Contributions to the practice and theory of state-machine replication

Pierre Sutra

INSTITUT POLYTECHNIQUE DE PARIS

TELECOM SudParis

IP PARIS

Habilitation à Diriger des Recherches

Contributions to the Practice and Theory of State-Machine Replication

Habilitation à Diriger des Recherches de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626 : École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité : Informatique

Thèse présentée et soutenue à Palaiseau, le 16 Janvier 2024, par

PIERRE SUTRA

Composition du Jury :

Petr Kuznetsov Professeur, Télécom Paris, France	Rapporteur
Étienne Rivière Professeur, Université catholique de Louvain, Belgium	Rapporteur
Roberto Palmieri Associate Professor, Lehigh University, USA	Rapporteur
Annette Bieniusa Professor, University of Kaiserslautern-Landau, Germany	Examinatrice
Michel Raynal Professeur Émérite, Université de Rennes, France	Examinateur
Gaël Thomas Directeur de Recherche, Inria, France	Examinateur

To Patricia, my sister, and my parents.

Disclaimer

An Habilitation à Diriger des Recherches (HDR) shows that one is fit to teach and conduct independant research. For that reason, this thesis presents some of the research works I conducted during my career, focusing on the contributions to the theory and practice of state-machine replication. On purpose, this document does not delve into all the technical details but instead provides an overview of this research.

Acknowledgments

First of all, I would like to thank warmly the rapporteurs of this HDR thesis: Petr Kuznetsov (Professeur, Télécom Paris, France) Roberto Palmieri (Associate Professor, Lehigh University, USA), and Étienne Rivière (Professeur, Université catholique de Louvain, Belgium). I am also grateful to the examinateurs who accepted to review this work: Annette Bieniusa (Professor, University of Kaiserslautern-Landau, Germany), Michel Raynal (Professeur Émérite, Université de Rennes, France), and Gaël Thomas (Directeur de Recherche, Inria, France).

This research would not have been possible without the help of the many people with who I have co-authored papers, debated, and shared ideas. Many thanks to all of you!

I want also to thank the members of the various universities, research institutes, and companies where I have had the chance to spend some time.

Last but no least, I am much grateful to the members of the Benagil group and the département INF at Télécom SudParis where I have the chance to work.

Contents

Chapter 1

Introduction

1.1 Background

Computers are playing a central role in our daily life. They are now everywhere, from the smallest processing unit in wearables to blades and supercomputers in the datacenter. These computers allow to accelerate exchanges and our modern economy crucially depends on them. As a consequence they must be highly available. To achieve this, modern computing infrastructures implement principled approaches for reliable (dependable) computing. One of them is data replication. In this approach, a logical item (e.g., a row in a database table) is physically copied across multiple computers.

Data replication brings two key benefits. The first and foremost one is to avoid data loss. If a copy fails, the others survive, possibly taking over shortly after the initial copy is unresponsive. The second benefit is with respect to performance. Client processes accessing a replicated item may do so by contacting any of its copies. For instance, in a geo-distributed context where data resides at different locations, a client can access the closest copy.

The history of data replication is rich, with early appearances dating back from the 70s. Since then, techniques and principles for data replication have appeared across a variety of domains in computer science including hardware design, operating system, storage and databases, cloud infrastructures, or more recently blockchain. Depending on the considered system, data replication comes in various flavors. If data is immutable (such as in a web cache, or in a content-sharing peer-to-peer system), this mainly boils down to an addressing problem, that is finding one machine having a copy. Once mutations are possible data replication becomes arduous. One of the key difficulties is that real systems are subject to (hardware or software) failures and asynchrony. To mask these, the machines replicating data must solve complex distributed problems.

An astonishing fact here is that several of these problems are sometimes not solvable. More precisely, the frontier between the solvable and the unsolvable depends on the considered problem and the assumptions on the underlying distributed system. Furthermore, this frontier is very thin. In fact, it was not understood before the 80s, and came at the surprise of many of the researchers and engineers working in the field. As of today, some substantial amount of work remains to be done to properly characterize the resolubility of these distributed problems and relate them one to another.

Data replication is a cornerstone paradigm in modern computing infrastructures, while jointly being a hairy problem to solve. For these reasons, it has been extensively studied both in industry

and academia. Data replication is closely related to the term "data consistency" which refers to how data is perceived from the perspective of the clients. Depending on data consistency, replication protocols fall generally into two families: weak and strong consistency. Weak consistency protocols chose to answer quickly to the client, sometimes at the cost of losing the side effects of her operations. On the contrary, strongly-consistent approaches prefer to update enough copies of the datum and coordinates them before providing an answer. One of the most successful techniques for strong consistency is state-machine replication. In this approach, client operations that access the replicated item are first ordered then play in this order against the local copy at each replica.

1.2 Content and Contributions

This habiliation thesis focuses on the state-machine replication problem and its variations. As detailed below, the document is split into three parts. In each part, we present historical perspectives on the covered topic, and underline our contributions (the citations in **bold**).

- The first part of this thesis unfolds the history of state-machine replication (SMR), from its early appearance in aircraft systems to its usage in modern cloud computing infrastructures. After this background presentation, we focus on leaderless solutions. This class of protocols parallelizes the task of ordering client accesses to replicated data. We present a framework to decompose and understand leaderless state-machine replication (LSMR) and assemble new protocols. Using this framework, several properties that characterize an implementation are stated. Based on these properties, we survey prominent LSMR protocols and analyze their design choices at the light of a trade-off between reliability and efficiency.
- When a data replica does not hold all the data items accessible in the system, one talks about partial replication (PR). The second part of this thesis provides an historical overview on PR and details past and present solutions to implement it. Two canonical approaches exist for PR: Execute-then-Order and Order-then-Execute. SMR belongs to the second category and it generalizes into partial state-machine replication (PSMR) when a machine may not replicate all the data items. We present PSMR in detail, underlines the interest of such an intermediate abstraction, and detail some possible implementations. Further we present a characterization of the minimal synchrony assumptions to solve PSMR in the unreliable failure detectors model.
- State-machine replication is closely related to the problem of agreement, aka. consensus, in distributed systems. The last part of this thesis offers a perspective on this problem with a focus on the shared-memory model. We first consider the conditions under which this task, and some of its key variations, are solvable. Then, when a solution exists, we discuss its cost in terms of time and space complexity.

1.3 Thesis Outline

Chapter 2 covers state-machine replication and the leaderless variation. In this approach, every machine has a physical copy of every data item. The natural generalization to partial replication is investigated in Chapter 3. Chapter 4 is about agreement in shared memory. Chapter 5 provides a brief summary of the content of this thesis, lists possible future research directions, then closes.

A full bibliography of my research work is available on pages 81–86. Page 87 lists open problems related to the research topics addressed in this thesis. (In the text, they appear with a sign **(Q)**.) The interested reader may also access online the software artifacts mentioned in this thesis.

Chapter 2

The Dawn of Leaderless State-Machine Replication

Distributed services form the basic building blocks of modern computing infrastructures. A large number of clients access these services, and when a client performs a request to a service, she usually expects it to be responsive and consistent.

The seminal state-machine replication (SMR) approach offers the above two guarantees. By replicating the service across multiple servers, client requests progress despite failures, and by executing them in the same order at all the replicas, service data remains consistent. Internally, SMR implements the service as a deterministic state machine together with a set of commands (the client requests). Each server maintains its own local copy of the state machine. An SMR protocol coordinates the execution of the commands applied to the copies, ensuring that the replicas stay in sync.

This section recalls the fundamentals of SMR, introduces its generalization that leverages the commutativity of state-machine commands and the leaderless variation. Further, we define some key properties for leaderless state-machine replication (LSMR) and use them to state some of its limits. We close this section with a brief review of existing LSMR protocols. Notice that for the sake of conciseness, we will focus mainly on the crash-stop failures model.

2.1 Foundations

2.1.1 Early history

State-machine replication (SMR) is introduced in the late 70s, early 80s [10, 14, 16, 19]. At that time, replication protocols are either synchronous, or require some form of perfect failure detection—sometimes without clearly stating it. In [3, 39], the authors describe Software Implemented Fault Tolerance to provide SMR-like redundancy in aircraft systems. A few years later Borg, Baumbach, and Glazer [16] describe a hardware implementation of SMR over a cluster of Motorola M6800s using Unix System III.

The first historical definition of SMR appears in the seminal paper of Lamport [10] on causality in distributed systems. In this paper, Lamport proposes an algorithm to "order totally the events" in a distributed system. The base idea of this algorithm is to compute for each event all the events that

are either causally before or concurrent. For this, each event (e.g., a lock request) is disseminated to all the processes in the system. When a process receives an event, all the events that follows it at that process are ordered after. This mechanism constructs a partial order which can be linearized arbitrarily and stored in a monotonically-growing queue at each replica. In [10], the queue is used to implement a mutual exclusion algorithm. Lamport also mentions that a generalization where "the synchronization is specified in terms of a state machine" is possible. This allows "to implement any desired form of multiprocess synchronization". State-machine replication (SMR) was born. The seminal SMR protocol in [10] is failure free. As we shall see later, the key difficulty in building SMR consists in handling failures and asynchrony.

The tutorial of Schneider [47] introduces the general problem of replicating a state machine and surveys existing techniques to build it. Schneider covers both fail-stop and Byzantine failures. The tutorial underlines the link between agreement, that is the consensus problem, and SMR. Two requirements are identified:

(**Agreement**) Every non-faulty copy of the state machine receives every command; and

(**Order**) Commands are processed in the same order by every non-faulty copy of the state machine.

The conjunction of these requirements ensures that all the copies make the exact same state transitions. Notice that SMR differs from primary replication [6] in the sense that SMR orders *first* commands before applying them.

The tutorial [47] also introduces the common proxy model, where server replicas disseminate the state-machine commands on behalf of the clients. Another interesting abstraction is the notion of stable command. Once a command is stable, it can be applied to the local copy of the state machine. This happens when no command lower than it (in the global order) can be received. The problem of reconfiguring the replica set is also mentioned in [47].

Notice that the above decomposition into an agreement and an order property is not exactly the usual one. In today's terms, SMR is defined as a replicated log. With more details, each process p holds a log, $\log_p$. Initially, each entry $i > 0$ in $\log_p[i]$ is empty. Every run of SMR must satisfy the properties below:

(**Validity**) A command appears in $\log_p$ at process p once, and only if it was submitted before.

(**Stability**) If $\log_p[i] = \mathsf{c}$ holds at some point in time, for some command c, it is also true at any later time.

(**Consistency**) For any two processes p and q, if $\log_p[i]$ and $\log_q[i]$ are both non-empty, then they are equal.

Commands are applied to the local copy of the state machine following the log order, that is $\log[i]$ may execute provided $\log[i-1]$ is already executed or $i = 0$. In what follows, we call this abstraction *Classic SMR*.

The above definition is better than the one provided in [47], which misses for instance that a command executes only once. Moreover, it is in some sense clearer: Validity and stability are both local properties. Hence, the sole property to construct is consistency which is immediate with consensus: each log entry is implemented with one consensus instance.

Relation with atomic broadcast In fact, the definition of SMR in [47] is the one of atomic broadcast. Atomic broadcast is a group-communication primitive that permits to disseminate messages in a distributed system and to deliver them in a total order. This abstraction extends naturally the FIFO and causal-order broadcast. From a computability perspective, SMR is equivalent to atomic broadcast: to execute a state-machine command, one simply broadcasts it and upon delivery applies it; conversely, atomic broadcast can be defined as a state-machine [77]. In fact, the modern definition of SMR as a totally-ordered log clarifies this relation between the two abstractions. Because the main usage of atomic broadcast is to replicate data, nowadays it is less frequent to encounter new results about it, state-machine replication being favored.

The ISIS project [49] describes the ABCAST protocol that packages both the agreement and ordering guarantees on message delivery. Amoeba [80] is a distributed operating system that uses internally an atomic broadcast protocol to order message between groups of processes. Published in 2004, the survey of Défago, Schiper, and Urbán [123] is still a reference work on existing approaches to implement atomic broadcast. Another key survey is by Hadzilacos and Toueg [66]. These two works classify the protocols based on their guarantees, such as uniformity, validity and ordering. They also identify several classes of algorithms depending on how the order is constructed. *Sequencers* (such as in Amoeba [80], or more recently in Corfu [236]) are leader-driven protocols. Another approach (at work in e.g., [10]) is to *deterministically merge* concurrent messages. Some solutions [17] use a *token* to queue the messages to deliver, in a similar fashion that how distributed mutual exclusion is solved [38].

2.1.2 The golden age

The impossibility result of Fischer, Lynch, and Paterson [21] establishes that consensus is not solvable in a purely asynchronous distributed system even if a single crash may happen. As a consequence, state-machine replication requires some form of partial synchrony. Over the past 40 years, there have been many proposals to define partially-synchronous systems. Popular models include, unreliable failure detectors [77], round-by-round models [86, 189], and various form of eventual synchrony [33].

In particular, Dwork, Lynch, and Stockmeyer [33] show that consensus is solvable when there exists eventually a stable period, that is a moment after which message delay and computation time are both upper-bounded—such bounds are unknown initially. The algorithm in [33] works in a sequence of rounds, or *ballots*, each ballot is led by a process. During a ballot, the leader attempts to take a lock by contacting a majority of processes. If this succeeds, the leader enacts a decision based on the decision of the previous ballots, or proposes its own value.

The algorithm of Dwork, Lynch, and Stockmeyer [33] is a non-parallel version of the seminal Paxos consensus protocol invented by Lamport [88]. The Paxos algorithm is concurrently discovered by Oki and Liskov [34] under the name viewstamped replication. In Paxos, a new ballot is started only when the current leader is not responsive. Moreover, when the leader takes a decision, it broadcasts a commit message to inform all the processes in the system. With those changes, latency is in $O(1)$ message delays instead of $O(n)$ as in [33].

We mentioned earlier that consensus is repeated to implement the replicated log in SMR. The leader of Multi-Paxos, the repeated variation of Paxos, drives all such instances as long as it is responsive. In particular, the next consensus instance starts directly at the ballot of the previous one. This permits to skip the locking phase (aka., phase 1 in Paxos) in which the proposals of prior ballots are discovered. The leader directly executes phase 2 for all the next consensus instances. If

a quorum of replicas *accepts* the proposed command (or a batch of them) in phase 2, the decision is *committed*.

In [77], the authors describe an algorithm similar to Paxos that uses an unreliable failure detector (namely, $\Diamond S$). Failure detector $\Diamond S$ ensures that *(i)* every failure is eventually detected at every correct process, and *(ii)* that eventually some correct process is never mistakenly detected as faulty by the correct processes. An improved version of the solution in [77] is proposed by Schiper [83]. Essentially, the improvement in [83] is the classical idea in Paxos where phase 2B messages are broadcast to all the processes; this cuts one message delay. $\Diamond S$ is equivalent to $\Diamond W$, that is itself equivalent to Ω, the eventual leader oracle. Chandra, Hadzilacos, and Toueg [76] show that, when a majority of processes is correct, Ω is the weakest failure detector to solve consensus. (The work in **[371]** describes a proof à la Chandra et al. but based on constructive arguments to find the critical configuration.)

The solutions in [77, 83] require reliable communication channels. In practice, this is expensive and engineers prefer Paxos which does not make any assumption over the underlying message-passing system.

All the above partially-synchronous consensus protocols work similarly. In the parlance of failure detectors, they are indulgent [95]. This means that they are at all time safe, even if the conditions under which they progress are not met yet. Here, we need to stress out that this departs from early works on consensus (as well as on atomic broadcast) which demand strong synchrony assumptions, e.g., perfect failure detectors, or do not handle failures at all. There is thus a *logical gap* between these early works and indulgent algorithms.

