 N. Randolph St. Room 3112
Arlington, VA 22203

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/AFOSR RTA2

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-AFOSR-VA-TR-2020-0094

12. DISTRIBUTION/AVAILABILITY STATEMENT
A DISTRIBUTION UNLIMITED: PB Public Release

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Cloud computing's ubiquitous infrastructure-as-a-service (IaaS) model has enabled a broad range of enterprise organizations to roll out online services at low cost. Cloud's multi-region support allows computational resources to be instantiated from different data centers around the world, enabling new classes of geographical-scale (or 'geo-scale') applications.
State machine replication (SMR) is the de facto standard for building highly scalable and available distributed applications and services. SMR replicates a service across a set of nodes, and executes client operations on the replicas in an agreedupon total order, ensuring consistency of the replicated state. The problem of determining a total order reduces to one of consensus.

15. SUBJECT TERMS
Geo-Distributed, Transactional Software

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			NGUYEN, TRISTAN
Unclassified	Unclassified	Unclassified	UU		19b. TELEPHONE NUMBER *(Include area code)* 703-696-7796

Standard Form 298 (Rev. 8/98)
Prescribed by ANSI Std. Z39.18

Final Report

AFOSR Grant FA9550-15-1-0098

Alvin: A Strongly Consistent and Highly Scalable Geo-Distributed Transactional Software System

Program Officer: Dr. Tristan Nguyen, tristan.nguyen@us.af.mil
Program: Systems and Software, RTC-10

Institution:
Virginia Polytechnic Institute and State University
North End Center, Suite 4200
300 Turner Street, NW
Blacksburg, VA 24061-0001

Principal Investigator:
Binoy Ravindran
Department of Electrical and Computer Engineering
1991 Kraft Dr. SW, Suite 2001, Blacksburg, VA 24061
Phone: 540-231-3777, Fax: 540-231-3362, E-mail: binoy@vt.edu

Abstract

Cloud computing's ubiquitous infrastructure-as-a-service (IaaS) model has enabled a broad range of enterprise organizations to roll out online services at low cost. Cloud's multi-region support allows computational resources to be instantiated from different data centers around the world, enabling new classes of geographical-scale (or "geo-scale") applications.

State machine replication (SMR) is the de facto standard for building highly scalable and available distributed applications and services. SMR replicates a service across a set of nodes, and executes client operations on the replicas in an agreed-upon total order, ensuring consistency of the replicated state. The problem of determining a total order reduces to one of *consensus*.

State-of-the-art consensus protocols are inadequate for newer classes of applications such as blockchains and for geo-scale infrastructures. To overcome these limitations, the project has developed a family of leaderless consensus protocols for a variety of fault models and geographical deployment conditions. The project's key achievements are:

- Dester: *Leaderless Hybrid Fault-tolerant Consensus.*
- ezBFT: *Fast, Leaderless Byzantine Fault-Tolerant (BFT) Consensus.*
- Spectrum: *Contention-agnostic Consensus Framework*
- Caesar: *Fast, Leaderless Crash Fault-Tolerant (CFT) Consensus*
- Alvin: *Leaderless Crash Fault-Tolerant (CFT) Transactional System*

Dester is a leaderless hybrid state machine replication protocol that incorporates a novel trusted subsystem, called TruDep, for achieving high performance in geo-scale deployments. Dester allows any replica to propose and commit client commands in two communication steps in most practical situations, while clients minimize latency by sending commands to the closest replica.

ezBFT solves leaderless BFT consensus, in which an operation is executed on all replicas in three communication steps under low contention. The client sends the request to the geographically-closest replica who proposes an initial execution order for that request along with its dependencies. ezBFT achieves this by performing speculative execution of submitted requests after the execution of their dependencies, which are requests that must be executed in the same order across all replicas to ensure consistency. When the client observes inconsistencies during speculative execution, it enforces the right execution order, which the replicas execute.

Spectrum is a novel consensus framework that enables a consensus-based system to become contention-agnostic by providing the ability to switch between different consensus protocols at runtime in a way completely oblivious to the users. Spectrum is best fit for use cases where there is a need to adapt protocols to serve workloads with varying conflicts over time. Spectrum can provide the best possible performance under any contentious workload generated by applications, thus giving the user a perception that the system is actually contention-agnostic.

Caesar is a consensus protocol designed for geo-scale deployments that is able to maintain high performance in the presence of both mostly non-conflicting workloads (named as such if less than 5% of conflicting commands are issued) and conflicting workloads (where at most 40% of commands conflict with each other). Caesar can agree on the order for an operation in one RTT, called *fast decision*, when the workload is non-conflicting. Under conflicting workloads, Caesar maximizes the number of fast decisions using a novel *wait condition*.

