Chemistry behind Agreement

Suyash Gupta
University of California, Berkeley
suyash.gupta@berkeley.edu

Mohammad Javad Amiri
University of Pennsylvania
mjamiri@seas.upenn.edu

Mohammad Sadoghi
University of California, Davis
msadoghi@ucdavis.edu

ABSTRACT

Agreement protocols have been extensively used by distributed data management systems to provide robustness and high availability. The broad spectrum of design dimensions, applications, and fault models have resulted in different flavors of agreement protocols. This proliferation of agreement protocols has made it hard to argue their correctness and has unintentionally created a disparity in understanding their design. To address this disparity, we study the chemistry behind agreement and present a unified framework that simplifies expressing different agreement protocols. Specifically, we extract essential elements of the agreement and define atoms that connect these elements. We illustrate how these elements can help to explain and design various agreement protocols.

1 INTRODUCTION

In the past five decades, the database and systems community has introduced a large suite of *agreement protocols* [9, 18, 43, 83, 86]. The key aim behind all of these protocols remains the same–how to make a set of nodes agree on a value. Despite this, several flavors of these protocols exist in the wild: commit protocols [37, 69, 95, 96], Crash Fault-Tolerant (CFT) protocols [4, 21, 25, 26, 33, 54, 54, 56, 64–68, 70, 71, 78, 81, 82, 85, 87, 103] protocols, Byzantine Fault-Tolerant (BFT) protocols [1, 6, 22, 24, 36, 38, 40, 42, 57, 58, 60, 75, 80, 104], and hybrid fault-tolerant protocols [13, 28, 38, 59, 72, 76, 88, 92, 94, 100]. But that begs the question: why are there so many protocols?

A key reason for designing various protocols was the application environment. For instance, the Two-Phase Commit (2PC) [37] and Three-Phase Commit (3PC) [96] protocols were designed to ensure that all the nodes agreed on a client transaction's fate (commit or abort). However, at the time of inception of these protocols, failures were rare, and a majority of failures were simple node failures that did not cause data loss. This led to the application of these protocols in sharded settings where each node holds a unique set of data-items. These protocols are still employed for agreement among on-premise servers that are strictly administered.

With data-failures becoming prevalent, the research community came up with the design of replicated systems where several copies (replicas) of the data are maintained [83]. In such a setting, CFT protocols help to achieve agreement among the replicas. Similarly, the rise of multi-party computations led to the introduction of malicious attacks on database replicas. To resolve these attacks, the research community proposed the design of BFT protocols [43]. With the introduction of cloud computing, several variants of these protocols are under use as an organization could have its data spread across clouds in a sharded-replicated architecture.

Alarmingly, although these protocols extend each other, the trivial disparities in their design have created distinct communities working around these protocols. This has led to the use of incompatible algorithmic designs, schematic representations, and proof systems for each style of protocols. For example, state diagram representation is often used to explain or design commit protocols, while fault-tolerant protocols use line diagram representations. Similarly, there have been attempts to create automatic proof systems for all but BFT protocols, which are still proven correct using pen-and-paper proofs. We believe this *disparity hinders innovation*. This disparity also impacts two recent design philosophies: (i) AI-powered distributed database management and analysis, and (ii) Sky Computing, which demands unified abstractions that can support communication and collaboration across multiple clouds [27, 99].

To this end, we analyze the design of existing agreement protocols and present a unified model that can intuitively explain different flavors of agreement. Specifically, in this work, we explore the *chemistry behind the agreement* and extract *elements* at the core of any agreement protocol. We visualize each agreement protocol as a compound formed from bonding these elements together. As a result, we also characterize *atoms* that help to bond different elements. Through this exercise, we do not want to re-design the existing agreement protocols but rather present an intuitive way of thinking about agreement protocols. We believe that our chemical representation of agreement protocols can ease the analysis of existing protocols and stimulate the growth of newer protocols.

Akin to chemistry, where the periodic table of elements has helped scientists to come up with newer compounds, we believe our elemental representation of agreement protocols can help reveal newer designs. Further, over the past decades, new elements have been added to the periodic table. We acknowledge this fact and recognize that the list of elements we present in this paper is in no way exhaustive, and newer elements may be added in the future.

This exercise of coming up with a chemical representation of agreement protocols has also revealed several interesting insights. Some of these we enlist next: (i) Only *three* elements are needed to construct a CFT protocol like Paxos [63]. (ii) 2PC and Paxos protocols share common elements, but differ in the way transactions are ordered. (iii) Similarly, 3PC and PBFT [24] (the premier BFT consensus) protocol have common elements. (iv) Majority of protocols require a subset of *four* elements to recover from failures.

To illustrate the practical use case of our chemical representation, we implement a modular architecture on top of the Bedrock [15] framework. Specifically, we implement each element as a module and permit their easy combination. This helps to experiment with different elements and design newer protocols. We used this architecture to implement *five* popular agreement protocols and present an initial experimental evaluation of the same.

 13th Annual Conference on Innovative Data Systems Research (CIDR '23). January 8-11, 2023, Amsterdam, The Netherlands.

2 AGREEMENT BUILDING BLOCKS

The goal of an agreement protocol is to establish a consensus among a set of nodes N on a proposal ρ. In this paper, we view agreement protocols as *compounds* constructed from bonding several atoms and elements together. As a result, next, we enlist the atoms and elements of agreement.

2.1 Atoms

An atom is the smallest indivisible unit of an element. In our chemical representation, we use this characteristic of the atoms to define functional properties of agreement that impact every element of the agreement protocol.

(1) **Failure.** Each agreement protocol states the behavior it expects from all the participating nodes. This behavior specifies how each node follows the protocol. Generally, an agreement protocol requires that a non-faulty node follows the protocol: it produces a deterministic output on a deterministic input.

Often, some nodes in the network act faulty. This could be due to simple crash failures or due to malicious (Byzantine) attacks. A node is assumed to be crashed if it operates at arbitrary speeds, fail-stops, or unexpectedly restarts. A Byzantine node, on the other hand, can collude, lie, or otherwise attempt to subvert the protocol.

(2) **Quorum.** Each agreement protocol reaches a final decision after consulting with all the nodes in the set N. Depending on the behavior expected from the nodes and the types of failure it can sustain, each agreement protocol states the minimum number of nodes required to make a decision. This minimum number of nodes necessary for decision-making is termed as a *quorum*; in our chemical representation, we represent quorum with notation Q.