2.1.3 Complexity results

Many authors study the complexity of consensus. Of course, any complexity lower bound regarding consensus also applies to SMR.

Time lower bounds. In [146], Lamport shows that executing a command in SMR requires three message delays in the general case. This can be lowered to two message delays in the conflict-free case, that is when no command is concurrently submitted to the replicated state machine. These measures are taken when clients are co-located with replicas. An additional message delays is required otherwise. Moreover, they are for the good case, that is when the system is synchronous and failure free.

In general, consensus requires one less message delay than SMR for a decision to be taken at all the processes. A single message delay can even be achieved under good conditions. Starting with [23, 60, 378], several works (e.g.,[179]) study these conditions which can be on the input values or the failure pattern (aka., early-stopping).

Halpern, Moses, and Waarts [105] introduce the notion of unbeatable consensus, that is a consensus protocol which strictly dominates others in terms of decision timing whatever the adversary and input values are. In [363], the authors propose matching (non-uniform and uniform) protocols for the synchronous model. The unbeatable property is established using knowledge-based reasoning [18]. The work in [129] studies how to solve consensus quickly once the system behaves synchronously.

About space usage. Dolev and Lenzen [252] show that any crash-resilient consensus algorithm deciding in exactly $f + 1$ rounds has $\Omega(n^2 f)$ worst-case message complexity, where n is the number

of processes in the system and f is the number of actual failures in the run. Of course, it is possible to solve consensus with lower message complexity but higher time complexity. A base illustration of this trade-off is when Paxos acceptors send 2B messages directly to the learners instead of the leader sending a commit message—that is, the optimization in [83]. **(Q)** To the best of our knowledge, the trade-off between time and space complexity has not been fully characterized yet, nor all the Pareto-optimal solutions found.

Ring Paxos [206] chains the messages addressed from the leader to the acceptors in order to improve bandwidth usage in SMR. Many other works explore how to achieve low message complexity (or bits complexity) in consensus (such as, recently [364]). At the very least, each process needs to send one message, that is $\Omega(n)$ is an obvious lower bound. Galil, Mayer, and Yung [72] develop an algorithm that runs in $O(n^{1+\epsilon})$ rounds, for any $0 < \epsilon < 1$, and sends the optimal number of messages (that is, $\Theta(n)$).

In the case of Byzantine failures, an $\Omega(n^2)$ message-complexity lower bound exists [13]. It holds whatever is the time complexity of the underlying protocol and comes from the problem of equivocation.[1]

2.1.4 Modern era

A plethora of SMR protocols were published over the last decade, many of them being variations of (multi-)Paxos. We survey some of the key designs and ideas below.

Zookeeper Atomic Broadcast (Zab) [203, 221] is the replication protocol at core of the Apache Zookeeper coordination kernel. The authors of Zab mention that the protocol "follows the abstract description of Paxos by Lampson [108]". However, Zab differs from Paxos over two important aspects: First, during a consensus instance, Zab agrees on a sequence of commands, instead of a unique command (or a batch of them). A new instance starts only if the leader is suspected. Hence, Zab is more related to Generalized Consensus [132] where c-structs are sequences of commands—more details about this shortly. The second difference is that Zab follows an Execute-Then-Order schema: upon receiving a command, the leader executes it first (tentatively), then appends its updates at the end of the sequence. Such a schema allows to execute non-deterministic commands at the leader and (in the specific case of Zookeeper) reduces bandwidth usage.[2]

Raft [269] implements a replicated log interface. The protocol is very similar to Paxos, only differing it in the way it elects a new leader: to reduce data movement, Raft tries to elect the most up-to-date replica. Raft is more understandable than Paxos essentially because the paper is easier to read than e.g., [88, 107], and because Ongaro and Ousterhout detail the logic from a system's perspective.

In Paxos, as long as the leader is responsive, it is in charge of all the upcoming consensus instances. In contrast, some protocols rotate the responsibility of instances, that is the first ballot of the instance is led by a specific replica. A base example is [33, 178] where replica $p_{k \in \mathbb{N}}$ is in charge of all the instances $l \equiv k \pmod{n}$. Such a rotation is also used to deal with Byzantine failures and mitigate performance attacks (such as in [196]). A key observation here is that the ordering of consensus instances in these protocols is static. This means that instance k comes before instance $k + 1$. Another class of protocols dynamically order instances. In that case, each replica independently commits commands in its instances. The ordering of two instances is then

[1] Equivocation refers to the ability of a Byzantine node to lie regarding the messages it has sent. The lower bound also holds in the message omission failures model.

[2] To some extend, this approach is more akin to primary-backup replication than SMR.

decided by a distributed services, such as a sequencer. As an example, in SDPaxos [323], instance $(123, p_3)$ may precede $(5, p_{42})$ if it arrives first at the sequencer. We will return shortly to approaches that rotate the leader, as this is one of the key ingredient to build leaderless SMR.

As observed in [108], Paxos can be rewritten to distinguish read and write quorums. Write quorums enact decisions, while read quorums serve to retrieve such decisions—or more precisely, a safe approximation of them. This rewriting requires that *(i)* any two write quorums of the same ballot intersect, and *(ii)* any write quorum of a ballot intersects with any read quorum of a higher ballot. It is possible to leverage such a decomposition to boost performance [193, 226]. In particular, Flexible Paxos [290] allows a narrow 2B phase where just $f + 1$ (possibly, $< \lfloor \frac{n+1}{2} \rfloor$) replicas are contacted by the leader.

Multicoordinated Paxos [152] uses more than one leader per ballot. During a multicoordinated ballot b, an acceptor accepts a value only if it is received from a quorum of leaders. If a collision occurs during such a ballot, a higher ballot is started. Multicoordinated Paxos increases dependability at the cost of latency: any minority of leaders can be slow without impacting progress. Another protocol resilient to asynchrony is [344].

Some researchers try to relate SMR protocols one to another. For instance, Renesse, Schiper, and Schneider [280] compare the Zab, Paxos and Viewstamped replication protocols. In [340], the authors explain how Raft and Paxos relate one to another. Such a comparison is also drawn in [332], where the authors present refinement mappings between the two protocols.

In practice The Paxos consensus, Multi-Paxos, and their many variations are complex protocols. Implementing these protocols in a way that the system is always-on for long periods of time is challenging. There are plenty of difficulties including memory management, system membership, and re-configuration.

On that matter, the code bases of Raft [389] and Zookeeper [397] are reference implementations. Chandra, Griesemer, and Redstone [153] present an engineering perspective on the problem. In [148], the authors describe a reconfiguration mechanism to change dynamically the set of SMR replicas. This mechanism is proved correct using the Temporal Logic of Actions (TLA) model checker. Lamport, Malkhi, and Zhou [193] suggest to use an external consensus service to handle system (re-)configuration. Interestingly, consensus is not always necessary in this task, as a quorum system suffices to implement dynamic atomic storage [213].

2.1.5 Hardware-based solutions

Leveraging hardware properties to solve SMR is a recurring topic in distributed systems. In [16], the network conveys a single message at a time. Anker et al. [140] propose a design based on two network adapters at each replica and two switches: replicas send their messages to the first switch that propagates them to the second switch which sends them back to the replicas using the second adapter. István et al. [291] implement Zab on an FPGA, achieving orders of magnitude higher performance than the standard Apache Zookeeper code base.

Recently many works have tried to refresh SMR protocols for the modern datacenter. Some of these works try to leverage the advent of software-defined networks (SDN).

NOPaxos [293] revisits the agreement and ordering properties of Schneider (see §2.1.1): the network is in charge of ordering, while reliability (agreement) stays a responsibility of the replication protocol. Ordering is done with a sequencer implemented at the network level with P4, a language to program software-defined switches. A failure of the sequencer is detected by the network controller

that spins a new one. At a replica, for each sequence number either the associated message or a drop notification is received. NOPaxos is a leader-based protocol. Each request is optimistically run at the leader, and a client knows the response once it receives the optimistic result and $f + 1$ acknowledgments from replicas. A replica asks the leader about the messages it missed. When the leader is suspected, NOPaxos transitions to a new one using a mechanism à la viewstamped replication.

NetPaxos [276] also explores ways to use the network layer to improve the performance of data replication. That work moves Paxos' logic into the switches, with one switch serving as the leader, while the others are Paxos acceptors. To implement NetPaxos, the authors rely on the OpenFlow API, with some specific ad-hoc extensions. The downside of such an approach is that it requires switches to store potentially large amounts of application state. In a follow-up work, the authors implements Paxos in P4 [286]. Interestingly, because P4 cannot synthetizes messages, the replication protocol is essentially a routing one.

Remote Direct Memory Access (RDMA) allows computers in a network to exchange data in main memory without involving the processor [255, 303]. Akin to Direct Memory Access (DMA), RDMA improves throughput and performance as it frees up resources. RDMA comes initially from high-performance computing environments, and it is now commonly found in datacenters.

Mu [339] is a recent proposal to execute fast consensus atop RDMA. Building upon Paxos [88], Mu executes the accept phase in a single one-sided round trip from the leader to the replicas. All-Concur [309] does not rely on a leader replica but instead permits any replica to issue an operation. The protocol allows reads to be executed at the local replica, but this comes at the price of data consistency.

Hermes [341] is another recent SMR protocol that relies on RDMA. The protocol uses a timestamping mechanism to order state-machine commands. At a replica, commands are applied in timestamp order, similarly to many prior replication schemes (e.g., [28]). The protocol timestamps each command at the machine proposing it. For read-modify-write (RMW) operation, this requires to re-submit them if the timestamp is outdated, degrading latency. As a consequence, just a base shared counter is problematic to implement with Hermes.

2.1.6 Leveraging commutativity

In their seminal works, Pedone and Schiper [117], and concurrently Lamport [132], introduce an alternative approach to Classic SMR. They make the key observation that if commands submitted to the state machine commute (e.g., increments over a counter), there is no need to order them. Leveraging this, they replace the totally-ordered log used in Classic SMR with a partially-ordered one. We detail this approach below, which we term *Generic SMR*.

Two commands c and d do not commute when for some state s, applying cd to s differs from applying dc. This means that either both sequences do not lead to the same state, or one of the two commands does not return the same response value in the two sequences. Generic SMR relies on the notion of *conflicts* which captures an over-approximation of the non-commutativity of two state-machine commands.[3] In Generic SMR, each variable $\log_p$ at some process p is a partially-ordered log, that is a directed acyclic graph [132]. In this graph, vertices are commands and any two conflicting commands have a directed edge between them. A command is decided once it is

[3]Some papers (such as [256]) may use the term non-commutativity, or interference, instead of conflicts. Alternatively, the database community considers that commands are in conflict when they are both non-commuting *and* concurrent.

in the partially-ordered log. As with Classic SMR, a decided command gets executed once all its predecessors are executed.

Leveraging commutativity in replicated systems is an old idea—Ellis [8] mentions this for distributed databases in a paper as early as 1977. Another interesting work on commutativity is due to Weihl [35], where the author investigates its use for concurrency control. In [117, 132], the key motivation is the lower-bound result on the time complexity to solve consensus. Indeed, recall from §2.1.3 that consensus requires 3 message delays in the contended case. If everything commutes just 2 message delays, that is a round-trip, are necessary. Generic SMR thus offers the promise to always reach the lower bound of the uncontended case. Furthermore, there is no need of a leader to decide the ordering, which also avoids a potential bottleneck. Hence, taking into account commutativity at the application level can boost SMR performance.

In typical application workloads [142, 202, 254], conflict rate is low. When this happens, the protocols proposed in [117, 132] are efficient. Unfortunately, this is no more the case when contention increases—such as with hot objects if the access distribution is skewed. To commit a contended command, Generalized Paxos [132] necessitates six message delays, which is a round-trip slower than Paxos.

For one-shot consensus, some works propose approaches to leverage contention-free scenarios, while not degrading contended ones [114, 144]. When consensus is repeated, FGGC **[226]** reaches the optimal latency, that is the protocol allows to commit a command in two message delays in contention-free scenarios and three message delays otherwise. The protocol also maintains optimal resiliency, allowing up to $f \leq \lfloor \frac{n-1}{2} \rfloor$ crash failures. To achieve this, FGGC requires to have a single fixed (write) quorum per ballot, centered around the leader.

As with Generalized Paxos, when a collision happens at ballot b, FGGC starts a fresh ballot. At ballot $b+1$, each replica spontaneously sends in a 2B message a merge of its proposal at ballot b with the one of the leader. Despite that this protocol reaches the theoretical lower bound, it is in practice expensive. This comes from the fact that FGCC requires frequent checkpointing to trim metadata **[226]**.

A recent protocol called SwiftPaxos **[391]** alleviates the above problem. The protocol permits replicas to vote on the order in which they receive state-machine commands. Differently from prior works, SwiftPaxos permits a replica to vote twice at a ballot: first for its own ordering proposal, and then to follow the leader. This mechanism avoids restarting the voting process when a disagreement occurs among replicas, saving computation time and message delays. It also removes the need to do frequent checkpointing.

The quest for a latency-optimal SMR protocol is also followed by Zielinski [139]. His optimistic generic broadcast runs concurrently three algorithms: a broadcast protocol, an atomic broadcast protocol, and a set of consensus instances. The broadcast protocol is used for dissemination. The consensus instances are in charge of ordering conflicting commands, each instance defining a pairwise order between two commands. If these decisions form a cycle, atomic broadcast is used to resolve them. This much original approach is one of the very first attempt to construct a leaderless protocol for SMR. We cover this family of protocols in the next section.

Characterization An intriguing question is to distinguish Classic and Generic SMR. On paper, Generic SMR should order commands only when conflicts occur. Unfortunately, it is difficult in general to track the exact causal path that led to the execution of a command.

In **[226]**, the authors propose that in conflict-free (synchronous) runs, *genuine* Generic SMR

protocols deliver all the commands in two message delays. Aguilera et al. [92] propose another characterization based this time on the way the protocol uses the failure detector oracles. The protocol is said *thrifty* when there exists a time after which the oracle is not used in every run where the number of conflicting commands is bounded.

Summary

State-machine replication (SMR) takes its roots in redundancy for aircraft systems. The early history is rich, with the FLP impossibility result [21] and the discovery of the very first solutions under partial synchrony [33, 34, 88]. Since then, many SMR protocols have been designed and used in computing infrastructures, ranging from small networked devices to the Cloud. Some of these protocols leverage the semantics of the replicated service for higher performance. Others are specific to a given replication schema (e.g., geo-replication), or to a given hardware (such as RDMA). These solutions are closely related one to another. To a few exceptions (such as [352]), they follow a ballot-based mechanism in which replicas repeatedly attempt to agree over the next state-machine command to process. In practice, SMR protocols are difficult to program due to their many interleavings and corner cases, and because these systems have to run over long periods of time.

2.2 Leaderless SMR

Leaderless state-machine replication is a recent family of replication protocols, introduced in [139, 178, 256]. This section presents the high-level framework of **[345]** to decompose and understand these protocols, as well as to assemble new ones. To this end, we first define the notion of dependency graph and explain how commands are decided in Leaderless SMR. Using the framework, three key properties are then introduced: Reliability, Optimal Latency and Load Balancing. The ROLL theorem that captures an inherent trade-off between these properties is detailed. Further, several Leaderless SMR protocols are presented and their designs explained at the light of this trade-off.

2.2.1 Genesis

Some recent works [139, 256] push one step further the idea of partially-ordered log, as proposed in Generic SMR. In a leaderless state-machine replication protocol, there is no primary to arbitrate upon the ordering of commands. Instead, any process may contribute to the order. A command is stable, and thus executable, once the transitive closure of its predecessors is known locally. As this transitive closure can be cyclic, the log is replaced with a directed graph.