Alvin is a geo-replicated transactional system that finds an effective tradeoff between performance and strong consistency. At the core of Alvin is a novel Partial Order Broadcast protocol (POB) that globally orders only conflicting transactions and minimizes the number of communication steps for non-conflicting transactions. Alvin's prototype is designed to be generic and customizable. It provides an API for its reliable ordering layer as well as an API for executing high performance distributed transactions.

The outcomes of the project has been published at various peer-reviewed conferences and journals, and has developed many Master's and PhD theses.

I. Overview of Project Achievements

The last decade has witnessed an ever increasing proliferation of online services enabled by ubiquitous cloud platforms. Cloud providers charge for per unit of computation resource utilized, thus minimizing the total cost of ownership of cloud resources. This has enabled businesses ranging from startups to large-scale enterprises to take advantage of cloud's flexibility to develop highly available (i.e., round-the-clock) applications and services. Moreover, cloud's multi-region support allows computational resources to be instantiated from different data centers around the world, enabling new classes of geographical-scale ("geo-scale") applications [1], [2].

Historically, developers have relied on *replication* to build highly scalable and available services, because a single host cannot handle the many millions of client requests issued per second. Moreover, distributed systems, in general, are prone to faults such as machine crashes, network outages, and software bugs. Replication has been perhaps the most viable solution to increase scalability and availability of applications for many decades. Developers have used replication to build mission-critical services such as databases [3], key-value stores and coordination systems [4].

State Machine Replication (SMR) is a common technique employed in today's distributed applications to tolerate server failures and maintain high availability [5]. The replication servers, or *replicas*, employ consensus (or agreement) protocols (e.g., Raft [6], Paxos [7]) to replicate the application state and ensure that the same sequence of client commands is executed on the shared state in the same order (i.e., a total order), ensuring consistency of the replicated state.

Consensus solutions can be broadly classified as Crash Fault-Tolerant (CFT) and Byzantine Fault-Tolerant (BFT), with the former being a subset of the latter. While CFT protocols have found their niche in datacenter applications [8], [3] of a single enterprise-class organization, BFT protocols are increasingly used in applications involving multiple independent organizations. For example, for distributed *ledger* or blockchain-based applications in the *permissioned* setting, consensus need to be reached among a set of known participants (e.g., independent organizations), despite no complete trust among them. In such cases, BFT protocols can ensure replica state consistency while withstanding malicious actors and other non-crash related faults. An example of a *permissioned* blockchain system that uses a BFT protocol is the Hyperledger Fabric [9].

Between the BFT and the CFT models, there exists a fault model, called the hybrid fault tolerance model, enables construction of byzantine fault tolerant protocols with replication resources similar to CFT protocols – namely $2f + 1$ nodes – by incorporate a small trusted subsystem that can only fail by crashing. Though some early solutions used ASICs and FPGAs resulting in very low performance [10], [11], the advent of trusted execution environments (TEEs) within commodity hardware such as Intel SGX [12] and ARM Trustzone [13] significantly reduces the performance overheads for hybrid protocols [14]. A software-based subsystem can be protected by such trusted environments providing the required level of trust.

Many consensus-based distributed systems are increasingly deployed in geographically distributed settings to cater to different application needs. Geographical-scale (or "geo-scale") deployments of such systems have an additional challenge: achieving low client-side latencies and high server-side throughput under the high communication latencies of a WAN. Since replicas need to communicate with each other and the clients to reach consensus, the number of communication steps incurred directly impacts the latency, as each step involves sending messages to potentially distant nodes. State-of-the-art protocols [15], [16] use various techniques to reduce communication steps, but they do so only in certain cases, often leading to poor performance in many plausible situations. Furthermore, no existing BFT consensus protocol reduce client-side latencies in a geo-scale setting, where, the latency per communication step is as important as the number of communication steps. In other words, a protocol can achieve significant cost savings if the latency incurred during a communication step can be reduced.

The downside of such lack of optimization is most manifest for single-leader or *primary* -based consensus protocols such as Multi-Paxos [7], PBFT [17], and Zyzzyva [15]: a replica is bestowed the *primary* status and is responsible for proposing the total-order for all client requests. While the clients that are geographically nearest to the primary may observe optimal client-side latency, the same is not true for distant clients. A distant client will incur higher latency for the first communication step (of sending the request to the primary). Additionally, the primary-based design limits throughput as the primary carries significantly higher load.

A leaderless protocol can solve the aforementioned problems. A client can send its requests to the nearest replica and can continue to do so as long as the replica is *correct*. The replica can undertake the task of finding an order among all the concurrent requests in the system, executing the request on the shared state, and return the result. Leaderless protocols [18], [19] have been previously proposed for the CFT model. However, to the best of our knowledge, such investigations have not been made for the BFT model.