We assume that there are $\mathbf{n}$ nodes in the set N and at most, $\mathbf{f}$ of them can fail. The protocol semantics defines the relation between $\mathbf{n}$ and $\mathbf{f}$. Traditional commit protocols like Two-Phase Commit (2PC) and Three-Phase Commit (3PC) expect a quorum size of $\mathbf{n} - 1$ as the leader needs to wait for the votes from all the participating nodes. CFT protocols like Paxos [63] and Mencius [74] require a quorum size of $\mathbf{f} + 1$ nodes; as a result, these protocols have $\mathbf{n} = 2\mathbf{f} + 1$ nodes. BFT protocols like PBFT [24], PoE [41], and HotStuff [34], on the other hand, expect at least $\mathbf{n} = 3\mathbf{f} + 1$ nodes as they require a larger quorum size of $2\mathbf{f} + 1$ nodes. Some protocols can tolerate a larger set of failures; Bosco [98] increases the network size from $5\mathbf{f} + 1$ to $7\mathbf{f} + 1$ to tolerate $2\mathbf{f}$ faulty nodes.

(3) **Topology.** Each communication protocol sticks to one or more topology for communicating between the nodes. This topology also helps to define the nature of each element. The most prevalent topology includes the star, clique, or ring.

In a star topology, the communication occurs in two steps: from a designated node (for example, the leader) to all other nodes and from all the nodes back to the designated node [34, 41]. Often, the star topology is also viewed as *centralized* communication as it is initiated by the designated node. In the clique topology, all (or a subset of) the nodes communicate with each other without any intermediaries [24]. Such a communication pattern is also referred to as *decentralized* communication. Some protocols also exhibit the ring (chain) topology, where each node sends messages to its successor in the ring and receives messages from its predecessor.

(4) **Data Distribution.** The design of any agreement protocol is based on how it expects the data to be distributed among its nodes. If each node in the network stores a unique set of data items, we term such a data distribution scheme as data sharding. If each node in the network stores all the data items, we term such a data distribution scheme as data replication. Some protocols offer a hybrid scheme of sharded-replication, where distinct groups of nodes manage distinct data items; within each group, the data is replicated among all the nodes. Depending on the data distribution scheme, we need to define new elements that can capture the characteristics of the underlying agreement protocol.

2.2 Elements

An element is composed of one or more atoms. In our chemical representation, an element represents the phases of an agreement protocol and it inherits the properties of this agreement protocol through different atoms. We present a general list of elements that are part of a majority of protocols. Note that this list is not meant to provide a fully exhaustive set of elements, but rather to demonstrate the overall methodology used to define elements. As new protocols are created, new elements may be found and added to this list. However, we envision each such new protocol employing some of these general elements.

(1) **Proposal (Pr).** Each agreement protocol aims to establish a consensus among the nodes in set N on a proposal ρ. As a result, the first element of our periodic table of agreement protocols is **Pr**, which signifies the start of an agreement. Each proposal typically includes information that identifies a client transaction, such as an identifier (ID), a sequence number or a ballot number, and an operation to be executed. As each ρ initiates an instance of the agreement protocol, without the loss of generality, we can refer to the proposer of a proposal as the leader. The leader l sends this proposal to all nodes for acceptance.

(2) **Vote (V).** The next step in any agreement protocol is to decide on the fate of the leader's proposal. To do this, each agreement protocol expects each node r, $r \in N$ to *vote* once it receives ρ from l. Often, r sends its vote to l (in star topology). In BFT protocols, up to $\mathbf{f}$ nodes (including the leader) may act maliciously. As a result, each honest r only casts a vote in the support of a *valid* proposal. However, malicious nodes are free to cast any vote. Considering the key role played by this phase, we assign it element **V**.

(3) **Prepare (Pp) and Commit (Co).** Based on the votes received from the nodes, the leader l is expected to conclude the *common* decision. As different agreement protocols set different expectations from the network, l may not receive support from all the nodes. Hence, l waits to only reach the quorum set by the protocol. Once the quorum is reached, the leader has a guarantee that a sufficient number of nodes have voted on this proposal.

If l wants to first *prepare* the nodes about this global decision, it broadcasts a *Prepare* message on the network. When the leader believes that the transaction can be committed at all the nodes, it broadcasts a *Commit* message on the network. Note that the decision to prepare or commit a transaction is not the leader's choice rather a step in the protocol. For this reason, we dedicate two distinct elements to represent the prepare (**Pp**) and commit (**Co**) phases.

This prepare (**Pp**) and commit (**Co**) stages follow the star topology where the leader (or a collector) is responsible for collecting votes and demonstrating that a quorum of nodes supports the proposal in a centralized manner. A common concern with such a design is that it relies on a specific node, which can suffer congestion or undergo failures. As a result, several protocols advocate the decentralized clique topology where each node broadcasts its vote on the network. This allows each node to reach a common decision without any intermediary. In this manner, two centralized (all-to-one and one-to-all) communication phases, e.g., vote and prepare, can be replaced with a decentralized (all-to-all) communication phase. To take into account this decentralized design, we add an $\oplus$ to the corresponding elements and show decentralized Prepare as $\mathbf{Pp}^{\oplus}$ and decentralized Commit as $\mathbf{Co}^{\oplus}$.

(4) **Execution (X).** A key task associated with agreement is the execution of client transactions. On the one hand, some protocols execute a transaction once its order is determined (*order-then-execute*) [24, 63]. On the other hand, some protocols reach agreement on the executed transaction (*execute-then-order*) [37, 96]. We consider execution as an element of the agreement protocol.

(5) **Checkpoint (Ch).** Periodically, nodes in N exchange their states to ensure they have a common state. This exchange phase also helps them to recover any missing committed transactions.

(6) **Leader Election (Le).** Often one or more nodes may fail, messages may get delayed or lost, or there could be adversarial attacks. If the nodes in N detect that the current leader has failed, then they may initiate a leader election phase. The newly elected leader is expected to help commit the current proposal and ensure that the nodes in N have a common state. Nodes may also participate in leader election phases if there is more than one proposer.

Some protocols have a deterministic leader election process where the next candidate is predesignated. Several other protocols, require each candidate to broadcast its proposal to all the nodes and nodes vote on their choice of candidate.

3 BONDING ELEMENTS INTO PROTOCOL

We now illustrate how existing state-of-the-art agreement protocols are compounds of our atoms (§2.1) and elements (§2.2). Specifically, we study the design of *four* famous protocols, which accelerated development of newer protocols.

Notations. We use *angle brackets* ($\langle\rangle$) to represent a sharded setting and *double vertical lines* ($\|\|\|$) to represent a replicated setting. To represent any phase that has clique communication topology, we add an $\oplus$ to the corresponding element. For transition to phases that require no communication between the nodes, we use the notation $\circ$. In most protocols, the leader proposes the order of execution for each transaction. However, in some protocols, each node individually decides whether it can consistently order a transaction; for such protocols, we add a $\ddagger$ at the state of decision making. In Figure 1, we use these notations to illustrate different agreement protocols; these flows represent the non-failure case.