2.2.2 Definition

Leaderless state-machine replication (LSMR) relies on the notion of *dependency graph*, a directed graph that records the constraints defining how commands are executed. For some command c, the incoming neighbors of c in the dependency graph are its *dependencies*. As detailed shortly, the dependencies are executed either before or together with c.

In LSMR, a process holds two mapping: *dep* and *phase*. The mapping *dep* is a dependency graph storing a relation from *Cmd* to $2^{Cmd} \cup \{\bot, \top\}$, where *Cmd* is the set of commands. For a command c, $phase(\mathsf{c})$ can take five possible values: pending, abort, commit, stable and execute. All the phases, except execute, correspond to a predicate over *dep*.

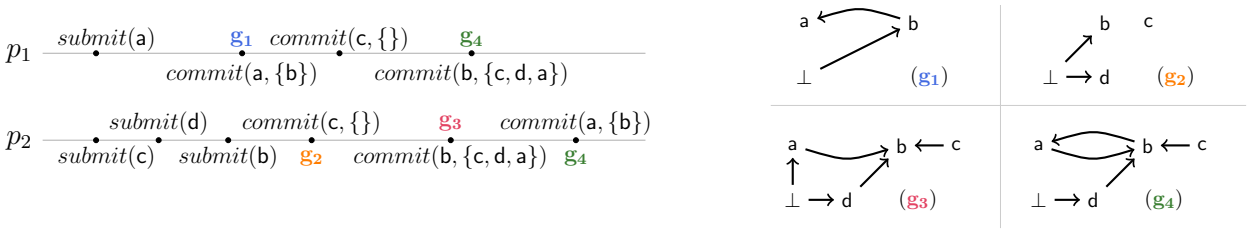


Figure 2.1: (from **[345]**) An example run of LSMR– *(left) processes p_1 and p_2 submit respectively the commands $\{\mathsf{a}\}$ and $\{\mathsf{b},\mathsf{c},\mathsf{d}\}$; (right) the dependency graphs formed at the two processes.*

Initially, for every command c, $dep(\mathsf{c})$ is set to $\bot$. This corresponds to the pending phase. When a process decides a command c, it changes the mapping $dep(\mathsf{c})$ to a non-$\bot$ value. Operation $commit(\mathsf{c}, D)$ assigns D taken in 2^{Cmd} to $dep(\mathsf{c})$. Command c gets aborted when $dep(\mathsf{c})$ is set to $\top$. In that case, the command is removed from any $dep(\mathsf{d})$ and it will not appear later on. Let $dep^*(\mathsf{c})$ be the transitive closure of the dep relation starting from $\{\mathsf{c}\}$. Command c is stable once it is committed and no command in $dep^*(\mathsf{c})$ is pending.

Figure 2.1 depicts an example run of LSMR that illustrates the above definitions. In this run, process p_1 submits command a, while p_2 submits in order c, d then b. The timeline in Figure 2.1 indicates the timing of these submissions. It also includes events during which process p_1 and p_2 commit commands. For some of these events, we depict the state of the dependency graph at each process (on the right of Figure 2.1). At the end of the run, the two processes obtain graph $\mathbf{g_4}$. In $\mathbf{g_4}$, a, b and c are all committed, while d is still pending. We have $dep(\mathsf{a}) = \{\mathsf{b}\}$ and $dep(\mathsf{b}) = \{\mathsf{a},\mathsf{d},\mathsf{c}\}$, with both $dep^*(\mathsf{a})$ and $dep^*(\mathsf{b})$ equal to $\{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d}\}$. Only command c is stable in $\mathbf{g_4}$.

Similarly to Classic and Generic SMR, Leaderless SMR protocols requires that validity holds. In addition, processes must agree on the value of dep for stable commands, and conflicting commands must see each other. More precisely,

(**Validity**) If command c is not pending then c was submitted.

(**Stability**) For each command c, there exists D such that if c is stable then $dep(\mathsf{c}) = D$.

(**Consistency**) If a and b are both committed and conflicting, then $\mathsf{a} \in dep(\mathsf{b})$ or $\mathsf{b} \in dep(\mathsf{a})$.

An aborted command is abandonned and later possibly later by its client. A committed command c gets executed once it is stable. In detail, to execute command c, a process first creates a set of commands, or *batch*, β that executes together with c. This grouping of commands serves to maintain the following invariant:

Invariant 1. *Consider two conflicting commands c and d. If p executes a batch of commands containing c before executing d, then $d \notin dep^*(c)$.*

Satisfying Invariant 1 implies that if some command c is in batch β, then β also contains its transitive non-executed dependencies. Inside a batch, commands are executed according to $\rightarrow$. Let $<$ be a canonical total order over Cmd. Then, $\mathsf{c} \rightarrow \mathsf{d}$ holds iff *(i)* $\mathsf{c} \in dep^*(\mathsf{d})$ and $\mathsf{d} \notin dep^*(\mathsf{c})$; or *(ii)* $\mathsf{c} \in dep^*(\mathsf{d})$, $\mathsf{d} \in dep^*(\mathsf{c})$ and $\mathsf{c} < \mathsf{d}$. If there is a one-way dependency between two commands, LSMR plays them in the order of their transitive dependencies; otherwise the algorithm breaks the tie using the arbitrary order $<$. This guarantees the following invariant.

INVARIANT 2. *Consider two conflicting commands* c *and* d. *If* p *executes* c *before* d *in the same batch, then* $\mathsf{c} \in dep^*(\mathsf{d})$.

One can show that the above two invariants together with the properties of LSMR ensure that the execution is linearizable [45]. This is the very same consistency property as Classic SMR, which ensures that data replication is transparent to the clients.

Generic and Leaderless SMR are very similar. In fact, **[345]** shows that Generic SMR reduces to LSMR without requiring any message exchange (what we call a strong reduction). Notice that such a reduction does *not* hold between Classic and Generic SMR. Indeed, computing a total order on commuting commands would require processes to communicate.

2.2.3 Deciding a command

The above reduction is of interest to relate the two abstractions that are Leaderless and Generic SMR. Yet, it offers a much incomplete picture on how the they compare in practice. Indeed, because the dependency graph might be cyclic, LSMR does not compute an ordering over conflicting commands. Instead, such commands must simply *observe* one another (Consistency property). This fundamental difference explains the absence of a leader in this class of SMR protocols, a feature that we capture in our framework below.

In LSMR, processes have to agree on the dependencies of stable commands. Thus, a subsequent refinement leads to consider a family of consensus objects $(\mathsf{CONS}_\mathsf{c})_{\mathsf{c} \in Cmd}$ for that purpose. For some command c, processes use CONS_c to decide either the dependencies of c, or the special value $(\top)$ signaling that the command is aborted. This agreement is driven by the command *coordinator* ($coord(\mathsf{c})$), a process initially in charge of submitting the command to the replicated state machine. In a run during which there is no failure and all processes are responsive, only $coord(\mathsf{c})$ calls CONS_c.

To create a valid proposal for CONS_c, $coord(\mathsf{c})$ relies on the dependency discovery service (DDS). This shared object offers a single operation $announce(\mathsf{c})$ that returns a pair (D, b), where $D \in 2^{Cmd} \cup \{\top\}$ and $b \in \{0, 1\}$ is a flag. When the return value is in 2^{Cmd}, the service suggests to commit the command. Otherwise, the command should be aborted. When the flag is set, the service indicates that a spontaneous agreement occurs. In such a case, the coordinator can directly commit c with the return value of the DDS service and bypass CONS_c; this is called *a fast path*. A *recovery* occurs when command c is announced at a process which is not $coord(\mathsf{c})$.

The DDS service ensures two safety properties. First, if two conflicting commands are announced, they do not miss each other. Second, when a command takes the fast path, processes agree on its committed dependencies.

More formally, assume that $announce_p(\mathsf{c})$ and $announce_q(\mathsf{c}')$ return respectively (D, b) and (D', b') with $D \in 2^{Cmd}$. Then, the properties of the DDS service are as follows.

(**Visibility**) If $\mathsf{c} \asymp \mathsf{c}'$ and $D' \in 2^{Cmd}$, then $\mathsf{c} \in D'$ or $\mathsf{c}' \in D$.

(**Weak Agreement**) If $\mathsf{c} = \mathsf{c}'$ and $b = true$, then $D' \in 2^{Cmd}$ and for every $\mathsf{d} \in D \oplus D'$, every invocation to $announce_r(\mathsf{d})$ returns $(\top, -)$.

To illustrate these properties, consider that no command was announced so far. In that case $(\varnothing, true)$ is a valid response to $announce(\mathsf{c})$. If $coord(\mathsf{c})$ is slow, then a subsequent invocation of $announce(\mathsf{c})$ may either return $\varnothing$, or a non-empty set of dependencies D. However in that case, because the fast path was taken by the coordinator, all the commands in D must eventually abort.

Algorithm 1 Deciding a command c – code at process p

```
1: submit(c) :=
2:     pre:  p = coord(c) ∨ coord(c) ∈ 𝒟
3:     eff:  (D, b) ← DDS.announce(c)
4:           if b = false then D ← CONS_c.propose(D)
5:           dep(c) ← D
6:           send(c, dep(c)) to 𝒫 \ {p}
7:
8: when recv(c, D)
9:     eff:  dep(c) ← D
```

Based on the above decomposition of LSMR, Algorithm 1 depicts an abstract protocol to decide a command. This algorithm uses a family of consensus objects ($(\mathsf{CONS_c})_{\mathsf{c}\in Cmd}$), a dependency discovery service (DDS) and a failure detector ($\mathcal{D}$) that returns a set of suspected processes.

To submit a command c, a process announces it then retrieves a set of dependencies. This set is proposed to $\mathsf{CONS_c}$ if the fast path was not taken (line 4). The result of the slow or the fast path determines the value of the local mapping $dep(\mathsf{c})$ to commit or abort command c. Notice that such a step may also be taken when a process receives a message from one of its peers (line 8).

During a nice run, the system is failure-free and the failure detector service behaves perfectly. As a consequence, only $coord(\mathsf{c})$ may propose a value to CONS_c and this value gets decided. In our view, this feature is the *key characteristic* of LSMR.

2.2.4 Properties and limits

State-machine replication helps to mask failures and asynchrony in a distributed system. As a consequence, a first property of interest is the largest number of failures (parameter f) tolerated by a protocol. After f failures, the protocol may not guarantee any progress.[4]

(**Reliability**) In every run, if there are at most f failures, every submitted command gets eventually decided at every correct process.

LSMR protocols exploit the absence of contention on the replicated service to boost performance. In particular, some protocols are able to execute a command after a single round-trip, which is clearly optimal [146]. To ensure this property, the fast path is taken when there is no concurrent conflicting command. Moreover, the command stabilizes right away, requiring that the DDS service returns only submitted commands.

*(*Optimal Latency*)* During a nice run, every call to $announce(c)$ returns a tuple (D, b) after two message delays such that *(i)* if there is no concurrent conflicting command to c, then b is set to *true*, *(ii)* $D \in 2^{Cmd}$, and *(iii)* for every $d \in D$, d was announced before.

The replicas that participate to the fast path vary from one protocol to another. Mencius use all the processes. On the contrary, Egalitarian Paxos (EPaxos) solely contacts $\lfloor \frac{3n}{4} \rfloor$ of them (or

[4]When f failures occur, the system configuration must change to tolerate subsequent ones. If data is persisted, the protocol simply stops when more than f failures occurs and awaits that faulty processes are back online.

equivalently, $f + \frac{f+1}{2}$ when $n = 2f+1$). For some command c, a *fast path quorum* for c is any set of $n - F$ replicas that includes the coordinator of c. Such a set is denoted $FQuorums(\mathsf{c})$ and formally defined as $\{Q \mid Q \subseteq \Pi \wedge coord(c) \in Q \wedge |Q| \geq n - F\}$. A protocol has the *Load Balancing* property when it may freely choose fast path quorums to make progress.

*(*Load Balancing*)* During a nice run, any fast path quorum in $FQuorums(\mathsf{c})$ can be used to announce a command c.

The previous properties are formally defined in **[345]**. Table 2.1 indicates how they are implemented by well-known leaderless protocols. The columns 'Reliability' and 'Load Balancing' detail respectively the maximum number of failures tolerated by the protocol and the size of the fast path quorum. Notice that by CAP [115], we have $F, f \leq \lfloor \frac{n-1}{2} \rfloor$ when the protocol matches all of the properties. Table 2.1 also mentions the optimality of each protocol with respect to the ROLL theorem. This theorem is detailed below:

Theorem 1 (**[345]**). *Consider a protocol that satisfies all the ROLL properties. Then, it is true that* $2F + f - 1 \leq n$.

Theorem 1 captures an inherent trade-off between performance and reliability. For instance, tolerating a minority of crashes, requires accessing at least $\lfloor \frac{3n}{4} \rfloor$ processes. This is the setting under which EPaxos operates. On the other hand, if the protocol uses a plain majority quorum in the fast path, it tolerates at most one failure.

A protocol is *ROLL-optimal* when the parameters F and f cannot be improved according to Theorem 1. In other words, they belong to the skyline of solutions (i.e., Pareto optimal). As an example, when the system consists of 5 processes, there is a single such tuple $(F, f) = (2, 2)$. With $n = 7$, there are two tuples in the skyline, $(2, 3)$ and $(3, 2)$. The first one is attained by EPaxos, while Atlas offers the almost optimal solution $(3, 1)$ (see Table 2.1). **(Q)** We note here that the question of finding a ROLL-optimal protocol for all values of the pair (n, f) is still open.

2.2.5 Protocols

This section surveys several LSMR protocols. It details their respective implementations of the DDS and CONS services, and indicates their optimality wrt. ROLL. Table 2.1 recapitulates.

Rotating coordinator For starters, let us consider a rotating coordinator algorithm (such as [196]). In this class of protocols, commands are all conflicting and ordered *a priori* by some relation $\ll$. Such an ordering is usually defined by timestamping commands at each coordinator and breaking ties with the process identities. When $coord(\mathsf{c})$ calls $\mathsf{DDS}.announce(\mathsf{c})$, the service returns a pair $(D, false)$, where D are all the commands prior to c according to $\ll$. Upon recovering a command, the DDS service simply suggests to abort it. In spirit, such a fault-tolerance mechanism is similar to the consensus algorithm on transaction commit by Gray and Lamport [145].

Clock-RSM This protocol improves upon the above schema by introducing a fast path [264]. It also uses physical clocks to speed-up the stabilization of committed commands. In detail, a command is first associated with a timestamp. Its coordinator then broadcasts this information to the other processes in the system. When it receives such a message, a process waits until its local

Protocols	*Properties*			
	Load Balancing ($n - F$)	Reliability (f)	Optimal Latency	ROLL-optimal
Rotating coord.	0	Min	×	×
Clock-RSM [264]	n	Min	×	×
Mencius [178]	n	Min	√	×
Caesar [302]	$\lceil \frac{3n}{4} \rceil$	Min	√	×
EPaxos [256]	LMaj	Min	√	if $n = 2f + 1$
Alvin [271]	LMaj	Min	√	if $n = 2f + 1$
Atlas **[338]**	$\lfloor \frac{n}{2} \rfloor + f$	any	√	if $n \in 2\mathbb{N} \cup \{3\} \wedge f = 1$
Tempo **[350]**	$\lfloor \frac{n}{2} \rfloor + f$	any	×	×

Table 2.1: The properties of several leaderless SMR protocols – Min stands for a minority of replicas ($\lfloor \frac{n-1}{2} \rfloor$), Maj a majority ($\lceil \frac{n+1}{2} \rceil$), and LMaj a large majority ($\lfloor \frac{3n}{4} \rfloor$).

clock passes the command's timestamp to reply. Once a majority of processes have replied, the DDS service informs the coordinator that the fast path was taken.