The project's achievements include the design and development of a family of leaderless consensus protocols for a variety of fault models and geographical deployment conditions. To summarize, we developed leaderless consensus protocols for three fault models: Dester in the Hybrid Fault Tolerance model, ezBFT in the Byzantine Fault Tolerance model and Caesar and Alvin in the Crash Fault Tolerance model. Furthermore, the project also developed Spectrum, a contention framework that acknowledges that there is no one-size-fits-all consensus solution and provides a mechanism to switch consensus protocols at runtime to adapt to workloads with changing contention over time. The rest of this section summarizes the key findings of the project.

A. Dester

One of the outcomes of the project is the design, implementation, and evaluation of Dester, a leaderless hybrid fault-tolerant state machine replication protocol built *ground-up* for achieving high performance in the geo-scale environments. The leaderless nature allows every replica to process client commands by only relying on a set of closest replicas, thus providing low client-side latencies and high system throughput. Dester is, in part, made possible by a novel trusted subsystem called TruDep that provides the necessary trust required to ensure secure, leaderless operation. TruDep was designed specifically for trusted execution environments (such as Intel SGX) with a goal of having as few lines of trusted code as possible, such that the possibility of bugs in the subsystem tends to zero.

We implemented Dester and evaluated it against state-of-the-art systems including Hybster and PBFT. For geo-scale deployment, we leverage SGX-capable virtual machine instances available in the Azure cloud platform [20]. The systems were deployed on three each located in different geographical region. Our trusted subsystem TruDep and Hybster's trusted counter TrInx were implemented in C using Intel SGX SDK and interfaced with the Go application.

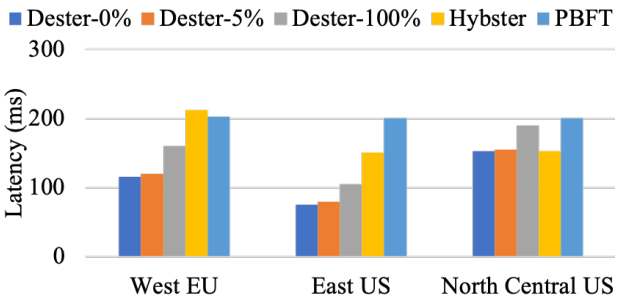

Fig. 1: Average latencies for Experiment 1. All primaries are in North Central US region. The latency is shown per region as recorded by the clients in that region.

One of the experiments to understand Dester's effectiveness in achieving optimal latency at each geographical region is presented in Figure 1. It shows the average latency (in milliseconds) experienced by clients located at

each region (shown on x-axis) for each of the protocols. For this experiment, we co-located a client at each node that sends requests to the replicas. For leader-based protocols (Hybster and PBFT), the leader was set to East US replica; thus, clients in other regions send their requests to the leader. For Dester, the client sends its requests to the nearest replica (which is in the same region). The clients send requests in closed-loop, meaning that a client will wait for a reply to its previous request before sending another one.

For Dester, the latency was measured at different contention levels: 0%, 5%, and 100%; the suffix in the legend indicates the contention. Among leader-based protocols, PBFT suffers the highest latency, by contacting an additional faraway replica and taking three communications steps. Hybster only provides optimal latency at leader-site. Dester in contrast is able to provide optimal latencies at every site, thus achieving huge latency savings of up to 50%. Dester's worst case behavior is evident when contention is 100%, which is an extreme case unlikely in practice. Regardless, Dester performs better than PBFT.

In summary, our experimental evaluations reveal that Dester provides up to 50% lower latency and 30% more throughput than Hybster in a geo-replicated setting. Complete details of Dester, including design and implementation details, theoretical results, experimental evaluation results, and full source codes are available at: http://www.hyflow.org.

B. EZBFT

Another outcome of the project includes the design, implementation, and evaluation of the EZBFT leaderless Byzantine Fault Tolerant (BFT) consensus protocol. EZBFT enables every replica in the system to process the requests received from the clients. Doing so (i) significantly reduces the client-side latency, (ii) distributes the load across replicas, and (iii) tolerates faults more effectively. EZBFT is the first BFT protocol to provide decentralized, deterministic consensus in the eventually synchronous model. It delivers requests in *three* communication steps in normal operating conditions. By minimizing the latency at each communication step, EZBFT provides a highly effective BFT solution for geo-scale deployments.

To enable leaderless operation, EZBFT exploits a particular characteristic of client commands: *interference*. In the absence of concurrent interfering commands, EZBFT's clients receive a reply in an optimal three communication steps. When commands interfere, both clients and replicas coherently communicate to establish a consistent total-order, consuming an additional zero or two communication steps. EZBFT employs additional techniques such as client-side validation of replica messages and speculative execution to reduce communication steps in the common case.