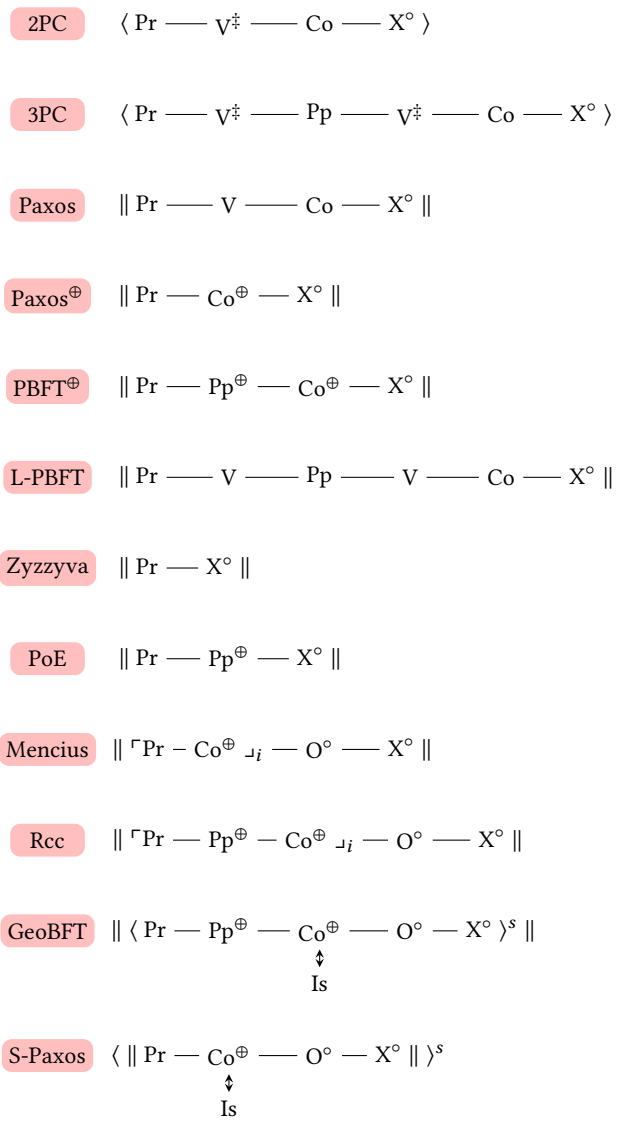

Figure 1: Chemical representation of the non-failure flow of different agreement protocols.

3.1 Two-Phase Commit (2PC).

The 2PC protocol is often referred to as the earliest agreement protocol, which helps a set of nodes, holding distinct items (data-sharding), decide whether they want to commit or abort a transaction [37, 97]. Even after 40 years of its inception, 2PC is still employed by systems like Google's Spanner [29]. The protocol works as follows: when the leader is ready to commit a transaction, it requests all the nodes to vote. After receiving all the votes, the leader broadcasts the global decision (commit or abort); post which nodes execute and commit the transaction.

The 2PC protocol can handle simple node failures: either the leader or the nodes can fail. Concurrent failures of the leader and one or more nodes can cause the protocol to *block*. If the leader fails, variants of 2PC permit leader election to ensure recovery.

3.2 Three-Phase Commit (3PC).

The blocking nature of 2PC protocol led to the design of the 3PC protocol [96]. The key intuition behind 3PC protocol is that the decision to commit a transaction requires an additional phase of agreement. Specifically, first the leader *prepares* the nodes about the common decision to commit (if

all the nodes voted to commit). The nodes acknowledge receipt of this common decision to the leader (can be visualized as re-sending the vote). Once the leader has received acknowledgments from all the nodes, it asks them to commit the transaction.

3.3 Paxos Consensus. Prior works have proved that under an unreliable network (messages can get lost or delayed) even the 3PC protocol can block [51, 63]. Further, failure of a single node in 2PC or 3PC before the leader receives its vote causes the transaction to abort. The reason for these problems is twofold: (i) only one node holds a copy of each data item, and (ii) the leader needs to wait for the votes of all the nodes. One way to eliminate these problems is to view the network as replicated; each node holds a copy of every data item. Paxos [63] is the most known protocol to guarantee agreement among the nodes in a replicated network.

Paxos guarantees agreement in a network of $\mathbf{n} = 2\mathbf{f} + 1$ replicas. We describe the Paxos protocol with a designated leader for simplicity. The leader instates agreement by proposing a client transaction to all other replicas. Unlike 2PC, in Paxos, the leader suggests an order for each transaction by assigning a sequence number (i.e., ballot number) to this proposal, which states the order to execute the transaction. On receiving the proposal from the leader, the replicas acknowledge the receipt of this proposal. Once the leader receives acknowledgments from a majority of replicas ($\mathbf{f} + 1$), it informs all the replicas that a consensus has been reached; the replicas can now execute the transaction in the proposed order.

Paxos can be implemented in a decentralized manner, shown by Paxos$^{\oplus}$ in Figure 1, by combining the vote and commit stages.

3.4 Practical Byzantine Fault-Tolerance Consensus (PBFT). Although Paxos can handle node failures and guarantees safe consensus under an unreliable network, it cannot sustain Byzantine or malicious attacks. PBFT [24] is often referred to as the first protocol to present a practical consensus in the presence of Byzantine nodes. To do so, PBFT requires an additional phase of agreement than Paxos and allows at most one-third of replicas fail or act maliciously ($\mathbf{n} = 3\mathbf{f} + 1$). We present a centralized (linear) and a decentralized (original) version of PBFT. In the decentralized version, PBFT$^{\oplus}$, the leader first assigns a sequence number to the client request and broadcasts the proposal to all replicas. In the prepare phase, replicas communicate with each other to validate the leader's proposal, and upon receiving votes from two-thirds of the replicas, each replica enters the commit phase. Similar to the prepare phase, in the commit phase, replicas communicate with each other in a decentralized manner to commit the client transaction.

In the centralized (linear) version of PBFT, L-PBFT, each of the prepare and commit phases is divided into two linear phases: one from the replicas to the leader and one from the leader to the replicas. In each phase, the leader waits for votes from two-thirds of the replicas before sending its message (either prepare or commit).

Remarks. Although these protocols target distinct settings, our elemental structuring makes it obvious that these protocols can be derived from each other. We observe that although 2PC and Paxos have similar phases, in 2PC, each node locally decides whether it can order the leader's proposal with respect to other transactions it has received and then votes to commit or abort that transaction. This is in contrast to Paxos where the leader determines the ordering. A similar observation holds in the case of 3PC and linear PBFT.