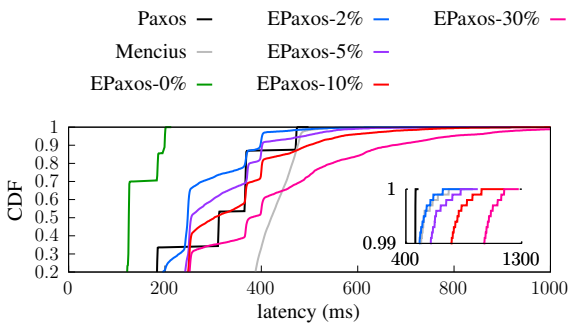
Clock-RSM considers that any two commands conflict. The protocol recovers all the commands below a certain timestamp at once. Because a command might now take the fast path, recovery cannot simply suggest to abort it—as with the rotating coordinator. Instead, Clock-RSM asks a majority of replicas if the command was seen. If this happens to be the case, the command is returned by the DDS service to be proposed to consensus.

Mencius The above two protocols require a committed command to wait all its predecessors according to $\ll$. Clock-RSM propagates in the background the physical clock of each process. A command gets stable once the clocks of all the processes is higher than its timestamp. Differently, Mencius [178] aborts prior pending commands at the time the command is submitted. In detail, *announce*(c) first approximates D as all the commands prior to c according to $\ll$. Then, command c is broadcast to all the processes in the system. Upon receiving such a message, a process q computes all the commands d smaller than c it is coordinating. If d is not already announced, q stores that d will be aborted. Then, q sends back this information to *coord*(c) which removes d from D. Upon recovering c, if the command was received, the over-approximation based on $\ll$ is returned together with the flag *false*. Otherwise when c is unknown, the DDS service suggests to abort it.

EPaxos In [256], the authors present Egalitarian Paxos (EPaxos), a family of efficient LSMR protocols. For simplicity, we now cover the variation which does not involve sequence numbers.

To announce a command c, the coordinator broadcasts it to a quorum of processes. Each process p computes (and records) the set of commands D_p conflicting with c it has seen so far. A call to *announce*(c) returns $(\cup_p D_p, b)$, with b set to *true* iff processes in the fast-path quorum spontaneously agree on dependencies (i.e., for any p, q, $D_p = D_q$). When a process in this initial quorum is slow or a recovery occurs, c is broadcast to everybody. The caller then awaits for a majority quorum to answer and returns (D, false) such that if $\lceil \frac{f+1}{2} \rceil$ processes answer the same set of conflicts for c, then D is set to this value (with $n = 2f + 1$). Alternatively, if at least one process knows c, the union of the response values is taken.

Correctness. Surprisingly, both the TLA$^+$ specification and the Golang implementation of EPaxos [380] use a single variable to track progress across ballots. The work in **[347]** proves



(a) Latency distribution when varying the conflict rate.

	SC	FI	QC	AU	TW
SC		123	25	199	184
FI			120	308	289
QC				202	182
AU					127

(b) Ping distance between sites (in ms).

Figure 2.2: (from **[345]**) Performance comparison of EPaxos, Paxos and Mencius – *5 sites: South Carolina (SC), Finland (FI), Canada (QC), Australia (AU), Taiwan (TW, leader); 128 clients per site;* no-op *service.*

that this is not sufficient: it exhibits an admissible execution in which replicas disagree on the dependencies of a command, breaking the agreement property of LSMR. Adding the missing variable and managing recovery properly correct the problem **[382]**. A fixed implementation of EPaxos is available online **[379]**.

Performance. When contention increases, Egalitarian Paxos exhibits a long tail latency [335, 337, 345]. This is illustrated in Figure 2.2. where we ran an experimental evaluation of EPaxos, Paxos and Mencius in Google Cloud Platform. Figure 2.2a plots the cumulative distribution function (CDF) of the command latency for each protocol. In this experiment, the system spans five geographical locations distributed around the globe, and each site hosts 128 clients that execute *no-op* commands in closed-loop. Figure 2.2b indicates the distance between any two sites. The conflict rate among commands varies from 0% to 30%.[5] We measure the latency from the submission of a command to its execution (at steady state). Two observations can be formulated at the light of the results in Figure 2.2. The tail of the latency distribution in EPaxos is larger than for the other protocols. Furthermore, it increases with the conflict rate.

To alleviate the above problems, Tollman, Park, and Ousterhout [358] propose two modifications to the EPaxos algorithm: First, the authors introduce some delay before returning the dependencies of a command at each replica. When the delay expires, the command is re-ordered wrt. the other concurrent conflicting commands. This helps to have the exact same dependencies reported by the fast path replicas. The delay is based on the average latency measured between pairs of replicas. Second, stability is computed by ignoring part of the dependencies of a command. This mechanism works in the case where EPaxos computes for each command a sequence number together with a dependency set. In detail, a committed command c is stable when for every command $\mathsf{d} \in \mathit{dep}(\mathsf{c})$, either d is also stable, or c has a higher sequence number than d. This idea clearly works when c and d have the same coordinator and commands are received in FIFO order. However, in our opinion, the general case remains unclear.

[5] Each command has a key and any two commands conflict, that is they must be totally ordered by the protocol, if they have the same key. When a conflict rate ρ is applied, each client picks key 42 with probability ρ, and a unique key otherwise.

Atlas In a geo-distributed setting, natural disasters leading to the loss of a full data center are rare, and planned downtimes can be handled by reconfiguring the unavailable replicas out of the system [88, 229, 243]. Furthermore, temporary data center outages (e.g., due to connectivity issues) typically have a short duration [294], and they rarely happen concurrently **[337]**. Leveraging these observations, **[337]** proposes Atlas, a leaderless protocol tailored for planet-scale deployments. Like Flexible Paxos [289], Atlas allows choosing the maximum number of replicas that can fail (f) independently of the overall number of replicas (n).

Atlas uses fast quorums of size $\lfloor \frac{n}{2} \rfloor + f$, and it is optimized for small number of failures (typically, $f \leq 3$). In detail, consider that a fast quorum Q returns the set of dependencies $(D_p)_{p \in Q}$ to the coordinator. The f-threshold union of Q, written $\biguplus_Q$, are all the commands reported at least f times by the replicas in Q. Atlas allows the coordinator to take the fast path when $\cup_{p \in Q} D_p$ equals $\biguplus_Q D_p$ In particular, the protocol always takes the fast path when $f = 1$ (for any value of n). Such an approach is safe for the following two reasons: *(i)* any D_p contains the dependencies computed at the coordinator (which are sent initially), and *(ii)* because there are at most after f failures, recovery retrieves the dependencies taken in the fast path by contacting either the coordinator which is obviously informed of this, or some subset of $|Q| - (f - 1)$ replicas in Q.

Caesar The work in **[345]** shows that long chains in the dependency graph cannot be avoided when the leaderless replication protocol is ROLL-optimal. In practice, these chains impair latency by delaying the execution of committed commands. They explain the tail latency phenomena reported in Figure 2.2.

To solve this issue, Caesar [302] orders commands with the help of logical timestamps, like a rotating coordinator. However, differently from it and much like in Mencius, it tries to abort prior instances faster. In detail, upon submitting a command c, the coordinator timestamps it with its logical clock then it executes a broadcast. As with EPaxos, when it receives c a process p computes the commands D_p received so far conflicting with c. Then, p awaits until there is no conflicting command d with a higher timestamp than c such that $\mathsf{c} \notin \mathit{dep}(\mathsf{d})$. If such a command exists, p replies to the coordinator that the fast path cannot be taken. The DDS service returns $(\cup_p D_p, b)$, where $b = \mathit{true}$ iff no process disables the fast path.

Consider two conflicting commands c and d with timestamp t_c and t_d. A core invariant of Caesar is: $t_c < t_d \Rightarrow c \in \mathit{dep}(c)$. This invariant ensures the visibility property of the DDS service.

Tempo Another timestamp-based protocol is Tempo **[350]**. Tempo borrows to Atlas the narrow path idea based on the threshold union operator. To take the fast path, f replicas must return the highest timestamp among all the timestamps returned by the fast quorum. This is similar to the condition in Atlas, where every dependency is returned f times: in both cases the supremum of all the reported values should be reported (at least) f times.

In the rotating coordinator approach, stability is sensitive to failures. Indeed, to stabilize a command, the coordinator needs to receive a message from all replicas. Clock-RSM removes this limitation by relying on loosely-synchronized clocks. Tempo does not make any assumption on the clocks. Instead, a command committed with timestamp t is stable once a large-enough quorum of replicas has bumped its local clock above t.

The stability check of Tempo is implemented as a background mechanism. When a replica bumps its clock to some value $t' > t$, it sends the *promise* to reject any command submitted with a timestamp in $[t, t']$. Promises are sent synchronously, or asynchronously at a given rate. For

faster decisions, replicas in the fast quorum piggyback theirs promises on their responses to the coordinator. Detecting stability in Tempo relies on a plain majority of replicas. This ensures that the stability-detection quorum intersects with any fast path quorum, as such quorums contain $\frac{n}{2} + f$ replicas. We note here that it would be possible to tailor these two quorums, further narrowing the fast path quorum at the cost of a larger stability-detection quorum. (This variation is not investigated in **[350]**.)

Tempo can timestamp commands at arbitrary fine grain, e.g., a single state-machine variable. This permits to leverage commutativity and boost parallelism. The protocol also manages partial replication, that is when each replica contains only part of the state machine. We cover such a variation in §3.2. To conclude our analysis on Tempo, let us note two drawbacks of the protocol: First, once committed a command has to wait for enough promises to get collected before execution. This can introduce a delay in some pathological (but rare in practice) cases. In the parlance of the framework in §2.2.2, Tempo is not latency optimal. Second, Tempo cannot leverage commutativity in the general case—for instance, two reads on the same state-machine variable. This problem is partly fixable using a more complex timestamping mechanism [365].

Accord Accord [377] is a recent proposal to enhance Apache Cassandra [205], a widely-known distributed data store. This replication protocol relies on the very same invariant as Caesar: replicas agree on the final timestamp of a command, but not on the dependencies. Dependencies are solely used during execution. They over-approximate the commands with a lower final timestamp. Before execution, commands that have a higher timestamp are trimmed from *dep*, that is aborted in our framework. Accord reaches optimal resilience, tolerating a minority of failures among the replicas. As in [290] the authors of Accord observe that one may reduce the size of the fast path quorum, as long as any two fast path quorums intersect over $f + 1$ replicas. (This idea is named an electorate in [377].) A glitch was found in an early version of the Accord replication protocol **[390]**.

Summary

In leaderless state-machine replication (LSMR), each replica contacts a quorum of its peers and may contribute to the ordering of commands. This approach removes the bottleneck due to to the leader in protocols such as Paxos, Viewstamped Replication and Raft. It also boosts availability: when a replica is slow or has failed, commands that commute with all of the pending commands it has submitted so far can still progress.

LSMR is better understood with the help of a framework in which the discovery of the dependencies of a command and the agreement on their final values is split into two disjoint services. Many existing protocols are decomposable this way. These protocols offer various trade-offs in the design space, each offering a compromise between complexity, speed, and availability.

Chapter 3

Partial Replication

State-machine replication, and its leaderless variation studied in the previous section, provide a general solution to replicate data. These techniques need to be adjusted when data is partially replicated, that is when system nodes store only a subset of the full dataset. A canonical example is a key-value store that both replicates and distributes data among a set of machines. Replication provides fault-tolerance, while distribution increases performance as well as the maximum number of bytes stored in the system. Many NoSQL storage systems (e.g., [164, 205, 227, 229, 239, 242, 333, 396] to cite a few) abide by such a design.

This section presents the context and general problem of partial replication, as well as our contributions to that matter. We first provide some historical background on the problem, linking it to distributed transactions and group communication primitives. Then, we present partial state-machine replication (PSMR) which, much like SMR, provides an intermediary abstraction to construct linearizable shared objects. We relate PSMR to the two notions of service partitioning and disjoint access parallelism. Further, we cover several PSMR protocols and lists some of their limits. This section closes with a characterization of the minimal synchrony assumptions under which PSMR is solvable.

3.1 Background

3.1.1 Motivation

Modern computing infrastructures are massively distributed, with billions of interconnected machines and devices. In these infrastructures, replication plays a key role by providing two fundamental properties. First, replication hides failures (whether they be hardware or software faults), allowing distributed services to be always on, or *highly available*. This is a pivotal property for the modern Cloud. Besides that, replication also permits to move data closer to its customers (for instance, a cache for web pages). Replication *improves performance* by cutting latency and/or providing additional computing power to process data.

Partial replication (PR) occurs when the machines in the system do not hold the dataset in full. Such a situation takes place when the dataset is too large to fit in, e.g., a petabyte-scale key-value store. In that case, data is sharded (aka., partitioned) across multiple machines. A central index (such as the metadata server in HDFS [211]) can help to locate which machine is holding what.

```
T(x) :=
 y ← read(x)
 write(y,42)
```

(a) Interactive

```
void raiseAllSalaries(int r,
     DataStore<String, Employee>
     store) = {
 store.newQuery().build()
  .list().stream().map(
    e -> {e.raise(a);}
  );
 store.flush();
}
```

(b) One-shot (Apache Gora [375])

```
db.pizzas.insertMany( [
 { _id: 0, type: "pepperoni", size:
     "small", price: 4 },
 { _id: 1, type: "cheese", size: "
     medium", price: 7 },
 { _id: 2, type: "vegan", size: "
     large", price: 8 }
] )
```

(c) Bulk write (MongoDB [201])

```
T1(x) :=
 y ← read(x)
 run(new T2(x, y))

T2(x, y) :=
 y1 ← read(x)
 if y1 ≠ y then
  abort()
 else
  write(y,42)
```

(d) Non-interactive

```
doubleCAS(a,u,v,b,x,y) :=
 tmp ← read(a)
 tmp2 ← read(b)
 if tmp = u and tmp2 = x then
  write(a,v)
  write(b,y)
  return 0
 return 1
```

(e) Two-phase

```
insert into cycling.cyclist_name (id,
     lastname, firstname)
 values (7bd2, 'knetemann', 'roxxane')
 if not exists;

update cycling.cyclist_name
 set firstname = 'roxane'
 where id = 7bd2
 if firstname = 'roxxane';
```

(f) Lightweight (CQL [205])

Figure 3.1: Various classes of transactions in different programming languages and frameworks. Sub-figures (b), (c) and (f) are actual examples taken from tests and/or the documentation. Sub-figures (a), (d) and (e) are in pseudo-code.

Another popular approach is to use (a variation of) consistent hashing [81], that allows to compute locally the owners of a data item. PR may also happen "naturally" because data is spread across several geographical locations. This situation is common nowadays with global services serving thousands to millions of clients around the globe [225]. A client operation can require the freshest version of a data item to execute (e.g., when booking a plane ticket). Depending on the locality of the workload and the spread of the access distribution, the item is replicated at some of the datacenters, but not all.

A first and foremost starting point to understand PR is the locality property of linearizability. Indeed, Herlihy and Wing [45] prove that linearizable objects are composable. Hence, when there is no invariant across partitions (shards), the overall system is consistent provided it is consistent per partition.[1] In that case, we can divide and conquer the PR problem: maintaining each data partition consistent is enough.

However, in the general case this is not sufficient. If an operation crosses the boundaries of two partitions, it is commonly expected that the operation is *atomic*, that is its effects at each partition should happen all, or none of them. Moreover, as in the single partition case, commands should be consistent across all partitions. A property termed isolation in database systems community (the "I" in ACID). We are interested with this general formulation of the PR problem here.

[1]Such a property is not free with other consistency criterion, e.g., sequential consistency [64].