Figure 2 presents the results of an experiment that shows EZBFT's effectiveness in achieving optimal latency at each geographical region. For this experiment, we deployed the protocols in four AWS regions and co-located a client that sends requests to the replica. For single primary-based protocols (PBFT, FaB, Zyzzyva), the primary was set to the replica in Virginia; thus, clients in other replicas send their requests to the primary. For EZBFT, the client sends its requests to the nearest replica (which is in the same region). The clients send requests in closed-loop, meaning that a client will wait for a reply to its previous request before sending another one.

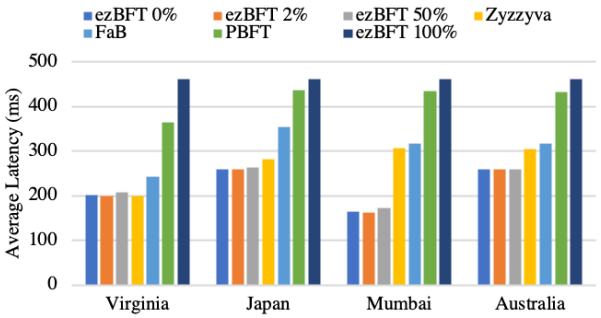

Fig. 2: Average latencies as recorded by the clients per region. All primaries are in Virginia.

To understand how EZBFT fares against state-of-the-art BFT protocols, we implemented EZBFT in Go as part of the Hyflow transactional/concurrency control middleware infrastructure

(www.hyflow.org), and conducted an experimental evaluation using a key-value store benchmark. The systems were deployed using the AWS EC2 infrastructure in different geographical regions.

Figure 2 shows the average latency (in milliseconds) observed by the clients located in their respective regions (shown on x-axis) for each of the four protocols. For ezBFT, the latency was measured at different contention levels: 0%, 2%, 50%, and 100%; the suffix in the legend indicates the contention. Among primary-based protocols, PBFT suffers the highest latency, because it takes five communication steps to deliver a request. FaB performs better than PBFT with four communication steps, but Zyzzyva performs the best among primary-based protocols using only three communication steps. Overall, ezBFT performs as good as or better than Zyzzyva, for up to 50% contention. In the Virginia region, both Zyzzyva and ezBFT have about the same latency because they have the same number of communication steps and their primaries are located in the same region. However, for the remaining regions, Zyzzva clients must forward their requests to Virginia, while ezBFT clients simply send their requests to their local replica, which orders them.

In summary, our evaluation reveals that ezBFT improves client-side latency by as much as 40% over PBFT, FaB, and Zyzzyva. Complete details of ezBFT, including design and implementation details, theoretical results, experimental evaluation results, and full source codes are available at: http://www.hyflow.org.

C. Spectrum

The project's achievements include the design, implementation, and evaluation of Spectrum, a novel consensus framework that enables a consensus system to become contention-agnostic by providing the ability to switch between different consensus protocols at runtime in a way completely oblivious to the users. Spectrum is best fit for use cases where there is a need to adapt protocols to serve workloads with varying conflicts over time. Spectrum can provide the best possible performance under any contentious workload generated by applications, thus giving the user a perception that the system is actually contention-agnostic.

Its main features include: *i)* A switching mechanism that is able to coordinate among supported consensus protocols to enable an oblivious transition, and capable of reaching a non-blocking agreement among nodes on the specific switch to be performed. *ii)* Specific methodologies, each specialized for the switch from a consensus protocol to another one. These include switching from single leader-based consensus to leaderless consensus, and vice versa. *iii)* Common interfaces for existing and future consensus protocols to enable the usage of those protocols as plugins in the framework, and a simple oracle to detect and react to a change of the workload.

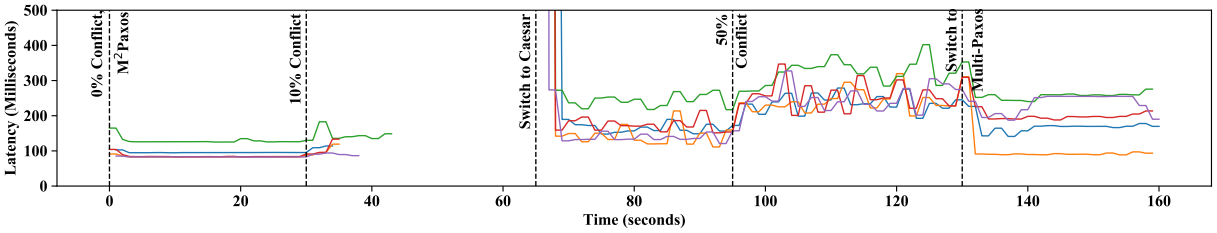

Fig. 3: Effectiveness of Spectrum in providing minimum possible latency for any conflicting workload.