4 COMPLEX ELEMENTAL PROTOCOLS

A key advantage of our elemental abstraction is that it can be used to design and explain a lot more protocols than the standard agreement protocols. This is useful in designing efficient variations that require fewer messages or phases, and yield higher throughput. Next, we illustrate this with some examples.

4.1 Speculative Agreement. As agreement protocols require multiple phases, prior works have introduced several optimizations to these protocols [41, 49, 61, 89]. One famous optimization extracted from database theory is optimistic or speculative execution. Several agreement protocols employ speculative execution to eagerly execute a transaction without waiting for the transaction to be committed at all the nodes. As a result, these protocols are able to reduce one or more phases, which helps them to yield higher throughputs and lower latencies.

SpecPaxos [89] and Zyzzyva [61] use speculative execution to allow their replicas to execute the client transaction as soon as they receive a valid proposal from the leader. This allows SpecPaxos and Zyzzyva to process requests in one and two fewer phases respectively, compared to their decentralized variants. Similarly, PoE [41] (a BFT protocol) requires its replicas to only prepare a transaction prior to its execution. This optimization though requires one more phase than Zyzzyva; it reduces one phase from PBFT$^{\oplus}$. However, these optimizations do not come for free; these protocols require their clients to wait for a larger number of matching responses. For example, SpecPaxos clients need responses from $\mathbf{f} + 1$ replicas, Zyzzyva clients need responses from all the $\mathbf{n} = 3\mathbf{f} + 1$ replicas, and PoE clients need responses from $2\mathbf{f} + 1$ replicas.

4.2 Parallel Consensus. A common characteristic of all the protocols that we have analyzed till now is that they rely on a single leader node. Generally, this leader continues sending out proposals until it fails. If the leader fails, these protocols present mechanisms to replace the leader (refer to §5). However, the leader replacement process is expensive and leads to a significant drop in system throughput. As a result, several recent protocols, such as Mencius [74], and Rcc [44], allow all the replicas to act as leaders at the same time. This leads to several instances of consensus running in parallel at each replica.

In Figure 1, we illustrate the compound structure of the Mencius protocol. As Mencius runs i, $1 \leq i \leq \mathbf{n}$, instances of Paxos in parallel, we add new notations in its compound structure to represent this behavior. We add *corner blocks* (⌜⌟) to capture elements that are run by each instance without any coordination.

Each instance of Mencius follows the elements of decentralized Paxos (**Pr**, and **Co**$^{\oplus}$). After all the instances have committed their respective transactions, a *global order* of all the transactions is generated on each replica. We represent this global ordering as a *new element* **O**. Post global ordering, each node executes the transactions. Notice that each instance works independently and does not need to wait for global ordering and execution. Similarly, we also illustrate the compound structure for a parallel BFT consensus protocol, Rcc [44].

4.3 Wide Area Agreement. A key limitation of various agreement protocols that we have described until now is that they are oblivious to the physical location of the nodes. Specifically, these protocols assume identical available bandwidth and round-trip costs

for all the nodes. Prior works [7, 34, 45, 51, 84] have shown that this is not the case and the nodes may be dispersed across the globe. If this is the case, the communication between two nodes that are globally distant will bottleneck the protocol performance. To resolve this challenge, protocols like Steward [7], GeoBFT [45], Ziziphus [14] and Gec [51] advocate a topology-aware architecture.

In Figure 1, we take GeoBFT as an example and present the compound representation for the same. In this protocol, the replicas are divided across shards, though all the replicas hold a copy of the data. As a result, this is not the usual form of data-sharding, rather a form of replica grouping. Specifically, all the neighboring replicas are grouped together in a single shard. Each shard runs the PBFT protocol on its own set of client transactions. Once a transaction is committed, it is communicated to every other shard. This communication across the shards is crucial, as shards are often geographically distant. Hence, a common aim is to minimize this communication cost. In our representation, we associate this *inter-shard communication* with a new element **Is** and use the notation of *bidirectional line* to represent this communication. Further, the number of shards can vary, so our representation illustrates the number of shards with a *superscript s* and denotes the replicated-sharded nature of this protocol by wrapping it with angle brackets and vertical lines. The vertical lines on the exterior denote that the protocol is essentially replicated.

4.4 Sharded-Replicated Agreement. Although geo-replicated sharding avoids expensive communication among globally dispersed replicas, it expects each node to order and execute the same set of transactions. Essentially, it is expensive to guarantee consistent replication across all the replicas. A popular way to avoid this cost is to have a sharded-replicated system [12, 29, 62].

CFT systems like Spanner [29] and MDCC [62], and BFT protocols like SharPer [10], Saguaro [11], Cerberus [52], ByShard [53], AHL [31], and RingBFT [91]. employ the sharded-replicated architecture, where each shard manages a distinct set of data-items, and each shard replicates its data. This architecture allows each shard to independently run agreement on the transactions that only require access to its data-items, execute the ordered transactions and reply to the clients. Such transactions are also termed as *intra-shard transactions*. In Figure 1, we represent the intra-shard consensus in CFT systems like Spanner; we give it a name *S-Paxos*. Although intra-shard consensus requires no communication among the shards, we add the element **Is** to represent forwarding transactions to the correct shard. Intra-shard consensus decreases replication and boosts system throughput as each shard runs consensus in parallel.

5 ELEMENTAL FAILURE MECHANISMS

Until now, the focus of our elemental representations was to explain the non-failure flow of different agreement protocols. However, failures are prevalent among databases, and each agreement protocol is expected to handle a subset of possible failures. Specifically, nodes or replicas can fail-stop, crash, or return arbitrary results. In extreme cases, some of the nodes may be controlled by the adversary due to which they may collude or not respond [24]. Similarly, the messages communicated among nodes may get delayed, dropped or lost. As a result, in this section, we introduce elements that help agreement protocols recover from failures.

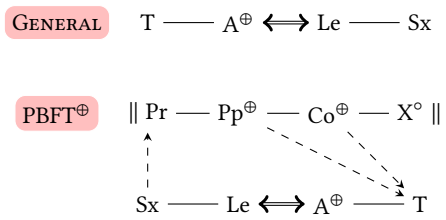

Figure 2: Chemical representation of the general failure flow and the complete flow of transaction in PBFT.

(1) **Timeout (T)**. On receiving a proposal from the leader, each node attempts to reach a common decision on that proposal. To do so, each node waits to transit from one element to another as per the protocol. However, due to unexpected failures, a node may get stuck on a specific element for a prolonged period of time. To allow nodes to progress, despite failures, protocols often require nodes to wait at each element for only a fixed period of time. Post this period, the node *timeouts* and switches to the failure recovery path. Clearly, timeout plays a critical role, and we acknowledge this fact by representing it with an element **T**.