3.1.2 Early days

The question of data replication is raised after the introduction of packet switching networks [2]. Researchers and engineers propose protocols to maintain the database constraints at each replica, and across them [7, 9]. Early consistency criteria require, e.g., that replicas converge eventually toward the same state [6].

Two key approaches, primary-copy and distributed locking, quickly emerge. In primary-copy replication, transactions are applied locally at a leader replica and then propagated to secondaries (backups). This approach is principled by Alsberg and Day [6] in a geo-distributed protocol à la chain replication [126]. Distributed locking consists in taking appropriate locks at the replicas, before applying the transaction [26]. One first such solutions is the protocol of Ellis [8]: replicas are organized into a ring, each transaction circulates in the ring, grabbing a (database-wide) lock at the replica, and once it returns to its originator, the transaction is executed everywhere. Garcia-Molina [11] refines this protocol to take only the necessary locks at each replica, improving parallelism. Such a variation works for *non-interactive* transactions, that is transactions whose variables do not depend from the result of a read within the transaction. Figures 3.1a and 3.1d illustrate respectively an interactive and a non-interactive transaction.

Quorum-based replication is introduced in [12]. Abbadi and Toueg [36] design an extension to the PR case built atop the view synchrony paradigm [27]. The protocol is very conservative in the sense that two conflicting transaction are likely to abort both. The same restriction happens in [119]. This replication protocol exchanges transactions in the background (via gossip) between partial replicas of a database. If two concurrent conflicting transactions are detected, they have to abort and restart.

Exception handling is common in programming languages. For databases, it takes the form of an `abort` statement, which essentially transforms the transaction into a no-op (transaction T2 in Figure 3.1d is an example). In modern terms, transactions belong to the larger class of abortable objects [253]. When a transaction executes without aborting, it *commits*. PR and data sharding introduces an additional complexity when dealing with aborts: Because execution do not happen at a single replica, it is necessary that all of the involved replicas agree to commit (or abort) the transaction. This is the atomic commitment (AC) problem [118]. The seminal two-phase commit protocol [9] solves AC in a failure-free environment. When failures can be perfectly detected, an additional third phase is required [217]. The most general (partial-synchrony) case is tricky, as it requires solving some form of agreement among the replicas. In fact, all of the partially-synchronous AC protocols designed in these early days of data replication were wrong according to Gray and Lamport [145].

Bubba [42] is a distributed transactional PR system built with AC and distributed locking. Each SQL transaction compiles to a dataflow whose distributed execution happens at run-time over a shared-noting distributed infrastructure.[2] Execution occurs using distributed locks followed by two-phase commit. The system is robust to failures thanks to logging techniques. Nonetheless, if a failure happens, it must halt until the failed machine restarts. Many seminal database systems that support PR designed in the 80s and early 90s, follow a similar approach [22, 25, 51, 65].

[2]Shared-nothing is the *de facto* standard today. However, at that time (80s-90s), mainframes were the dominant computing infrastructure.

3.1.3 Middleware solutions

At the turn of the 21st century, database is already a mature technology with rich capabilities (such as indexing, fault-tolerance, and query processing.) However, replica control, that is keeping copies consistent despite updates, in these systems is quite complex. Most text book replication protocols (e.g., [26]) have low performance and are, therefore, hardly ever used in practice [79]. Typical problems plaguing these solutions are lack of scalability, unacceptable response times, high deadlock probability, and high network traffic. To solve these problems, several papers investigate middleware-based techniques. Below, we survey some of these techniques for PR.

Clustered JDBC (C-JDBC) is a database connector for Java that supports PR [121]. Internally, C-JDBC wires multiple JDBC connectors to replicate a database table across multiple (possibly heterogeneous) backends. C-JDBC is made fault-tolerant thanks to a logging mechanism and the JGroups communication library [151]. SI-Rep [116] is also implemented as a JDBC connector. In this approach, each transaction is first forwarded to a database that executes it tentatively. SI-Rep then inspects the writeset. If it is empty, the transaction commits right away and the result is sent back to the client. Otherwise, the transaction is validated to prevent write-conflicts as required by Snapshot Isolation (SI).

Middle-R [136] is a family of protocols for full replication built atop the seminal Postgres-R system [98, 138].) For performance, a protocol pipelines execution and ordering of the transactions using an optimistic delivery signal at the replicas. Middle-R also permits concurrent non-conflicting transactions to run in parallel while read-only transactions execute against a snapshot. Notice that this might break serializability because snapshots are not guaranteed to be monotonic [160]. One protocol in the family allows reordering transaction after they are delivered; this is similar to [120] which is used in MySQL Group Replication [387]. Such a technique is extended to PR in [109].

3.1.4 Classification and comparison

In [100], and then [101], Wiesmann et al. describe a framework to understand and compare replication techniques. They distinguish various forms: active, passive, eager, and lazy. In passive replication, aka. primary backup [194, 207], execution takes place first at a leader replica that then disseminates the result. This is functionally similar to (eager) primary-copy replication in database systems, as described above. Executing first permits to support non-determinism, because sending updates suffices, and a richer programming model (e.g., interactive transactions). In active replication, operations are first disseminated to the replicas before execution. In the database world, such an approach is called (eager) update everywhere replication. It encompasses techniques like distributed locking and mechanisms based on total order broadcast, and timeout (e.g., [135]). SMR is a form of active replication. However, it can also be adjusted to order operations that are executed first (like for instance in the database state-machine approach [120]). As common with optimistic replication, when a conflict occurs, the operations are aborted and have to re-execute. Of course, this technique weakens the programming model (e.g., I/Os are not always possible). Contrary to the previous approaches, lazy replication [55] chooses availability over data consistency in the CAP trade-off [115]. The system answers to the client as soon as possible, and the data replicas are reconciliated in the background. For conciseness, this thesis does not cover weak consistency in detail.

The classification in [100] does not mention PR directly. Instead, it considers variations of the aforementioned techniques when operations are transactions and cannot execute at a single node.

Interestingly, Wiesmann et al. [100] distinguish database and distributed systems. They argue that replication in distributed systems is to ensure fault-tolerance, while in database the objective is to boost performance. With the arrival of the NoSQL movement and cloud infrastructures, both concerns matter. Wiesmann and Schiper [137] compare the aforementioned replication techniques in practice. The authors show that distributed locking does not scale with the conflict rate—as previously diagnosed in [79]. Techniques based on group-communication primitives (e.g., [97, 120]) are the most efficient. We will come back to the pivotal role of such primitives in modern replicated systems shortly.

The work in **[260]** presents a framework to decompose deferred update replication (DUR). DUR encompasses all the techniques where transactions are executed then their updates propagated to be applied upon commit. DUR improves parallelism because transactions may execute concurrently at several replicas. This is particularly beneficial to long transactions (e.g., a stock query). On the other hand, as DUR is inherently optimistic, it is efficient when the workload exhibits a low to medium conflict rate. When the propagation of updates is ordered, DUR permits to at least one conflicting transactions to survive and commit. This is better than aborting unilaterally conflicts, as in e.g., [119]. In the parlance of [172], such an approach is more permissive to correct interleavings of transactions.

In **[208]**, the authors coin the term *genuine partial replication*. Genuineness characterizes protocols in which only the replicas that store items accessed by the transaction take steps to order it. This prevents bottlenecks to appear in the system, ensuring that such protocols scale better. We will come back to this notion in greater detail in §3.3.3.

3.1.5 Modern designs

PR protocols propel the data storage systems at core of nowadays computing infrastructures. They build upon a vast prior art, stretching over a period of roughly 40 years, bridging lessons from the past and nowadays requirements. In particular, most of these protocols are designed with the following concerns in mind:

(**Elastic Shared-nothing**) Commodity servers are the base building blocks of current computing infrastructures. Most data sets (e.g., AI, OLTP) fits in the main memory of these systems, which now have a few TBs per machine. PR protocols scale up and down on such infrastructures, depending on demand.

(**Partial synchrony**) In these systems, synchrony assumptions are difficult. This comes from the complexity of the software stack and the use of commodity hardware. Moreover, protocols are using general-purpose language runtimes and operating systems that are not made for real-time.

(**Non-interactive workload**) Application workloads are now automated with very few human input. As a consequence, transactions are often non-interactive. For instance, Kallman et al. [177] identifies a large class of real-world applications that are made up of one-shot and/or two-phase transactions. One-shot means that the transaction can be split into independent chunks that run in parallel (that is, there is no dependencies among the chunks). Two-phase means that the transaction can be (re-)written as a sequence of reads followed by a sequence of writes: Phase one is a collection of queries. Based on the result of these queries, the transaction may be aborted. If it does not, phase two is executed which consist in queries and

updates where there is no possibility of an integrity violation. Figures 3.1b and 3.1e illustrate respectively a one-shot and a two-phase transaction.

(**Geo-replication**) The very first replication protocols for the early packet-switching networks are geo-distributed (such as [6]). This comes from the fact that these infrastructures were connecting machines in different locations. Current trends in distributed applications require always-on distributed services. As a consequence, it is now common to have storage systems stretched over several geographical sites, to tolerate failures (or the maintenance) of a full datacenter. One talks about geo-replication. A canonical example of such systems is Google Spanner [229].

Part of these design decisions are discussed in [175]. Its authors investigate the architecture of relational databases from the 90s, and show that they do not match recent needs, workloads and hardware. In particular, Harizopoulos et al. [175] observe that typical applications have large online transaction processing (OLTP) volumes. These transactions are non-interactive, short-lived and access small amounts of data per transaction. They also use heavily indexed lookups (scans are rare), and the database schema is often a star or a snowflake.

These design considerations are key in the birth of the NoSQL movement, started at the end of the 2000's. Many storage systems remove the one-size-fits-all SQL interface in favor of better scalability and higher availability. They offer limited transactionality, such as single-row transactional, or no transaction at all. Early examples of such NoSQL systems are Amazon's Dynamo [154], MongoDB [201], Megastore [214], BigTable [143] and PNUTS [169].

Around the same period of time, the distributed systems community introduces the first practical systems relying on (partially-synchronous) state-machine replication. Coordination kernels, such as Chubby [142] and ZooKeeper [203], are designed with the Cloud in mind. These highly-available systems simplify the task of coordination in replicated and highly-parallel systems (e.g., Yarn [268] and HDFS [244]). Coordination kernels have evolved today into cloud-native solutions such Etcd [381] and Atomix [376] that are built atop Raft [269] and Kubernetes [285]).

3.1.6 Recent solutions

Today, PR protocols follow most of the above design decisions. They are constructed from the experience accumulated at the end of the 2000's when, carried by the NoSQL movement, a blossom of new storage systems appeared. At core, these protocols rely on group-communication primitives for fault-tolerance and conflict resolution. In the spirit of Wiesmann et al. [100], we may consider the following two (coarse-grained) classes of solutions:

(Execute-then-Order.) This first class of solutions implements deferred update replication (DUR). In a nutshell, DUR works as follows: A transaction executes tentatively at one or more data replicas, buffering writes. Once the transaction commits, and if it succeeds, updates are applied across the system. Below, we detail some prominent PR protocols that follow such an approach.

Serrano et al. [161] depict a DUR protocol that supports interactive snapshot-isolated (SI) transactions. When a transaction starts, it gets assigned a timestamp set to the value of the local clock. This timestamp is used to read consistent versions when executing reads. Each new operation is forwarded to an appropriate replica where it executes. In case the operation is a write, it starts a dummy transaction at the replica. Upon commit, a read-only transactions commits right away, while an update one is atomic broadcast to all the replicas. When delivering such a transaction, a replica performs a certification test and decides locally to commit/abort the transaction. If the transaction

commits, the associated dummy transaction commits and the local clock is incremented. In [181], the authors refine this protocol to make it fault-tolerant. Fault-tolerance is ensured by delegating commit decision to a distinguished replica, called the certifier. SIPRe [165] bears similarities with the previous approach: When a transaction starts, a message is atomically broadcast to all the replicas. This message defines a consistent snapshot for the transaction. Like the above two solutions, a read-only transaction commits locally without any synchronization. An update transaction is atomic broadcast to all replicas when it commits, where it is certified locally.

GMU [241] enforces Extended Update Serializability (EUS), a consistency criterion where updates are serializable but queries may disagree on the order in which they take place. To achieve this, GMU employs a particular vector clock to compute consistent snapshots for read operations. Read-only transactions commit locally, and GMU commits update transactions with 2PC; all replicas holding an item read or written by the transaction participate in 2PC. P-store **[208]** supports interactive transactions. Reads may access any replica holding the appropriate data item. Updates are buffered locally. Upon commit, the transaction is atomic multicast to all the replicas holding a data item accessed by the transaction. These replicas votes to commit the transaction if it reads no outdated data; otherwise it aborts.

Scatter [219] organizes groups of data replicas as a distributed hash table (DHT) [111]. Each group runs Paxos to maintain its integrity and consistently replicate data. Inside a group, the leader define primaries that can read locally data with leases. Depending on the load of the subset of keys a group replicates, it can split, merge with another group or be re-partitioned. A transactions that touches several data replica groups is implemented with two-phase commit (2PC). Because there is no concurrency control for these transactions, they execute at the read-committed level of isolation.

Walter [225] generalizes SI to drop the necessity to see globally-monotonic snapshots—a consistency criterion named parallel snapshot isolation (PSI). Internally, this PR protocol uses a primary-copy approach and 2PC. Walter is tailored for geo-replication, each data item having a primary site. When a transaction starts, the local site assigns a vector of timestamps to the transaction to define its snapshot. An update transaction commits either in the fast or the slow path. If the transaction only modifies local items, it uses the fast path and commits if there is no write-conflicts with an already committed transaction. Otherwise, the transaction takes the slow path and Walter executes a (geo-distributed) 2PC protocol. Once a transaction is committed, it is propagated in the background to all the concerned replicas before it becomes visible.

Jessy **[246]** weakens PSI to permit reading versions created after the start of the transaction. The associated consistency criterion is called non-monotonic snapshot isolation (NMSI). To commit a transaction, Jessy uses atomic multicast followed by a voting phase. (A 2PC-based variation is also presented and evaluated in **[260]**).

Before going further, we note here that there are vivid debates [274, 307, 374] over the definition of snapshot isolation (SI), its generalization (GSI), and the more permissive versions that are PSI and NMSI. This comes from the fact that various specifications and/or variations have been proposed over the years, with different system models in mind. Elle [342] is a software based on Jepsen [386] to verify that a database implements a given transactional consistency criteria. The verification uses the graph-based specification of a consistency criterion proposed by Adya [89].

Spanner [229] is a geo-distributed transactional data store that follows the DUR approach. Each partition (tablet in [229]) is replicated with Paxos. When a transaction is submitted, Spanner timestamps it using synchronized clocks that rely internally on the GPS signal (TrueTime API). Writes are buffered locally while reads happens at the leader replica (of the concerned partition),

taking locks there. At commit time, Spanner executes a fault-tolerant 2PC among the Paxos leaders of the partitions concerned with the transaction. During 2PC, each participants locks the items updated by the transactions. The protocol then assigns a second timestamp to the transaction. Commit happens once this timestamp has passed everywhere in the system. If the transaction commits, its updates are propagated and applied at the appropriate partitions. The timestamps also allow to execute lock-free (strongly-consistent) queries.

Researchers [328, 357] have studied how to reduce the time complexity at commit time in Spanner. The design of Spanner has also inspired many follow-up works, some of them reaching the level of industrial-grade systems such as YugabyteDB [396]. CockrochaDB [348] relies on Raft (instead of Paxos) to implement each partition. Much like Spanner, this system uses leases to execute fast strongly-consistent reads at the leader. For transactions crossing partitions, Cockroach offers serializability, but not strict serializability as Spanner. Nonetheless, clients that stick to the same replica are ensured to see transactions in the order they execute them (that is, they have session guarantees [70]).