Spectrum framework was implemented in Java as part of the Hyflow transactional/concurrency control middleware infrastructure (http://www.hyflow.org). The framework incorporated the following consensus protocols: Multi-Paxos, Caesar, and M^2Paxos. These three protocols cover

the majority of the conflict spectrum and allow to demonstrate the ability of the framework to cope with various amounts of conflicting workloads.

To demonstrate the effectiveness of the approach, we conducted experiments by deploying the framework in five Amazon AWS data center sites around the world using the AWS EC2 service. One experiment, as shown in Figure 3, is meant to show that Spectrum's ability to switch at runtime enables to provide the minimum possible user-perceived latency for any conflicting workload. The experiment was conducted by increasing the amount of conflicts over time in order to provoke a switch. The x-axis presents the time in seconds since the beginning of the experiment, and the y-axis shows user-perceived latency in milliseconds.

Spectrum was initialized with M^2PAXOS as the starting consensus protocol and a running non-conflicting workload. In this case, the oracle does not trigger any switch as M^2PAXOS is the best protocol for this workload. At time $t = 30s$, the amount of conflicts in the workload is increased to 10%. At this point, M^2PAXOS experiences a livelock caused by conflicting ownership acquisitions, and thus the client requests timeout. The oracle steps in at $t = 65s$ and switches to CAESAR, and in few seconds, Spectrum responds and delivers commands to the client. At $t = 95s$, the conflict was increased to 50%, and CAESAR starts performing poorly, but not as worse as M^2PAXOS at 10% conflict. The oracle triggers the switch to Multi-Paxos at $t = 130s$, and this reduces the latency as at this amount of conflict, a leader-based protocol is better than any other group of protocols.

Complete details of Spectrum, including design and implementation details, theoretical results, experimental evaluation results, and full source codes are available at: http://www.hyflow.org.

D. Caesar

We developed CAESAR, a Consensus protocol designed for geo-scale deployments that is able to maintain high performance in the presence of both mostly non-conflicting workloads (named as such if less than 5% of conflicting commands are issued) and conflicting workloads (where at most 40% of commands conflict with each other).

CAESAR solves consensus by taking into account the various scenarios that occur when messages are delivered asynchronously. CAESAR inherits the advantages of existing approaches – Mencius [5] and EPaxos [16] – and overcomes the inherent pitfalls in these existing approaches. In summary, CAESAR can agree on the order for an operation in one RTT, called *fast decision*, when the workload is non-conflicting. Under conflicting workloads, CAESAR maximizes the number of fast decisions using a novel *wait condition*. Under scenarios where a fast decision is not possible despite the presence of the *wait condition*, one more RTT is taken to decide on the order for the operation, and this is called *slow decision*.

CAESAR was implemented in Java as part of the Hyflow transactional/concurrency control middleware infrastructure (http://www.hyflow.org), and evaluated using the key-value store benchmark. Using the key-value store interfaces, different workloads were injected by varying the percentage of conflicting commands, and various performance parameters were measured.

CAESAR was contrasted against state-of-the-art Consensus protocols including EPaxos [16], M^2Paxos [21], Mencius [5]; and Multi-Paxos [22]. For evaluation, the systems were deployed using the Amazon EC2 infrastructure in five geographical regions around the world.

In Figure 4, we report the average latency incurred in each site by CAESAR, EPaxos, and M^2Paxos to order and execute a command. Each cluster of data shows the behavior of a system while increasing the percentage of conflicts in the range of {0% – no conflict, 2%, 10%, 30%, 50%, 100%}. At 0% of conflicts, EPaxos and M^2Paxos provide comparable performance because both employ two communication steps to order commands and the same size for quorums, with EPaxos slightly faster because it does not need to acquire the ownership on submitted commands

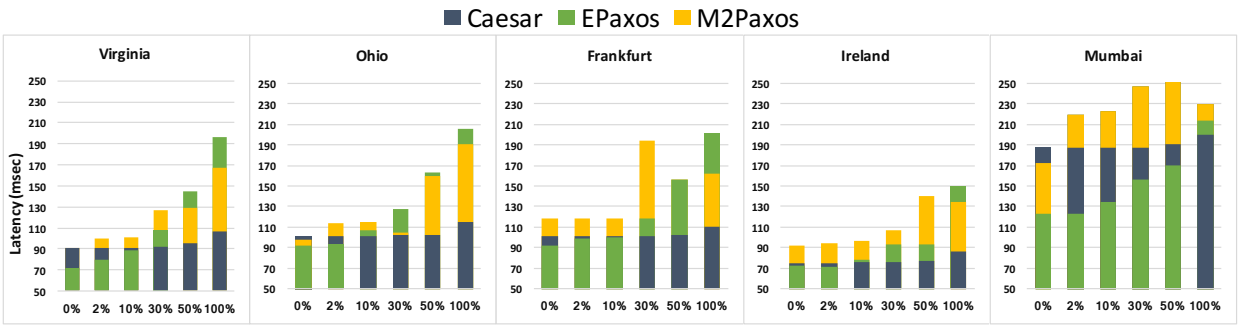

Fig. 4: Average latency for ordering and processing commands by changing the percentage of conflicting commands. Batching disabled. Given a percentage value, bars of competitors are overlapped: e.g., in the case of 30% conflicting commands and the node in Virginia, latency values are 90 msec, 108 msec, and 127 msec, for CAESAR, EPaxos, and M^2Paxos, respectively.

before ordering. The performance of CAESAR is slightly slower than EPaxos because of the need to contact one additional node to reach consensus. However, this overhead is on average 18%.