(2) **Announce (A)**. Once a node timeouts, it announces this fact to all the other nodes in the network. Often, the announcer also sends its current state. This state may include all the transactions (and the global decisions) since the last checkpoint. The aim of this announcement is to inform the other nodes and to plan the next steps for recovery. We use element **A** to represent this phase.

(3) **Leader Election (Le)**. Post announcement, the participating nodes may conclude that the current leader has failed. In such a case, they need to elect a new leader. As stated in Section 2.2, several agreement protocols suggest running a leader election algorithm, which democratically elects a leader based on the number of votes received in support of each candidate. The leader election algorithm requires nodes to exchange votes and count the ballots in the favor of each candidate. Some protocols also suggest a deterministic leader election where the next leader (if the current one fails) is predefined at the start of the protocol [24, 41]. As the leader drives consensus, we denote the leader election phase with element **Le**.

(4) **State Exchange (Sx)**. Post leader election, the new leader is expected to bring all the nodes to the common state. To do so, often, the leader constructs the common state from all the received announcements and sends this constructed state to all the nodes. When a node receives the constructed state from the new leader, it updates its state. We use element **Sx** to reflect state exchange.

5.1 Failure Flows

We now use the elements that we described earlier in this section to present a generic compound representation for the failure flow. To do so, we introduce some additional notations. We use *doubly bidirectional arrows* to represent a connection to element **Le**. We denote the entry and exit to failure-flow with *unidirectional arrows*.

In Figure 2, we first illustrate the general failure path adopted by most protocols. Following this, we illustrate the full transactional flow in PBFT$^{\oplus}$. The doubly bidirectional arrow connecting elements **A** and **Le** states that depending on the protocol, a leader election

may or may not take place. Further, we associate $\oplus$ with element **A** as every node in the network may send an announcement message to every other node. Such a communication pattern is representative of the clique topology. In PBFT$^{\oplus}$ protocol, as replicas may timeout either during the prepare or commit phase, we provide entry to the failure path through these elements. Post-completion of the failure path, the new leader is expected to initiate consensus on the new client transactions. As a result, we have an exit path from the failure flow.

6 EXPERIMENTAL EVALUATION

To illustrate that our chemical representations ease analyzing and designing agreement protocols, we create an experimental prototype on top of Bedrock [15] framework. Bedrock is an experimental system that helps to design and evaluate different agreement protocols. Bedrock follows a modular architecture that associates a plug-and-play module with each element. Any developer can select modules of their choice and design a protocol.

We use these modules to create *five* existing agreement protocols: 2PC, 3PC, Paxos, PBFT, and PoE. We use these protocols to conduct an initial study that benchmarks their performance against each other. We ran experiments on the Amazon EC2 cluster; each node is deployed on a VM, which is a c4.2xlarge instance with 8 vCPUs and 15GB RAM, Intel Xeon E5-2666 v3 processor clocked at 3.50 GHz. When reporting throughput, we use an increasing number of client requests until the end-to-end throughput is saturated and state the throughput and latency just below saturation. The results reflect end-to-end measurements from the clients. Clients execute in a closed loop. We use micro-benchmarks commonly used to evaluate BFT systems, e.g., BFT-SMaRt [19]. Each experiment is run for 120 seconds (including 30s warm-up and cool-down). The reported results are the average of five runs.

We evaluate the throughput and latency of the protocols by increasing the number of nodes **n** in a failure-free situation. We vary the number of nodes in an experiment from 3 to 100, use batching with a batch size of 400 and a workload with client request/reply payload sizes of 128/128 byte. Figure 3 depicts the results. In large networks, Paxos demonstrates 52% higher throughput and 51% lower latency compared to 2PC. This is because Paxos tolerates **f** failures (out of 2**f** + 1 nodes) while 2PC requires all nodes to participate. Increasing the number of nodes shows a performance trade-off between 3PC and PBFT. On the one hand, 3PC has a linear message complexity while PBFT incurs quadratic message complexity. On the other hand, in 3PC, all nodes are supposed to participate while PBFT tolerates one-third failure. As shown, increasing the number of nodes also has a large impact on PBFT (65% reduction) due to its quadratic message complexity. increasing the number of nodes has less impact on the throughput of PoE (39% reduction) compared to PBFT (65% reduction) due to its linear message complexity.

7 RELATED WORK

The past five decades have brought forth several different flavors of agreement: commit protocols [37, 96], CFT protocols [63, 74], and BFT protocols [8, 24, 46–48, 50, 61, 79, 90, 91]. Different surveys and empirical studies have analyzed subsets of agreement protocols, e.g.,

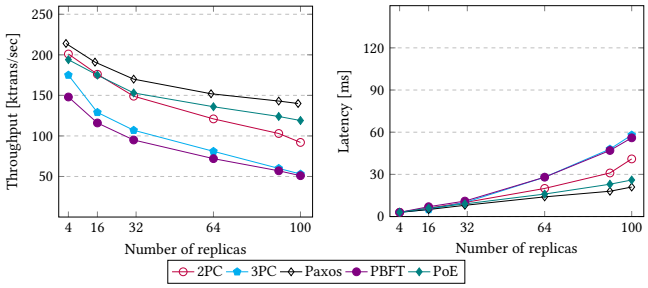

Figure 3: Performance with different number of nodes

BFT protocols [2, 3, 5, 9, 16, 17, 20, 23, 30, 32, 35, 39, 86, 93, 105]. As it is impossible to state all the protocols, following works provide a good survey [9, 15, 32, 43, 83, 86, 102]. Recent protocols like Tapir [106], Janus [77] and Calvin [101] try to combine commitment and crash fault-tolerance into a single design. Sujaya et al. [73] provide a framework that is able to describe a subset of existing commitment and CFT protocols.

Our work is inspired by the Data Calculator [55], which provides the first periodic table for expressing data structures. However, our focus is on agreement protocols. We extract elements and atoms that can express all the flavors of agreement and our elements are not restricted to just commit and CFT protocols.

8 CONCLUSION AND FUTURE WORK

In this paper, we undertook the exercise of understanding various flavors of agreement. This allowed us to present a framework for explaining different types of agreement protocols. We envision this framework as a periodic table of agreement, where multiple elements bond together to create different protocols. Our elemental structure also revealed unexpected relationships between commit, CFT and BFT protocols. To illustrate that our vision works in practice, we presented a modular design that allows the design and analysis of different protocols.