(Order-then-Execute.) This second class of PR protocol orders transactions *a priori*, before attempting to execute them at the concerned partitions.

Built atop the seminal observations in [175], H-Store [177] kick-starts the NewSQL movement at the end of the 2000s. This database system is designed at core for in-memory computing atop a cluster of shared-nothing machines. Transactions are non-interactive, written as a sequence of stored procedures, each targeting a single partition. At each machine, a partition is assigned to a single-threaded execution engine, run by one (or more) CPU cores. As noted by Cowling and Liskov [230], the original H-Store paper [177] does not provide a functional protocol for coordinating transactions. In a follow-up work, Jones, Abadi, and Madden [204] describe a PR protocol that relies on a central coordinator to order and forward transactions to the appropriate partitions. Each partition is replicated using primary-backup, and commit/abort happens using two-phase commit (2PC). This protocol logic is implemented in the C++ prototype [383].

A second example of such protocols is Calvin [245]. Calvin offers to the programmer a non-interactive transactional API for two-phase transactions. At regular intervals, input transactions are ordered with the help of a deterministic merge function, named sequencer. This sequencer is implemented as a fault-tolerant service atop Paxos. The merge output defines an order used to deterministically execute the different parts of the transaction across the system. At each partition, execution is identical thanks to a deterministic locking manager. The design of FaunaDB [394], a relational geo-distributed database, follow the one of Calvin.

Granola [230] resembles to Calvin in the sense that the system is also tailored for two-phase transactions. However, this system is even more restrictive, as it solely supports one-shot transactions. This means that there is no dependency among transaction chunks, except at most a single abort/commit message. This class of transactions is illustrated in Figure 3.1b. In Granola, each transaction is timestamped using a Skeen-like atomic multicast protocol: each partition computes a timestamp for the transaction, and the highest one among all is chosen. This schema is made fault-tolerant with Paxos, in a manner similar to [104] and **[195]**.

CorfuDB [249, 250, 313] is a distributed storage that provides strongly-consistent shared objects. Objects can be composed into transactions that are executing against multiple partial replicas. Transactions are opaque [174], that is a transaction reads always a consistent snapshot even if it aborts, and committed transactions form a sequential execution that respects real time. CorfuDB offers a convenient Java API that allows the programmer to indicate whether a method is a mutator,

an accessor, or a mix of the two. This indication is used to execute client operations at lower cost, e.g., using leases for reads (accessors). The PR protocol in CorfuDB relies on a central sequencer to order operations. In spirit, this is similar to timestamp-based transactional memory, such as TinySTM [171]. CRESON **[311]** is a a storage system whose functionalities resemble to the ones of CorfuDB. It permits to create custom shared Java objects, compose them, and decorate their methods that are accessors for higher performance. Each object is linearizable and partially replicated for fault-tolerance. Internally, CRESON relies on SMR which is implemented atop a novel NosQL abstraction called listenable key-value store.

Janus [296] extends the leaderless Egalitarian Paxos (EPaxos) protocol to the PR case. This protocol targets one-shot transactions, written as stored procedures, similarly to Granola. To commit a transaction, Janus runs the same steps as EPaxos at each partition. Upon commit, the protocol computes the transitive closure of the dependency relation across all partitions, to complete locally the dependency graph. Transactions are run like in EPaxos, that is in the order defined by the graph, synchronizing partitions where necessary. The authors of Janus [296] compare extensively their protocol against prior art, including MDCC [254], Tapir [282] and the approach in Spanner, that is 2PC over Paxos. They show that Janus is consistently faster in a range of micro-benchmarks and benchmarks (including TPC-C [392])

Summary

For about two decades, storage systems (transactional or not) are built atop shared-nothing infrastructures of commodity servers. These systems replicate data for higher availability and fault-tolerance sometimes over several geo-distributed locations. Strong consistency is becoming a de facto standard in these systems as it simplifies the life of the programmer. It is implemented with the help of a replication protocol that relies on some form of state-machine replication (SMR), using underlying deterministic components (e.g., a single threaded execution in H-Store [177]). Partial replication (PR) is used both to increase performance by parallelizing accesses, and to store more data. When consistent operations across partitions are required, storage systems implement transactions. Modern applications (e.g., OLTP) that use these systems execute mostly non-interactive transactions.

3.2 Partial State-Machine Replication

From a system perspective, we notice a discrepancy between the single and multi-partition case: SMR defines a replication protocol, upon which one can implement linearizable objects, while transactions are client-level abstractions. In other words, we lack an intermediary abstraction for PR that would help to build correct-by-design protocols. In this section, we try to bridge this gap with the notion of *partial* state-machine replication.

3.2.1 Definition

Partial state-machine replication (PSMR) is defined in **[350]**. In this generalized version of SMR, each process replicates only part of the service state. With more details, the service state is divided into *partitions*, so that each variable defining the state belongs to a unique partition. Partitions are arbitrarily fine-grained; for instance, just a single state variable. Each command accesses one or more partitions. A process replicates a single partition, but multiple processes may be co-located at the same machine. We write $\mathcal{P}_{\mathsf{p}}$ for the set of all the processes replicating a partition p, $\mathcal{P}_{\mathsf{c}}$ for

the set of processes that replicate the partitions accessed by a command c, and $\mathcal{P}$ for the set of all processes.

A PSMR protocol allows a process i to submit a command c on behalf of a client. For simplicity, we assume that each command is unique and the process submitting it replicates one of the partitions it accesses: $i \in \mathcal{P}_{\mathsf{c}}$. For each partition p accessed by c, the protocol then triggers an upcall $execute_{\mathsf{p}}(c)$ at each process storing p, asking it to apply c to the local state of partition p. After c is executed by at least one process in each partition it accesses, the process that submitted the command aggregates the return values of c from each partition and returns them to the client.

PSMR ensures the highest standard of consistency of replicated data – *linearizability* [45] – which provides an illusion that commands are executed sequentially by a single machine storing a complete service state. To this end, a PSMR protocol has to satisfy the specification that follows. Given two commands c and d, we write $\mathsf{c} \mapsto_i \mathsf{d}$ when they access a common partition and c is executed before d at some process $i \in \mathcal{P}_{\mathsf{c}} \cap \mathcal{P}_{\mathsf{d}}$. We also define the following *real-time order*: $\mathsf{c} \rightsquigarrow \mathsf{d}$ when c returns before the command d was submitted. Let $\mapsto \; = (\bigcup_{i \in \mathcal{P}} \mapsto_i) \cup \rightsquigarrow$. A PSMR protocol ensures the following properties:

(**Validity**) If a process executes some command c, then it executes c at most once and only if c was submitted before.

(**Ordering**) The relation $\mapsto$ is acyclic.

(**Liveness**) If a command c is submitted by a non-faulty process or executed at some process, then it is executed at all non-faulty processes in $\mathcal{P}_{\mathsf{c}}$.

Ordering ensures that commands are executed in a consistent manner throughout the system. For example, it implies that two commands, both accessing the same two partitions, cannot be executed at these partitions in contradictory orders.

As expected, when there exists a single partition, PSMR boils down to SMR. Indeed, the difference with §2.1 is purely cosmetical. Instead of appending a command to the local copy of the log, it can be executed. Conversely, calling $execute(\mathsf{c})$ in PSMR can append c to a local log. In more formal terms, the two abstractions are strongly equivalent: local computation suffices to reduces one to another. (We write $A \preceq B$ when we may use B to implement A with only local computation. A and B are strongly equivalent when $A \preceq B$ and $B \preceq A$.)

PSMR allows to process commuting commands in parallel in many cases, e.g., if two commands access unrelated data items in a storage system. However, PSMR does not capture commutativity in general. For instance, two concurrent increments over the same variable are needlessly ordered by PSMR. A simple fix is to modify the definition of $\mapsto$ to track conflicts, as with LSMR (§2.2).

3.2.2 An example

To illustrate the above, consider Infinispan [239]. This key-value store implements a concurrent map: operation $\mathtt{put}(k, x)$ associates x to the key k, $\mathtt{get}(k)$ returns the last value associated with k, and $\mathtt{remove}(k)$ deletes any association with k. Each PSMR partition corresponds to a namespace, or *cache* in Infinispan parlance. Within a given cache, operations are executed exactly once, and in the same order at all its replicas. These two properties are captured by the PSMR abstraction.

Infinispan also supports the (so-called) distributed caches, where data within a cache is replicated following a consistent hashing strategy [81]. Because partitions are arbitrarily fine-grained, PSMR can also model this configuration of the system. In that case, a key corresponds to one partition.

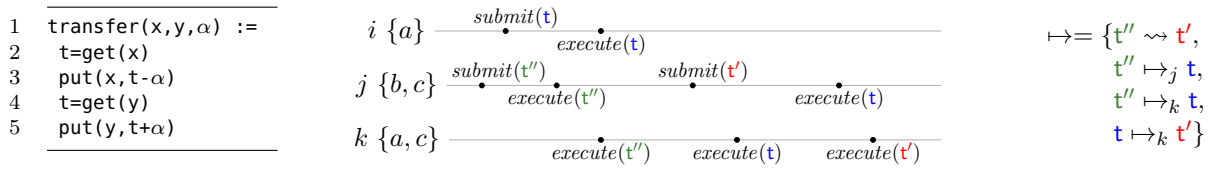


Figure 3.2: Using PSMR in a banking application. From left to right, the command `transfer`(x, y, α), an example run, and the order relation ($\mapsto$) in this run.

Additionally to the CRUD operations over a map, Infinispan provides transactions—including JTA ones from the J2EE standard [257]. Figure 3.2 illustrates such transactions in the context of a simple bank application. Operation `transfer`(x, y, α) moves an amount α of some asset from account x to y. Implementing this operation in Infinispan in depicted in Figure 3.2(left). Figure 3.2(middle) presents a (failure-free) run during which three concurrent transfers take place: $\mathtt{t} =$ `transfer`$(a, b, \$10)$, $\mathtt{t}' =$ `transfer`$(b, c, \$5)$, and $\mathtt{t}'' =$ `transfer`$(c, a, \$3)$ These transactions are executed in different orders at the processes i, j, and k. Process i replicates account a, j replicates both b and c, while k replicates a and c. The ordering relation $\mapsto$ in this run appears in Figure 3.2(right). Since $\mapsto$ is acyclic, this (partial) run is sound for the PSMR abstraction.

3.2.3 Programming with PSMR

PSMR is expressive enough to cover a wide spectrum of programming models. Figure 3.2(left) is an example of *one-shot* transaction. This type of transaction consists of independent pieces of code, such as stored procedures, each accessing a different partition [267]. A typical example of one-shot transactions is a query against multiple data structures (e.g., a set of relational tables) [177]. A batch of updates is another form of one-shot transaction. Figure 3.1c illustrates this case with the bulk write API of MongoDB [201, 385]. One-shot transactions are frequently executed in modern web services [295]. Implementing this class of transactions with PSMR is straightforward: each command defines the piece of code it executes at a given partition.

In [185], the authors introduce mini-transactions which consists of reads followed by conditional writes. All the conditions must be true for the writes to execute. Lightweight transactions in Apache Cassandra [205] are a similar abstraction written in the Cassandra Query Language (CQL). Figure 3.1f illustrates this class of transactions using CQL. Each statement is restricted to a single partition. These transactions provide a restricted form of conditional updates to the programmer. Notice that for both types of transactions (mini and lightweight) data items are known in advance by looking at the source code; that is, these transactions are non-interactive. PSMR may also implement non-interactive transactions. In that case, as explained in [296], the execution of a transaction at each partition is deferred until its dependencies (e.g., the content of a variable) are known.

The most general type of transactions are *interactive*. In that case, the data items are not known in advance, as illustrated in Figure 3.1a. It is possible to rewrite transactions of this type into non-interactive ones, using the technique of Thomson and Abadi [212]. Figure 3.1d illustrates the transformation of Figure 3.1a into a sequence of non-interactive transactions.

Starting with sagas [30], many proposals exist to transform, or *chop*, transactions [75]. In Figure 3.1d, the base idea is to execute the transaction optimistically, buffering its output locally—this first transaction is called a reconnaissance query in [245]. Once the transaction finishes, the

side effects are applied at the appropriate partitions, provided that none of the data items the transaction read in the first phase is outdated. This is the classical termination phase in deferred-update replication **[260]**. If any such read fails, the transaction aborts, which usually leads to a retry. Notice that such a transformation does not work when the transaction contains I/Os. This limitation also exists with hardware transaction, e.g., with Intel TSX [393]. In such a case, transactuations [343] offer a possible solution where writes to the outside world are executed after commit.

3.2.4 Existing solutions

As reviewed above, there are a plethora of distributed transactional systems. Some of them, e.g., Granola [230] and Calvin [245], implement PSMR for restricted classes of state-machine commands. More recently, Bezerra, Pedone, and Renesse [263] proposes a replication protocol that relies on atomic multicast. This protocol works in two phases: During the first phase a command c is atomic multicast to all replicas in $\mathcal{P}_\mathsf{c}$. Upon delivery, each replica executes c at its local partitions according to the command's logic. In case c requires data from a different partition, the replica blocks expecting an appropriate message from a distant replica. When c terminates its execution, the replica awaits that, for every partition p of c, a replica in $\mathcal{P}_\mathsf{p}$ also finishes executing c. This signaling phase is necessary to maintain linearizability—more details about this shortly.

As mentioned in §3.1.6, Janus [296] adapts EPaxos to the context of PR. To execute some command c, *coord*(c) sends c to all the partitions it accesses in the pre-accept phase. In each partition p, a fast quorum in $\mathcal{P}_\mathsf{p}$ tries to agree optimistically on the value of *dep*[c] in p. If this fails at *any* partition, *coord*(c) executes the accept phase of Paxos at all partitions. Agreement takes place on the union of the dependencies returned in the pre-accept phase by some quorum in each partition. Once committed, a multi-partition command awaits that it forms a strongly-connected component in the dependency graph. Notice that at a replica, the transitive closure of a component might require to inquiry replicas outside of $\mathcal{P}_\mathsf{c}$. The work in **[182]** uses a similar technique for sharded databases: Each transaction execute optimistically, buffering writes (DUR). The transaction is then atomic broadcast within *each* of its partitions. A partition maintains locally a serializability graph. To certify the transaction, the transitive closure of its serializability graph is computed across all partitions.

Tempo **[350]** is an LSMR protocol specifically tailored for PR. The single partition case was covered in §2.2.5. With multiple partitions, Tempo starts similarly to when there is a single-partition, except that the protocol now uses one coordinator per partition. Once the command is committed at each partition, replicas exchange the corresponding timestamps. The command's final timestamp is computed as the maximum of these timestamps. A command executes once it is stable at all the partitions it accesses. As in the single partition case, commands are executed in timestamp order.

Built upon Atlas **[337]**, Tempo is efficient when simultaneous failures are rare. In Tempo, each partition is replicated at r processes, of which at most f may fail. Following Flexible Paxos [289], f can be any value such that $1 \leq f \leq \lfloor \frac{r-1}{2} \rfloor$. This allows using small values of f regardless of the replication factor r, which is appropriate in a geo-distributed setting [229, 350]. Tempo uses fast path quorums of $\lfloor \frac{n}{2} \rfloor + f$ processes. As detailed in §2.2.5, such an assumption makes sense when data replicas are independent (e.g., in a geo-distributed system). In this context, failures are uncorrelated and considering small values for f makes sense.

Accord [377] is built with different assumptions in mind, where replication happens also within

each data center. Consequently, it matters that the protocol survives a higher number of faults. Accord is the new replication layer at core of the upcoming version 5 of Apache Cassandra, a popular distributed data store [205]. The protocol achieves the highest availability ($f < r/2$ per partition) thanks to a complex recovery procedure, akin of EPaxos'. As indicated in §2.2.5, a bug was found in an early version of this procedure **[390]**.