When the percentage of conflicting commands increases up to 50%, CAESAR sustains its performance by providing an almost constant latency; all other competitors degrade their performance visibly. The latency provided by the node in India is higher than other nodes. Here, CAESAR is 50% slower than EPaxos only when conflicts are low, because CAESAR has to contact one more faraway node (e.g., Virginia) to deliver fast.

Complete details of CAESAR, including design and implementation details, theoretical results, experimental evaluation results, and full source codes are available at: http://www.hyflow.org.

E. Alvin: High-Performance Geo-Replicated Transactional System

Geo-replicated concurrency control protocols can be classified under two approaches. The first approach ensures high consistency, but restricts the type of transactions that are allowed [23], [16]. This enables exploiting specific protocol optimizations to achieve high performance. The second approach allows general-purpose transactions, but weakens the consistency criterion for better performance [24]. This has the negative effect of reduced programmability, as programmers must cope with potential inconsistent states in application behaviors.

Motivated by this gap between strong consistency/poor performance and weak consistency/-good performance, we developed a geo-replicated transactional system called Alvin [25], which finds an effective tradeoff between performance and strong consistency. At the core of Alvin lies a novel Partial Order Broadcast protocol (*POB*) that globally orders only conflicting transactions and minimizes the number of communication steps for non-conflicting transactions.

While the idea of defining the agreement of consensus on the basis of message semantics is not new and has been previously introduced in Generalized Consensus [26] and Generic Broadcast [27], POB encompasses a novel approach for ordering transactions' commits that overcomes the limitations of existing single leader-based solutions (i.e., Generalized Paxos [26]) when deployed geographically. POB does not rely on a designated leader to either order transactions or support conflict resolution in case of conflicting concurrent transactions.

POB has been designed to inherit the benefits of state-of-the-art, multi-leader, state machine replication protocols specifically proposed for GDS such as *Mencius* [5] and *EPaxos* [16], and, at the same time, to overcome their limitations. In particular, POB, like Mencius [5], has the advantages of defining the final order of messages on the sender nodes. Typically, this technique avoids expensive distributed decisions by determining an a priori assignment of delivered positions

to messages. This approach suffers from potentially expensive waiting conditions that are needed to ensure that the delivery of a message in position p does not precede the delivery of a message in position $p' < p$. However, POB, unlike Mencius, relies on a quorum of replies, instead of waiting for the information about delivered positions from all nodes. This makes POB's performance robust even in scenarios where nodes are far apart (as is often the case in GDS), or when the message sending rate is unbalanced among nodes.

Furthermore, Alvin exports design choices to programmers to customize the POB and P-CC according to the needs of the application and system at hand. As an example, Alvin offers two strong consistency criteria that programmers can select: Serializability (SR) [28], which requires all transactions including read-only ones to be broadcast via POB; and Extended Update Serializability (EUS) [29], [30] (i.e., PL-3U [29]), which allows read-only transactions to execute locally at the cost of generating some non-serializable schedules that are usually silent to the application.

We built the Alvin prototype in the *Go* programming language and evaluated it on the Amazon EC2 infrastructure using up to 7 geographically distributed sites with benchmarks including Bank [31] and TPC-C [32]. As competitors, we implemented two certification-based transactional systems [33] that rely on MultiPaxos [7] and EPaxos [16] for their ordering layer. Our experiments reveal that, when configured to exploit EUS, Alvin provides significant speedup for seven-datacenter with TPC-C workloads by as much as 4.8× compared to EPaxos. This significant gain is due to a more efficient execution of read-only workload, which is enabled by EUS's semantics. If Alvin runs under SR, it gains up to 26% over EPaxos because it does not pay the cost of graph analysis needed by EPaxos for delivering transactions. On Bank, due to its small transactions and trivial dependency graphs, that cost is not significant, thus EPaxos behaves similarly to Alvin. MultiPaxos highlights the drawbacks of having a single leader in GDS, thus its performance is lower than other (multi-leader) competitors.

Complete details of Alvin, including design and implementation details, theoretical results, experimental evaluation results, and full source codes are available at: http://www.hyflow.org.