We envision this paper as the first step in designing a unified framework that can easily express different agreement protocols. This exercise of designing a unified elemental framework has revealed several unanswered questions. For example, how can we express deterministic protocols that target implicit commitment of transactions? How do asynchronous protocols that do not make any timing assumptions fit in this framework? Similarly, several BFT protocols advocate partial ordering of client transactions instead of a unique total ordering. The unified framework also needs to pave the path for expressing node recovery and reconfiguration under different failure models. Additionally, a keen reader would have observed an implicit ordering relation between different elements, while some elements are mutually exclusive. Our initial unification model skips these relationships, which need to be expressed in the future. Furthermore, the framework should allow a reader to argue about properties like totality, validity, consistency, and termination.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful feedback and suggestions. This work is funded by NSF grants CNS-2104882 and STTR-2112345.

REFERENCES

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. *Operating Systems Review (OSR)*, 39(5):59–74, 2005.

[2] I. Abraham, G. Gueta, D. Malkhi, L. Alvisi, R. Kotla, and J.-P. Martin. Revisiting fast practical byzantine fault tolerance. *arXiv preprint arXiv:1712.01367*, 2017.

[3] I. Abraham, D. Malkhi, et al. The blockchain consensus layer and bft. *Bulletin of EATCS*, 3(123), 2017.

[4] A. Ailijiang, A. Charapko, M. Demirbas, and T. Kosar. Wpaxos: Wide area network flexible consensus. *IEEE Transactions on Parallel and Distributed Systems*, 31(1):211–223, 2019.

[5] S. Alqahtani and M. Demirbas. Bottlenecks in blockchain consensus protocols. In *Int Conf. on Omni-Layer Intelligent Systems (COINS)*, pages 1–8. IEEE, 2021.

[6] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Prime: Byzantine replication under attack. *Transactions on Dependable and Secure Computing*, 8(4):564–577, 2011.

[7] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing*, 7(1):80–93, 2010.

[8] M. J. Amiri, D. Agrawal, and A. E. Abbadi. CAPER: A cross-application permissioned blockchain. *Proc. VLDB Endow.*, 12(11):1385–1398, 2019.

[9] M. J. Amiri, D. Agrawal, and A. El Abbadi. Modern large-scale data management systems after 40 years of consensus. In *Int. Conf. on Data Engineering (ICDE)*, pages 1794–1797. IEEE, 2020.

[10] M. J. Amiri, D. Agrawal, and A. El Abbadi. SharPer: Sharding permissioned blockchains over network clusters. In *SIGMOD Int. Conf. on Management of Data*, pages 76–88. ACM, 2021.

[11] M. J. Amiri, Z. Lai, L. Patel, B. T. Loo, E. Lo, and W. Zhou. Saguaro: An edge computing-enabled hierarchical permissioned blockchain. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 2023.

[12] M. J. Amiri, B. T. Loo, D. Agrawal, and A. El Abbadi. Qanaat: A scalable multi-enterprise permissioned blockchain system with confidentiality guarantees. *Proc. of the VLDB Endowment*, 15(11):2839–2852, 2022.

[13] M. J. Amiri, S. Maiyya, D. Agrawal, and A. El Abbadi. SeeMoRe: A fault-tolerant protocol for hybrid cloud environments. In *Int. Conf. on Data Engineering (ICDE)*, pages 1345–1356. IEEE, 2020.

[14] M. J. Amiri, S. Maiyya, D. Shu, D. Agrawal, and A. El Abbadi. Ziziphus: Scalable data management across byzantine edge servers. In *Int. Conf. on Data Engineering (ICDE)*. IEEE, 2023.

[15] M. J. Amiri, C. Wu, D. Agrawal, A. E. Abbadi, B. T. Loo, and M. Sadoghi. The bedrock of bft: A unified platform for bft protocol design and implementation. *arXiv preprint arXiv:2205.04534*, 2022.

[16] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis. Sok: Consensus in the age of blockchains. In *Conf. on Advances in Financial Technologies (AFT)*, pages 183–198. ACM, 2019.

[17] C. Berger and H. P. Reiser. Scaling byzantine consensus: A broad analysis. In *Proceedings of the 2nd Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, pages 13–18, 2018.

[18] P. A. Bernstein and E. Newcomer. *Principles of Transaction Processing for Systems Professionals*. Morgan Kaufmann, 1996.

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

[20] A. Bessani, J. Sousa, and E. E. 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. IEEE, 2014.

[21] F. Brasileiro, F. Greve, A. Mostéfaoui, and M. Raynal. Consensus in one communication step. In *Int. Conf. on Parallel Computing Technologies (PaCT)*, pages 42–50. Springer, 2001.

[22] E. Buchman, J. Kwon, and Z. Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[23] C. Cachin and M. Vukolić. Blockchain consensus protocols in the wild. In *Int. Symposium on Distributed Computing (DISC)*, pages 1–16, 2017.

[24] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[25] T. D. Chandra, R. Griesemer, and J. Redstone. Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407, 2007.

[26] A. Charapko, A. Ailijiang, and M. Demirbas. Pigpaxos: Devouring the communication bottlenecks in distributed consensus. In *SIGMOD Int Conf on Management of Data*, pages 235–247, 2021.

[27] S. Chasins, A. Cheung, N. Crooks, A. Ghodsi, K. Goldberg, J. E. Gonzalez, J. M. Hellerstein, M. I. Jordan, A. D. Joseph, M. W. Mahoney, A. Parameswaran, D. Patterson, R. A. Popa, K. Sen, S. Shenker, D. Song, and I. Stoica. The Sky Above The Clouds, 2022.

[28] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche. Upright cluster services. In *Symposium on Operating systems principles (SOSP)*, pages 277–290. ACM, 2009.

[29] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford. Spanner: Google's globally distributed database. volume 31, New York, NY, USA, aug 2013. Association for Computing Machinery.

[30] M. Correia, G. S. Veronese, N. F. Neves, and P. Verissimo. Byzantine consensus in asynchronous message-passing systems: a survey. *International Journal of Critical Computer-Based Systems*, 2(2):141–161, 2011.

[31] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 123–140, New York, NY, USA, 2019. Association for Computing Machinery.

[32] T. Distler. Byzantine fault-tolerant state-machine replication from a systems perspective. *ACM Computing Surveys (CSUR)*, 54(1):1–38, 2021.

[33] D. Dobre, M. Majuntke, M. Serafini, and N. Suri. Hp: Hybrid paxos for wans. In *European Dependable Computing Conf. (EDCC)*, pages 117–126. IEEE, 2010.

[34] M. Y. et al. HotStuff: BFT consensus with linearity and responsiveness. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pages 347–356. ACM, 2019.

[35] F. Gai, A. Farahbakhsh, J. Niu, C. Feng, I. Beschastnikh, and H. Duan. Dissecting the performance of chained-bft. In *Int Conf on Distributed Computing Systems (ICDCS)*, pages 595–606. IEEE, 2021.