3.2.5 Related work

In the definition of PSMR, a command is projected over a partition, where it defines its effects there. For instance in Figure 3.2(left) , `transfer`(x, y, α) projected over account a leads to: $t \leftarrow$ `get`(a); `put`$(a, t+\alpha)$. A dual take is to consider that a command is composed of a sequence of sub-commands, each applying to a partition. This is the proposal of Marandi, Primi, and Pedone [222], in which they define the notion of state partitioning ordering.

There is a small mistake in [222] that is corrected in **[266]**. This paper also defines the notion of *consistent partitioning* for a service. It formulates two theorems that state when a partitioning is consistent, that is when commands composed of sub-commands are linearizable. The first theorem is simply a re-writing of the locality property of linearizability: if there is no invariant linking partitions, then any interleaving of the sub-commands is correct. The second theorem relies on the notion of semantic graph. This graph generalizes the serializability graph in database theory: For some history h, the nodes in this graph are the commands in h. An edge exists between `c` and `d` either if `c` precedes (in real-time) `d` in h, or some sub-command of `c` precedes some sub-command of `d` and it does not commute with it. The theorem establishes that a partition is consistent when for every history h, there exists a linearization l such that the semantic graph induced by l is acyclic. This second theorem is used to construct Zoofence, a partitioned Zookeeper service.

Service composition for coordination kernel services is also studied in ZooNet [292] and WanKeeper [301]. A follow-up work of [222] is [263], in which the authors define the notion of scalable state-machine replication (S-SMR). [326] investigate the problem of dynamically partitioning S-SMR and [373] how to apply this idea to disaggregated memory systems. Unfortunately, S-SMR is not an intermediary abstraction but an actual algorithm. In all of these follow-up work, proving a protocol requires to show that it implements SMR.

Summary

SMR is a widely-used notion in distributed storage systems. PSMR offers a similar intermediary abstraction when data is partially replicated (sharded). In PSMR, partitions are arbitrarily fine-grained, which permits to capture parallelism in the system. Compared to prior works such as serializability theory, PSMR also models durability and progress. PSMR is already at work in many storage systems that implement an Order-then-Execute approach.

3.3 Resolubility

SMR reduces to atomic broadcast, which itself reduces to consensus. In the failure detector model, consensus requires the Σ and Ω failures detectors to be solvable when processes crashes. This section answers the same question for PSMR, by first reducing it to the atomic multicast problem.

3.3.1 Atomic multicast

Atomic multicast (AMcast) is a group communication primitive that allows to disseminate messages between distributed processes. As mentioned earlier, this primitive is used to build transactional systems, such as P-store **[208]**. In what follows, we consider the most standard definition for this problem [49, 66, 122]. In the parlance of Hadzilacos and Toueg [66], it is named uniform global total order multicast. Other variations are detailed later.

Given a set of messages $\mathcal{M}$, the interface of atomic multicast consists of operations $multicast(m)$ and $deliver(m)$, with $m \in \mathcal{M}$. Operation $multicast(m)$ allows a process to *multicast* a message m to a set of processes denoted by $dst(m)$. This set is named the *destination group* of m. When a process executes $deliver(m)$, it delivers message m, typically to an upper applicative layer. Consider two messages m and m' and some process $i \in dst(m) \cap dst(m')$. Relation $m \overset{i}{\mapsto} m'$ captures the local delivery order at process i. This relation holds when, at the time i delivers m, i has not delivered m'. The union of the local delivery orders gives the *delivery order*, that is $\mapsto = \cup_{i \in \mathcal{P}} \overset{i}{\mapsto}$. The runs of atomic multicast must satisfy:

(**Integrity**) For every process i and message m, i delivers m at most once, and only if i belongs to $dst(m)$ and m was previously multicast.

(**Termination**) For every message m, if a correct process multicasts m, or a process delivers m, eventually every correct process in $dst(m)$ delivers m.

(**Ordering**) The transitive closure of $\mapsto$ is a strict partial order over $\mathcal{M}$.

Integrity and termination are two common properties in group communication literature. They respectively ensure that only sound messages are delivered to the upper layer and that the communication primitive makes progress. Ordering guarantees that the messages could have been received by a sequential process.

If the sole destination group is $\mathcal{P}$, that is the set of all the processes, the definition above is the one of atomic broadcast.

3.3.2 Relation with PSMR

With an appropriate mapping between partitions and destination groups, atomic multicast and PSMR are strongly equivalent. In detail, atomic multicast reduces to PSMR when for every partition p there exists a destination group g with $g \subseteq \mathcal{P}_\mathsf{p}$. Conversely, if for each partition p, $\mathcal{P}_\mathsf{p}$ is a subset of some destination group g, one may solve PSMR with atomic multicast. However there is a subtlety. As observed by Bezerra, Pedone, and Renesse [263], the common definition of atomic multicast is not strong enough for this second reduction: if some command d is submitted after a command c get delivered, atomic multicast does not enforce c to be delivered before d, breaking linearizability. To sidestep this problem, a stricter variation must be used. Let us write $m \rightsquigarrow m'$ when m is delivered in real-time before m' is multicast. Atomic multicast is *strict* ($\text{AMcast}_{\text{strict}}$) when ordering is replaced with:

(**Strict Ordering**) The transitive closure of $(\mapsto \cup \rightsquigarrow)$ is a strict partial order over $\mathcal{M}$.

Notice that strictness is free when there is a single destination group. Indeed, if i delivers m before j broadcasts m', then necessarily $m \overset{i}{\mapsto} m'$. This explains why atomic broadcast does not mention such a property.

Proposition 1. *PSMR and AMcast$_{strict}$ are strongly equivalent.*

Proof. (PSMR $\preceq$ AMcast$_{\text{strict}}$) For each command c, we define a message m_{c} that contains c in its payload and $dst(m_{\mathsf{c}}) = \mathcal{P}_{\mathsf{c}}$. Upon submitting c, m_{c} is multicast. When this message is delivered, c is unpacked from m_{c} and applied at the local partition. To see why such an implementation is correct, observe that if $\mathsf{c} \mapsto_i \mathsf{d}$ then $m_{\mathsf{c}} \overset{i}{\mapsto} m_{\mathsf{d}}$. Indeed by definition of relation $\mapsto_i$, commands c and d have a common partition, say p, and at process $i \in \mathcal{P}_{\mathsf{p}}$, c is executed before d. Hence, we have $i \in dst(m_{\mathsf{c}}) \cap dst(m_{\mathsf{d}})$, and m_{c} is delivered first at process i. Similarly, we may establish that: $(\mathsf{c} \rightsquigarrow \mathsf{d}) \Rightarrow (m_{\mathsf{c}} \rightsquigarrow m_{\mathsf{d}})$. From what precedes, by the acyclicity of $(\mapsto \cup \rightsquigarrow)$ over $\mathcal{M}$, relation $(\mapsto \cup \rightsquigarrow)$ over Cmd is also acyclic. This shows that the ordering property of PSMR holds. Validity and liveness then follow respectivly from the integrity and termination properties of AMcast$_{\text{strict}}$.

(AMcast $\preceq$ PSMR) For each message m, we define a command c_m. The state machine is an automaton that returns message m upon executing command c_m. Let $\mathcal{G}$ be all the destination groups induced by the messages in $\mathcal{M}$. We consider a partitioning function $\mathfrak{P}(\mathcal{G}) \subseteq 2^{\mathcal{P}}$ such that: *(i)* for every destination group $g \in \mathcal{G}$, there exists $(g_i)_i \subseteq \mathfrak{P}(\mathcal{G})$ with $g = \cup_i g_i$, *(ii)* for any two g, h in $\mathfrak{P}(\mathcal{G})$, $g \cap h$ is empty. A process i replicates the (at most single) partition it belongs to. Command c_m accesses all the partitions induced by its destination group. When m is multicast, c_m is submitted to the replicated state machine. Message m is delivered locally when command c_m gets executed. The correctness of such a construction is established as follows: If $m \overset{i}{\mapsto} m'$ then i delivers m first from the state machine. Let $g \in \mathfrak{P}(\mathcal{G})$ be the partition i belongs to. Partition g is accessed by both c_m and $\mathsf{c}_{m'}$, leading to $\mathsf{c}_m \mapsto_i \mathsf{c}_{m'}$. Such a relation over the messages also holds wrt. real time, that is $(m \rightsquigarrow m') \Rightarrow (\mathsf{c}_m \rightsquigarrow \mathsf{c}_{m'})$. Consequently, the strictness of the delivery order is a consequence of the ordering property of PSMR. Integrity and termination are also maintained by the PSMR abstraction and our mapping from messages to commands. □

From what precedes, we may study the resolubility of atomic multicast in lieu of PSMR. This is the approach we follow hereafter.

3.3.3 Genuineness

At first glance, atomic multicast boils down to the atomic broadcast problem: to disseminate a message it suffices to broadcast it, and upon reception only messages addressed to the local machine are delivered. With this approach, every process takes computational steps to deliver every message, including the ones it is not concerned with. (As expected, the very same observation can be made with SMR and PSMR.)

In practice, the above approach does not scale: when messages are addressed to small destination groups in a large distributed system, atomic broadcast bottlenecks **[195]** [238]. Guerraoui and Schiper [104] defines a minimality property to rule out such solutions. In detail, A is a *genuine* atomic multicast protocol when it satisfies:

(**Minimality**) In every run ρ of A, if some correct process i sends or receives a (non-null) message in ρ, there exists a message m multicast in ρ with $p \in dst(m)$.

This definition is appealing because it is unambiguous in a formal sense. In particular, it rules out naive approach to solve the problem using atomic multicast. However, we also observe that once a process is in the destination group of a message, it may start helping others. Thus, from a performance perspective this is not fully satisfactory.

In **[208]**, the authors try to address this shortcoming with the notion of *genuine partial replication*. Similarly to [104], they try to capture the parallelism of a solution to implement PR. The definition is given for distributed transactional systems that implements DUR. This means that a transaction executes first without concurrency control then it is certified during a termination phase.

(**Genuine PR**) For any submitted transaction T , only database sites that replicate data items read or written by T exchange messages to certify T.

The work of **[350]** introduces a similar definition in the context of PSMR.

(**Genuine PSMR**) Only the processes in $\mathcal{P}_{\mathsf{c}}$ take steps to order and execute command c.

At first glance, the above two definitions look better than [104]. Unfortunately, both **[208]** and **[350]** lack a proper formalization.

A possible solution is to use the causal past and future operators introduced in **[345]**. In detail, let $\preceq$ and $\succeq$ be respectively the reflexive closure of the happen-before relation ($\prec$) and the reflexive closure of the converse of $\prec$. Then, for some run ρ, we note $(\rho|\mathsf{c})$ the sub-sequence of steps in ρ to submit command c and execute it at the coordinator, that is $(\rho| \succeq \mathit{submit}_{\mathit{coord}(\mathsf{c})}(\mathsf{c})| \preceq \mathit{execute}_{\mathit{coord}(\mathsf{c})}(\mathsf{c}))$.

(**Genuine PSMR**) For every run ρ, for every command c, if $s \in (\rho|\mathsf{c})$ then $\mathit{proc}(s) \in \mathcal{P}_c$, where $\mathit{proc}(s)$ is the process taking step s.

Again, this is also not satisfactory. Indeed, some execution paths in $(\rho|\mathsf{c})$ may contain events related to the ordering of other commands than c.

A recent paper **[371]** introduces *strong genuineness* that captures precisely the desired parallelism in a genuine solution. This notion characterizes solutions to the atomic multicast problem as follow. A run ρ is fair for some correct process i when i executes an unbounded amount of steps in R. By extension, ρ is fair for $P \subseteq \mathit{Correct}(R)$, or for short P-fair, when it is fair for every i in P.

(**Group parallelism**) Consider a message m and a run ρ. Note $P = \mathit{Correct}(\rho) \cap \mathit{dst}(m)$. If m is delivered by a process, or atomic multicast by a correct process in ρ, and ρ is P-fair, then every process in P delivers m in ρ.

Group parallelism bears similarity with x-obstruction freedom [312], in the sense that the system must progress when a small enough group of processes is isolated. A protocol is said *strongly-genuine* when it satisfy both the minimality and the group parallelism properties. Similarly, strong genuineness can be defined in the context of PSMR. In this case, this captures that a command may progress with only the replicas of the partitions it accesses.

A notion related to genuineness is disjoint-access parallelism (DAP) [67]. DAP demands that processes operating on disjoint parts of the logical state do not interfere with each other. More precisely, it is defined in terms of a conflict graph whose nodes are the operations in an execution. An edge exists between two operations when they conflict, that is they access the same part of the logical state (such as the same item in a key-value store). Two processes contend when they access the same base object and one of these accesses is non-trivial (i.e., it modifies the state). DAP demands that processes contend only if their operations conflict, avoiding any need for synchronization [233]. Ellen et al. [231] prove that no universal construction can be both DAP and wait-free in the case where the implemented shared object can grow arbitrarily. Many other definitions of DAP exist,

Genuiness	Order	Weakest	
×	Global	$\Omega \wedge \Sigma$	[76, 124]
√	·	$\notin \mathcal{U}_2$	[104]
·	·	$\leq P$	[180]
·	·	μ	
·	Strict	$\mu \wedge (\wedge_{g,h \in \mathcal{G}}\ 1^{g \cap h})$	
·	Pairwise	$(\wedge_{g,h \in \mathcal{G}}\ \Sigma_{g \cap h}) \wedge (\wedge_{g \in \mathcal{G}}\ \Omega_g)$	**[371]**
√√	Global	**if** $\mathcal{F} = \varnothing$ **then** $\mu \wedge (\wedge_{g,h \in \mathcal{G}}\ \Omega_{g \cap h})$ **else** $\geq \mu \wedge (\wedge_{g,h \in \mathcal{G}}\ \Omega_{g \cap h})$	

Table 3.1: (from **[367]**) Weakest failure detector for atomic multicast. *(√√= strongly genuine)*

e.g., for transactional memory (TM) [173]. In [186], the authors prove that DAP is not attainable for TM when read-only transactions make progress without writing the shared memory. The PCL theorem [315] further generalizes this result to visible read-only (non-interactive) transactions, and a larger panel of consistency criteria. Following this line of works, **[248]** studies DAP in the context of distributed transactional systems. It is shown that DAP and SI are mutually exclusive. To achieve this, SI is decomposed into three key properties over transactional histories. **[248]** establishes that two of these properties cannot be maintained if the transactional system ensures DAP.

3.3.4 Minimal synchrony assumptions

Existing genuine atomic multicast algorithms that are fault-tolerant have strong synchrony assumptions on the underlying system. Some protocols (such as [180]) assume that a perfect failure detector is available. Alternatively, a common assumption is that the destination groups are decomposable into disjoint groups, each of these behaving as a logically correct entity (as in the proof of Proposition 1). Such an assumption is a consequence of the impossibility result established in [104]. This result states that genuine atomic multicast requires some form of perfect failure detection in intersecting groups. Consequently, almost all protocols published to date (e.g., [94, 106, 306, 325, 354]) assume the existence of such a decomposition.

A key observation in **[371]** is that the impossibility result in [104] is established when atomic multicast allows a message to be disseminated to *any* subset of the processes. However, when there is no such need, weaker synchrony assumptions may just work. For instance, when each message is addressed to a single process, the problem is trivial and can be solved in an asynchronous system. Conversely, when every message is addressed to all the processes in the system, atomic multicast boils down to atomic broadcast, and thus ultimately to consensus. Now, if no two groups intersect, solving consensus inside each group seems both necessary and sufficient. Following this line of thoughts, the work in **[371]** characterizes the necessary and sufficient synchrony assumptions to solve genuine atomic multicast. The results are established when the asynchronous message-passing system is augmented with unreliable failure detectors [77, 216].