II. Summary of Project's Software and Publications[1]

i) "Achieving Decentralization in Hybrid Fault Tolerance," B. Arun, and B. Ravindran, *The 40th IEEE International Conference on Distributed Computing Systems (ICDCS 2020)*, July 2020, Singapore. Under Review.

ii) "Taming the Contention in Consensus-based Distributed Systems," B. Arun, S. Peluso, R. Palmieri, G. Losa, and B. Ravindran, *IEEE Transactions On Dependable and Secure Computing. Minor Revision* Under Review.

iii) "Generalized Consensus for Practical Fault Tolerance ," M. Garg, S. Peluso, B. Arun, and B. Ravindran, *The 20th ACM/IFIP/USENIX International Middleware Conference*, December 2019, Davis, CA, USA.

iv) "ezBFT: Decentralizing Byzantine Fault-Tolerant State Machine Replication," B. Arun, S. Peluso, and B. Ravindran, *The 39th IEEE International Conference on Distributed Computing Systems (ICDCS 2019)*, July 2019, Dallas, Texas, USA.

v) "Speeding up Consensus by Chasing Fast Decisions," B. Arun, S. Peluso, R. Palmieri, G. Losa, and B. Ravindran, *The 47th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2017)*, June 2017, Denver, CO, USA, To appear.

vi) "A Low-latency Consensus Algorithm for Geographically Distributed Systems," Balaji Arun, *MS Thesis*, Virginia Tech, February 2017.

[1]Publications reported here include papers and theses published, accepted, and in progress for the reporting period. All publications are available at http://www.hyflow.org.

vii) "Optimizing Distributed Transactions: Speculative Client Execution, Certified Serializability, and High Performance Run-Time," Utkarsh Pandey, *MS Thesis*, Virginia Tech, August 2016.

viii) "Brief Announcement: A Family of Leaderless Generalized-Consensus Algorithms," G. Losa, S. Peluso, and B. Ravindran, *The 35th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC 2016)*, July 25-28, 2016, Chicago, Illinois, USA.

ix) "Exploiting Parallelism of Distributed Nested Transactions," D. Niles, R. Palmieri, and B. Ravindran, *The 9th ACM International Systems and Storage Conference (SYSTOR 2016)*, June 6-8, 2016, Haifa, Israel.

x) "Making Fast Consensus Generally Faster," S. Peluso, A. Turcu, R. Palmieri, G. Losa, and B. Ravindran. The *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2016)*, June 28 – July 1st, 2016, Toulouse, France.

xi) "*Be General and Don't Give Up Consistency in Geo-Replicated Transactional Systems*", A. Turcu, S. Peluso, R. Palmieri, B. Ravindran, In proceeding of the *18th International Conference on Principles of Distributed Systems* (OPODIS), December 15-19, 2014, Cortina, Italy.

xii) "*Improving Performance of Highly-Programmable Concurrent Applications by Leveraging Parallel Nesting and Weaker Isolation Levels*", Duane F. Niles, Jr., MS Thesis, June 2015.
URL: https://vtechworks.lib.vt.edu/bitstream/handle/10919/54557/Niles_DF_T_2015.pdf

xiii) "*On the Fault-tolerance and High Performance of Replicated Transactional Systems*", Sachin Hirve, PhD Dissertation, September 2015.
URL: http://vtechworks.lib.vt.edu/bitstream/handle/10919/56668/Hirve_S_D_2015.pdf

xiv) "*On Improving Distributed Transactional Memory through Nesting, Partitioning and Ordering*", Alexandru Turcu, PhD Dissertation, January 2015.
URL: https://vtechworks.lib.vt.edu/bitstream/handle/10919/51593/Turcu_A_D_2015.pdf

xv) "*Exploiting Parallelism of Distributed Nested Transactions*", D. Niles, R. Palmieri, B. Ravindran, Under review at the *9th ACM International Systems and Storage Conference* (SYSTOR), June 6 – 8, 2016, Haifa, Israel.

References

[1] "Cloud Spanner - Google Cloud," https://cloud.google.com/spanner/, accessed: 2018-11-06.

[2] "CosmosDB - Azure," https://azure.microsoft.com/en-us/services/cosmos-db/, accessed: 2018-11-06.

[3] Cockroach Labs, "CockroachDB," 2017.

[4] CoreOS, "Etcd: Distributed reliable key-value store," 2017.

[5] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building Efficient Replicated State Machines for WANs," in *USENIX OSDI*, 2008.

[6] D. Ongaro and J. Ousterhout, "In Search of an Understandable Consensus Algorithm," in *USENIX ATC*, 2014.

[7] L. Lamport, "Paxos made simple," *ACM SIGACT News (Distributed Computing Column)*, vol. 32, no. 4, pp. 51–58, December 2001.

[8] J. Baker et al., "Megastore: Providing scalable, highly available storage for interactive services," in *CIDR*, 2011.