[36] J. Gehrke, L. Allen, P. Antonopoulos, A. Arasu, J. Hammer, J. Hunter, R. Kaushik, D. Kossmann, R. Ramamurthy, S. T. V. Setty, J. Szymaszek, A. van Renen, J. Lee, and R. Venkatesan. Veritas: Shared verifiable databases and tables in the cloud. In *9th Biennial Conference on Innovative Data Systems Research, CIDR*. www.cidrdb.org, 2019.

[37] J. Gray. Notes on data base operating systems. In *Operating Systems, An Advanced Course*, pages 393–481. Springer-Verlag, 1978.

[38] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. K. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu. Sbft: a scalable decentralized trust infrastructure for blockchains. In *Int. Conf. on Dependable Systems and Networks (DSN)*, pages 568–580. IEEE/IFIP, 2019.

[39] D. Gupta, L. Perronne, and S. Bouchenak. Bft-bench: Towards a practical evaluation of robustness and effectiveness of bft protocols. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 115–128. Springer, 2016.

[40] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi. Proof-of-execution: Reaching consensus through fault-tolerant speculation. In *Int Conf. on Extending Database Technology (EDBT)*, pages 301–312, 2021.

[41] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi. Proof-of-execution: Reaching consensus through fault-tolerant speculation. In *Proceedings of the 24th International Conference on Extending Database Technology, EDBT*, pages 301–312. OpenProceedings.org, 2021.

[42] S. Gupta, J. Hellings, and M. Sadoghi. Brief announcement: Revisiting consensus protocols through wait-free parallelization. In *33rd International Symposium on Distributed Computing (DISC 2019)*, volume 146, pages 44:1–44:3. Schloss Dagstuhl, 2019.

[43] S. Gupta, J. Hellings, and M. Sadoghi. *Fault-Tolerant Distributed Transactions on Blockchain*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2021.

[44] S. Gupta, J. Hellings, and M. Sadoghi. RCC: resilient concurrent consensus for high-throughput secure transaction processing. In *37th IEEE International Conference on Data Engineering, ICDE*, pages 1392–1403. IEEE, 2021.

[45] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi. ResilientDB: Global scale resilient blockchain fabric. *Proc. VLDB Endow.*, 13(6):868–883, 2020.

[46] S. Gupta, S. Rahnama, E. Linsenmayer, F. Nawab, and M. Sadoghi. Reliable transactions in serverless-edge architecture. *CoRR*, abs/2201.00982, 2022.

[47] S. Gupta, S. Rahnama, S. Pandey, N. Crooks, and M. Sadoghi. Dissecting BFT Consensus: In Trusted Components we Trust! *CoRR*, abs/2202.01354, 2022.

[48] S. Gupta, S. Rahnama, and M. Sadoghi. Permissioned blockchain through the looking glass: Architectural and implementation lessons learned. In *Proceedings of the 40th IEEE International Conference on Distributed Computing Systems*, 2020.

[49] S. Gupta and M. Sadoghi. Easycommit: A non-blocking two-phase commit protocol. In *Proceedings of the 21st International Conference on Extending Database Technology*, pages 157–168. OpenProceedings.org, 2018.

[50] S. Gupta and M. Sadoghi. Blockchain transaction processing. In *Encyclopedia of Big Data Technologies*, pages 1–11. Springer, 2019.

[51] S. Gupta and M. Sadoghi. Efficient and non-blocking agreement protocols. *Distributed Parallel Databases*, 38(2):287–333, 2020.

[52] J. Hellings, D. P. Hughes, J. Primero, and M. Sadoghi. Cerberus: Minimalistic Multi-shard Byzantine-resilient Transaction Processing. *CoRR*, abs/2008.04450, 2020.

[53] J. Hellings and M. Sadoghi. ByShard: Sharding in a Byzantine Environment. *Proc. VLDB Endow.*, 14(11):2230–2243, 2021.

[54] H. Howard, D. Malkhi, and A. Spiegelman. Flexible paxos: Quorum intersection revisited. In *20th International Conference on Principles of Distributed Systems*,

2017.
[55] S. Idreos, K. Zoumpatianos, B. Hentschel, M. S. Kester, and D. Guo. The data calculator: Data structure design and cost synthesis from first principles and learned cost models. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 535–550, New York, NY, USA, 2018. Association for Computing Machinery.
[56] F. P. Junqueira, B. C. Reed, and M. Serafini. Zab: High-performance broadcast for primary-backup systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 245–256. IEEE, 2011.
[57] 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 *European Conf. on Computer Systems (EuroSys)*, pages 295–308. ACM, 2012.
[58] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan. Themis: Fast, strong order-fairness in byzantine consensus. *The Science of Blockchain Conf. (SBC)*, 2022.
[59] R. M. Kieckhafer and M. H. Azadmanesh. Reaching approximate agreement with mixed-mode faults. *Transactions on Parallel and Distributed Systems*, 5(1):53–63, 1994.
[60] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. *Operating Systems Review (OSR)*, 41(6):45–58, 2007.
[61] R. Kotla et al. Zyzzyva: Speculative byzantine fault tolerance. *ACM Trans. Comput. Syst.*, 27(4):7:1–7:39, 2009.
[62] T. Kraska, G. Pang, M. J. Franklin, S. Madden, and A. Fekete. Mdcc: Multi-data center consistency. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, page 113–126, New York, NY, USA, 2013. Association for Computing Machinery.
[63] L. Lamport. Paxos made simple. *ACM SIGACT News*, 32(4):51–58, 2001. Distributed Computing Column 5.
[64] L. Lamport. Generalized consensus and paxos. 2005.
[65] L. Lamport. Fast paxos. *Distributed Computing*, 19(2):79–103, 2006.
[66] L. Lamport. The part-time parliament. In *Concurrency: the Works of Leslie Lamport*, pages 277–317. 2019.
[67] L. Lamport and M. Massa. Cheap paxos. In *Int. Conf. on Dependable Systems and Networks (DSN)*, pages 307–314. IEEE, 2004.
[68] B. Lampson. The abcd's of paxos. In *PODC*, volume 1, page 13. Citeseer, 2001.
[69] E. Levy, H. F. Korth, and A. Silberschatz. An optimistic commit protocol for distributed transaction management. *ACM SIGMOD Record*, 20(2):88–97, 1991.
[70] H. C. Li, A. Clement, A. S. Aiyer, and L. Alvisi. The paxos register. In *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, pages 114–126. IEEE, 2007.
[71] B. Liskov and J. Cowling. Viewstamped replication revisited. 2012.
[72] S. Liu, P. Viotti, C. Cachin, V. Quéma, and M. Vukolic. Xft: Practical fault tolerance beyond crashes. In *Symposium on Operating systems design and implementation (OSDI)*, pages 485–500. USENIX Association, 2016.
[73] S. Maiyya, F. Nawab, D. Agrawal, and A. El Abbadi. Unifying consensus and atomic commitment for effective cloud data management. *Proceedings of the VLDB Endowment*, 12(5):611–623, 2019.
[74] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building efficient replicated state machines for wans. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, OSDI'08, page 369–384, USA, 2008. USENIX Association.
[75] J.-P. Martin and L. Alvisi. Fast byzantine consensus. *Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.
[76] F. J. Meyer and D. K. Pradhan. Consensus with dual failure modes. *Transactions on Parallel and Distributed Systems*, (2):214–222, 1991.
[77] S. Mu, L. Nelson, W. Lloyd, and J. Li. Consolidating concurrency control and consensus for commits under conflicts. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 517–532, USA, 2016. USENIX Association.
[78] F. Nawab, D. Agrawal, and A. El Abbadi. Dpaxos: Managing data closer to users for low-latency and mobile applications. In *SIGMOD Int. Conf. on Management of Data*, pages 1221–1236. ACM, 2018.
[79] F. Nawab and M. Sadoghi. Blockplane: A global-scale byzantizing middleware. In *35th International Conference on Data Engineering (ICDE)*, pages 124–135. IEEE, 2019.
[80] R. Neiheiser, M. Matos, and L. Rodrigues. Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation. In *ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 35–48, 2021.
[81] B. M. Oki and B. H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Symposium on Principles of distributed computing (PODC)*, pages 8–17. ACM, 1988.
[82] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *Annual Technical Conf. (ATC)*, pages 305–319. USENIX Association, 2014.
[83] M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems*. Springer, 2020.
[84] J. M. Patel, J. Yu, N. Kabra, K. Tufte, B. Nag, J. Burger, N. E. Hall, K. Ramasamy, R. Lueder, C. J. Ellmann, J. Kupsch, S. Guo, D. J. DeWitt, and J. F. Naughton. Building a scaleable geo-spatial DBMS: technology, implementation, and evaluation. In *Proceedings ACM SIGMOD International Conference on Management of Data*, pages 336–347. ACM Press, 1997.
[85] F. Pedone. Boosting system performance with optimistic distributed protocols. *Computer*, 34(12):80–86, 2001.
[86] M. Platania, D. Obenshain, T. Tantillo, Y. Amir, and N. Suri. On choosing server- or client-side solutions for bft. *ACM Computing Surveys (CSUR)*, 48(4):1–30, 2016.
[87] R. Plunkett and A. Fekete. Optimal approximate agreement with omission faults. In *Proceedings of the 9th International Symposium on Algorithms and Computation*, ISAAC '98, page 467–475, Berlin, Heidelberg, 1998. Springer-Verlag.
[88] D. Porto, J. Leitão, C. Li, A. Clement, A. Kate, F. Junqueira, and R. Rodrigues. Visigoth fault tolerance. In *European Conf. on Computer Systems (EuroSys)*, page 8. ACM, 2015.
[89] D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and A. Krishnamurthy. Designing distributed systems using approximate synchrony in data center networks. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation*, NSDI'15, page 43–57, USA, 2015. USENIX Association.
[90] S. Rahnama, S. Gupta, T. Qadah, J. Hellings, and M. Sadoghi. Scalable, resilient and configurable permissioned blockchain fabric. *Proc. VLDB Endow.*, 13(12):2893–2896, 2020.
[91] S. Rahnama, S. Gupta, R. Sogani, D. Krishnan, and M. Sadoghi. RingBFT: Resilient Consensus over Sharded Ring Topology. In *Proceedings of the 25th International Conference on Extending Database Technology, EDBT*, pages 2:298–2:311. OpenProceedings.org, 2022.
[92] M. Serafini, P. Bokor, D. Dobre, M. Majuntke, and N. Suri. Scrooge: Reducing the costs of fast byzantine replication in presence of unresponsive replicas. In *Int. Conf. on Dependable Systems and Networks (DSN)*, pages 353–362. IEEE, 2010.
[93] A. Singh, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. Bft protocols under fire. In *NSDI*, volume 8, pages 189–204, 2008.
[94] H.-S. Siu, Y.-H. Chin, and W.-P. Yang. A note on consensus on dual failure modes. *Transactions on Parallel and Distributed Systems*, 7(3):225–230, 1996.
[95] D. Skeen. Nonblocking commit protocols. In *ACM SIGMOD Int. Conf. on Management of data*, pages 133–142, 1981.
[96] D. Skeen. A quorum-based commit protocol. Technical report, Cornell University, 1982.
[97] D. Skeen and M. Stonebraker. A Formal Model of Crash Recovery in a Distributed System. *IEEE Trans. Softw. Eng.*, 9(3):219–228, 1983.
[98] Y. J. Song and R. van Renesse. Bosco: One-step byzantine asynchronous consensus. In *Int. Symposium on Distributed Computing (DISC)*, pages 438–450. Springer, 2008.
[99] I. Stoica and S. Shenker. From Cloud Computing to Sky Computing. HotOS '21, page 26–32, New York, NY, USA, 2021. Association for Computing Machinery.
[100] P. Thambidurai, Y.-K. Park, et al. Interactive consistency with multiple failure modes. In *Symposium on Reliable Distributed Systems (SRDS)*, pages 93–100. IEEE, 1988.
[101] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: fast distributed transactions for partitioned database systems. In *SIGMOD Int. Conf. on Management of Data*, pages 1–12. ACM, 2012.
[102] R. Van Renesse and D. Altinbuken. Paxos made moderately complex. *ACM Comput. Surv.*, 47(3), feb 2015.
[103] R. Van Renesse, N. Schiper, and F. B. Schneider. Vive la différence: Paxos vs. viewstamped replication vs. zab. *IEEE Transactions on Dependable and Secure Computing*, 12(4):472–484, 2014.
[104] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Symposium on Principles of Distributed Computing (PODC)*, pages 347–356. ACM, 2019.
[105] G. Zhang, F. Pan, M. Dang'ana, Y. Mao, S. Motepalli, S. Zhang, and H.-A. Jacobsen. Reaching consensus in the byzantine empire: A comprehensive review of bft consensus algorithms. *arXiv preprint arXiv:2204.03181*, 2022.
[106] I. Zhang, N. K. Sharma, A. Szekeres, A. Krishnamurthy, and D. R. K. Ports. Building consistent transactions with inconsistent replication. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15, page 263–278, New York, NY, USA, 2015. Association for Computing Machinery.