[9] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukoli?, S. W. Cocco, and J. Yellick, "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18. ACM, pp. 30:1–30:15.

[10] D. Levin, J. R. Douceur, J. R. Lorch, and T. Moscibroda, "Trinc: Small trusted hardware for large distributed systems," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, ser. NSDI'09. USENIX Association, 2009, pp. 1–14.

[11] R. Kapitza, J. Behl, C. Cachin, T. Distler, S. Kuhnle, S. V. Mohammadi, W. Schröder-Preikschat, and K. Stengel, "Cheapbft: Resource-efficient byzantine fault tolerance," in *Proceedings of the 7th ACM European Conference on Computer Systems*, ser. EuroSys '12. ACM, 2012, pp. 295–308.

[12] V. Costan and S. Devadas, "Intel SGX explained." vol. 2016, no. 086, pp. 1–118, 2016.

[13] A. Limited., "ARM security technology: building a secure system using TrustZone technology," http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, ARM Technical White Paper, Accessed: 2018-11-06.

[14] J. Behl, T. Distler, and R. Kapitza, "Hybrids on steroids: SGX-based high performance BFT," in *Proceedings of the Twelfth European Conference on Computer Systems*, ser. EuroSys '17. ACM, 2017, pp. 222–237.
[15] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: Speculative byzantine fault tolerance," pp. 45–58, 2007. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294267
[16] I. Moraru, D. G. Andersen, and M. Kaminsky, "There is More Consensus in Egalitarian Parliaments," in *ACM SOSP*, 2013.
[17] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002. [Online]. Available: http://doi.acm.org/10.1145/571637.571640
[18] B. Arun, S. Peluso, R. Palmieri, G. Losa, and B. Ravindran, "Speeding up Consensus by Chasing Fast Decisions," in *IEEE/IFIP DSN*, 2017.
[19] S. Peluso, A. Turcu, R. Palmieri, G. Losa, and B. Ravindran, "Making fast consensus generally faster," in *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2016, pp. 156–167.
[20] R. J. Dudley and N. Duchene, *Microsoft Azure: Enterprise Application Development*. Packt Publishing, 2010.
[21] S. Peluso, A. Turcu, R. Palmieri, G. Losa, and B. Ravindran, "Making Fast Consensus Generally Faster," in *IEEE/IFIP DSN*, 2016.
[22] L. Lamport, "The part-time parliament," *ACM Trans. Comput. Syst.*, pp. 133–169, 1998.
[23] Y. Zhang, R. Power, S. Zhou, Y. Sovran, M. K. Aguilera, and J. Li, "Transaction chains: Achieving serializability with low latency in geo-distributed storage systems," ser. SOSP, 2013.
[24] S. Almeida, J. Leitão, and L. Rodrigues, "Chainreaction: a causal+ consistent datastore based on chain replication," in *EuroSys 2013*.
[25] A. Turcu, S. Peluso, R. Palmieri, and B. Ravindran, "Be General and Don't Give Up Consistency in Geo-Replicated Transactional Systems," in *OPODIS*, 2014.
[26] L. Lamport, "Generalized Consensus and Paxos," Microsoft Research, Tech. Rep. MSR-TR-2005-33, March 2005.
[27] F. Pedone and A. Schiper, "Generic broadcast," in *Distributed Computing*, ser. Lecture Notes in Computer Science, P. Jayanti, Ed. Springer Berlin Heidelberg, 1999, vol. 1693, pp. 94–106. [Online]. Available: http://dx.doi.org/10.1007/3-540-48169-9_7
[28] P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987.
[29] A. Adya, "Weak consistency: A generalized theory and optimistic implementations for distributed transactions," Ph.D. dissertation, 1999, aAI0800775.
[30] S. Peluso, P. Ruivo, P. Romano, F. Quaglia, and L. Rodrigues, "When scalability meets consistency: Genuine multiversion update-serializable partial data replication," in *Proceedings of the 2012 IEEE 32Nd International Conference on Distributed Computing Systems*, ser. ICDCS '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 455–465. [Online]. Available: http://dx.doi.org/10.1109/ICDCS.2012.55
[31] S. Hirve, R. Palmieri, and B. Ravindran, "Archie: A speculative replicated transactional system," in *Proceedings of the 15th International Middleware Conference*, ser. Middleware '14. New York, NY, USA: ACM, 2014, pp. 265–276. [Online]. Available: http://doi.acm.org/10.1145/2663165.2666090
[32] T. P. P. Council, "TPC-C benchmark," 2010, http://www.tpc.org/tpcc/.
[33] F. Pedone, R. Guerraoui, and A. Schiper, "The database state machine approach," *Distrib. Parallel Databases*, vol. 14, pp. 71–98, July 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=640475.640518