In a nutshell, the characterization in **[371]** is as follows (see Table 3.1 for a summary): Let $\mathcal{G}$ be all the destination groups and $\mathcal{F}$ be the cyclic families in it, that is the subsets of $\mathcal{G}$ whose intersection graph is hamiltonian. The weakest failure detector to solve genuine atomic multicast is $\mu = (\wedge_{g,h \in \mathcal{G}}\ \Sigma_{g \cap h}) \wedge (\wedge_{g \in \mathcal{G}}\ \Omega_g) \wedge \gamma$, where Σ_P and Ω_P are the quorum and leader failure detectors restricted to the processes in P, and γ is a new failure detector that informs the processes in a cyclic family $\mathfrak{f} \in \mathcal{F}$ when $\mathfrak{f}$ is faulty.

Regarding strongly-genuine atomic multicast, **[371]** establishes that $\mu \wedge (\wedge_{g,h \in \mathcal{G}}\ \Omega_{g \cap h})$ is the

weakest when $\mathcal{F} = \varnothing$. The case $\mathcal{F} \neq \varnothing$ is a bit more intricate. First of all, **[367]** observes that in this case the problem is failure-free solvable: given a spanning tree T of the intersection graph of $\mathcal{G}$, we can deliver the messages according to the order $<_T$, that is, if m is addressed to g intersecting with $h, h', \ldots$ with $h <_T h' <_T \ldots$, then $g \cap h$ delivers first m, followed by $g \cap h'$, etc.[3] A failure-prone solution would apply the same logic. This is achievable using $\mu \wedge (\wedge_{g,h \in \mathcal{G}} \ \Omega_{g \cap h}) \wedge (\wedge_{g,h \in \mathcal{F}} \ 1^{g \cap h})$, where $g \in \mathcal{F}$ holds when for some family $\mathfrak{f} \in \mathcal{F}$, we have $g \in \mathfrak{f}$, and 1^P is the indicator failure detector that signals if all the processes in P are faulty or not. **(Q)** It is conjectured in **[367]** that this failure detector is the weakest.

Summary

PSMR is equivalent to the strict atomic multicast problem. As a consequence, the solvability of PSMR follows from the one of atomic multicast in partially-synchronous systems. There is a range of protocols to solve this last problem. A genuine protocol relies only on the destination group of a message to deliver it. Conversely, a non-genuine solution may use additional processes outside the destination group. A protocol is strongly-genuine when a message can progress with just the destination group taking steps. All of these approaches have distinct minimal synchrony assumptions.

[3] Strictly speaking, a spanning tree is required per connected component of the intersection graph.

Chapter 4

On Agreeing in Shared Memory

Previous chapters underline that distributed agreement (consensus) is a central problem when data is replicated. This chapter studies consensus, with a focus on the shared-memory model. First, we consider the conditions under which this task, and some of its key variations, are solvable. Then, when a solution exists, we discuss its cost in terms of time and space complexity. As previously, our contributions on these questions are put in perspective with respect to existing works.

4.1 Computability

4.1.1 The FLP impossibility

Charting the boundary between solvable and unsolvable tasks is a central topic in distributed computing literature. Maybe the most acclaimed result on this matter is the FLP impossibility result by Fischer, Lynch, and Paterson [21]. FLP [21] establishes that (binary) consensus is not possible even if a single process may fail in an asynchronous message-passing distributed system. The result of Loui and Abu-Amara [32] extends FLP to the shared-memory model. In this model, the impossibility also holds when $n > 2$ processes have access to test-and-set but only two values can be stored in a memory cell.

An interesting path to obtain FLP is the Borowsky-Gafni simulation, or BG simulation [58]. An algorithm is f-resilient, with $1 \leq f \leq n-1$, if it solves some problem even in executions where up to f processes crash. The BG simulation permits a $(f+1)$ distributed system that supports f failures to wait-free simulate a f-resilient system of $n > f$ processes. As a consequence, a proof about the impossibility of consensus in a 2-processes shared-memory system gives us the result in [32]; this is much simpler. In other words, the BG-simulation tells us that computability questions with f crash failures reduce to wait-free computability questions with $(f+1)$ processes.

In [43], the author introduces a natural generalization of consensus, called k-set agreement. In this task, each process proposes a value and it is required that each correct process decides on a value proposed by a process and at most k distinct values are decided. For $k = 1$, k-set agreement is exactly consensus, and for $k = n$, the number of processes in the system, k-set agreement is trivial, since every process can decide its own proposed value. Chaudhuri [43] shows that k-set agreement can be solved by a f-resilient asynchronous algorithm, when $f < k$. This means that if the number of failures is strictly smaller than the number of possible decision values, then k-set agreement is

solvable. Chaudhuri [43] also conjectured that k-set agreement is unsolvable if $f \geq k$. For the case $k = 1$, this conjecture matches FLP, namely, 1-set agreement, with a single crash failure. Notice that for the wait-free case, i.e., $f = n - 1$, this conjecture says that only the trivial n-set agreement task has a wait-free solution. [58], [61], and [63] prove the wait-free case of Chaudhuri's conjecture. These three papers all use topological arguments, linking the fields of algebraic topology and distributed computing [90]. (This nature is "palpable" in earlier works that rely on combinatorial arguments, such as [48].) The work in [58] also proves that the f-resilient case follows from the impossibility of the wait-free case. To this end, the authors introduce and use the BG simulation presented earlier.

4.1.2 About progress

An important observation is that the FLP impossibility proof breaks liveness but not safety. In fact, consensus is obstruction-free solvable when enough processes are correct. Obstruction-freedom requires that a process makes progress when it runs in isolation. The work in **[314]** considers the case of anonymous processes, that is processes without identities that execute the exact same code. This paper shows that n multi-writer multi-reader (MWMR) registers of unbounded values are enough for any colorless task that is obstruction-free solvable with identifiers and any number of registers. Colorless tasks are the ones that do not involve some kind of symmetry breaking argument [298], in contrast to e.g., the renaming problem [69].

In [220], the authors classify liveness properties along two dimensions. First which methods of the shared object progress, one or all of them. Second, how benevolent the scheduler is in this task—the scheduler is sometimes called an adversary [4]. Taubenfeld [312] introduce x-obstruction-freedom which requires the system to progress when at most x processes are taking steps. The k-set agreement problem is x-obstruction-freedom solvable when $x \leq k$. A matching algorithm for $x \leq k$ that uses $(n - k + x)$ shared MWMR registers is detailed in **[314]**.

Let us note that in message-passing systems, safety can also always be maintained as long as a quorum of processes is correct [124]. This comes from the fact that a shared memory can be implemented with the help of a space-bounded simulation in these systems [71]. (As a consequence, the impossibility in the 2-processes shared-memory case of consensus implies FLP.) Conversely, if the system can be partitioned, this construction is not attainable, as shown in, e.g., the proof of Brewer's conjecture [115].

4.1.3 Solutions for agreeing

Randomized protocols can solve consensus with probability 1, where deterministic protocols fail. The seminal algorithm of Ben-Or [15] is a randomized consensus protocol for Byzantine message-passing distributed systems. The protocol tolerates up to $5f < n$ failures and has exponential expected running time—this reduces to constant time when $f \leq \sqrt{n}$. Bracha and Toueg [20] show that randomness does not help with handling more failures in the distributed case: For crash failures, at most $\lfloor \frac{n-1}{2} \rfloor$ faults are tolerated. This reduces to $\lfloor \frac{n-1}{3} \rfloor$ when processes are byzantine. Both lower bounds are tight, and Bracha and Toueg describe matching protocols for the two cases.

Aspnes and Herlihy [41] describes a randomized algorithm for shared memory, in which processes reach agreement after (in expectation) $O(n^4)$ total operations. The authors start from an exponential algorithm and refine it into a polynomial one using a (weak) shared coin. The coin is implemented with the help of an atomic counter. The Lambda of consensus [125] is a safe-agreement object that abstracts the round structure of many consensus protocols. The implementation of

Lambda using randomness in [125] is essentially the crash-tolerant algorithm in [20].

More recently, the DAG-rider distributed protocol [353] relies on randomness to implement BFT SMR against an adaptive adversary—that is, an adversary who can dynamically corrupt up to t processes. To avoid equivocation, this algorithm is built atop a reliable BFT broadcast protocol (such as Bracha's [29]). Execution is split into rounds, during which each process broadcasts a single proposal. A process may move to round $r+1$ once it has heard from $2f+1$ proposals in round r. When it moves to round $r+1$, the process chains its new proposal to the $2f+1$ proposals in round r. Once enough rounds have passed, DAG-Rider decides locally by just looking at the DAG of the execution and querying a random oracle.

As mentioned earlier in §2.1.2, another approach to sidestep FLP is to use *unreliable failure detectors* [77]. The alpha abstraction [156] extracts the safety part of Lambda and captures the safe-agreement property of round-based consensus algorithms. alpha can be built atop Σ, the quorum failure detector. This abstraction is designed to be used complementary to Ω, the weakest failure detector for consensus in shared memory. For k-set agreement, Bouzid and Travers [199] propose an extended version called alpha_k, which permits up to k values to be returned across rounds. The paper also describes an implementation for message-passing systems atop Σ_k, a generalization of Σ in which any $k+1$ quorums intersect. One can show that Σ_k is necessary to solve k-set agreement [188]. Progress in [199] is made with Ω^k. This failure detector introduced by Neiger [74] outputs a sets of k processes that eventually converge to include one non-faulty process.

In message-passing systems, $\Sigma_{n-1} \wedge \Omega^{n-1}$ is equivalent to the loneliness failure detector ($\mathcal{L}$) [170]. This detector is the weakest for set agreement, that is $(n-1)$-set agreement. **(Q)** In the message-passing system model, the weakest failure detector for $1 < k < n-1$ has so far eluded the community [149, 240].

In shared memory, the weakest failure detector for k-set agreement was found by Gafni and Kuznetsov [192]. This failure detector is called anti-omega k ($\neg\Omega_k$). Informally, $\neg\Omega_k$ outputs sets of $n-k$ processes such that some non faulty processes eventually never appear in the output; this is computationally equivalent to the Ω^k and vector-Ω^k [184] failure detectors.

Other approaches exist to enrich an asynchronous system and make agreement possible. For instance, the heard-of (HO) model of Charron-Bost and Schiper [190] principles and extends the round-by-round failure detectors [86]. In this model, a predicate defines what is received and by who at a each asynchronous round. This is particularly a good fit for agreement because of the communication-closed nature of these protocols. HO was used successfully to establish formal verification results about consensus and state-machine replication protocols [287]. In this model, a necessary and sufficient condition to solve agreement is that a neck round exists, that is a round during a majority of processes heard of each other.

Several works try to bridge the models in which agreement is solvable. For instance, there are multiple randomized algorithms (e.g., [159]) to implement leader election. Charron-Bost, Guerraoui, and Schiper [93] show that a synchronous system is strictly stronger than an asynchronous one augmented with P, the perfect failure detector. Unreliable failure detectors are appealing from the perspective of comparing problems [176, 218]. However, they also have clear limitations. One of them is that processes are still asynchronous, that is there is no upper bound on their processing speeds. As a consequence, these oracles must have liveness properties that hold asymptotically forever [200], and not during some period of time (as in, e.g., HO). There is thus a logical gap with real life since actual computing infrastructures do not satisfy this behavior. **(Q)** To date, there is no model of partial synchrony that satisfies both theory and practice on all aspects.

4.1.4 The consensus hierarchy

Another approach to sidestep FLP consists in adding synchronization primitives, such as compare-and-swap. A central result on the use of synchronization primitives is [50]. In this paper, Herlihy studies the relative computability power of synchronization objects. Computability is there defined in terms of *consensus number*. An object o has a consensus number of k when it may solve consensus among k processes, but not with $k+1$. Herlihy [50] proves that objects such as test-and-set have a consensus number of 2, whereas compare-and-swap has an infinite consensus number. The existence of an object at each level of the consensus number hierarchy is proved in [54], with the notion of k-bounded peek queue. An object quite similar, called the k-sliding object, is discovered by Mostéfaoui, Perrin, and Raynal [320]. At a given consensus level, objects are also comparable. The works in [283, 316] show that there are infinite hierarchy of deterministic objects at a given level, each weaker than the next. There are also proposals to consider other hierarchies, such as the one based on k-set agreement [305].

Kruskal, Rudolph, and Snir [24] as well as Ruppert [82] study read-modify-write objects. Objects in this class are expressed as a function which first reads the state of the object, then updates it based on the value read, before returning an appropriate response to the caller. Ruppert [82] establishes that objects of this class may solve consensus among n processes when they are n-discerning. In a nutshell, this characterization says that the object is able to discriminate two teams of processes. This permits to solve the team consensus problem where each team proposes a given value. Team consensus is stronger than weak consensus [131], a feeble form of distributed agreement, which is itself stronger than consensus. In weak consensus, processes decides on a common value in $\{0, 1\}$ so that there is an execution in which 0 is decided, and another in which 1 is decided.

The implication of the consensus hierarchy is that modern machines need to provide objects with *infinite* consensus number. Otherwise, these objects are not universal, that is, they cannot implement all objects or solve all coordination tasks in a wait-free manner for any number of processes/threads. This motivates the inclusion of strong-enough synchronization primitives in hardware. For instance, the instruction set of Intel x86 includes compare-and-swap and test-and-set [210], ARMv6 processors provide load-and-store exclusives [130], and IBM POWER8 as well as the Rock processor include a double compare-and-swap on two (non-necessarily contigüous) memory locations [395].

They are some inherent limitations to the consensus hierarchy. In [50], the consensus number of an object is defined with respect to an implementation that may use a single instance of the object. This hierarchy is not robust, in the sense that two close levels k and $k+1$ of the hierarchy never collapse. Jayanti and Toueg [54] show that, in fact, none of h_1, h_m and h_1^r are robust (where r denotes that read/wrte registers are available and m that multiple copies of an item are available). The robustness is solely possible in the h_m^r case, that is when any amount of objects might be used in conjunction with registers. In fact, robustness was established only for deterministic one-shot objects [96] and deterministic read-modify-write and readable objects [82]. For some non-deterministic constructions, the hierarchy is not robust [84, 99]. **(Q)** The general question whether the consensus hierarchy is robust for deterministic objects is open.

Summary

The line between computable and non-computable is thin in distributed computing. For agreement, it might be crossed by a slight change on either how processes share information, i.e., under which paradigm, how many of them may fail, and/or under which conditions they make progress. When agreement is not possible, proposals have been made to augment the system to permit it. There are several lines of approach, including randomness and failure detectors, each with its pros and cons. In practice, machines provide hardware support to the programmer under the form of one or more synchronization primitives. These primitives need to be strong enough to permit arbitrary forms of coordination in a parallel program. Agreement is a key reference to compare shared abstractions. To date, the terms of the comparison (e.g., bounded vs. unbounded object copies) are subject to debate.

4.2 Space and Time Complexity

4.2.1 Memory consumption

To avoid the aforementioned problems in comparing shared objects using the consensus hierarchy, Ellen et al. [336] propose to compare objects based on their ability to implement abstraction-free consensus. More precisely, this work suggests to use the minimum number of memory locations of unbounded size which are needed to solve obstruction-free consensus when using different sets of instructions.

Counting the number of memory locations to implement a higher-level abstraction is a well-established field of research. A landmark result is [59] which shows that n registers are necessary for a critical section. The proof invents the (so-called) covering argument: In a nutshell, the idea is to run the algorithm until a process is poised to write a register. When this happens, this process is parked and the run further extended by considering other processes until another register is covered. The previous argument is repeated until the desired number of registers are exhausted.

Several authors investigate the space complexity of obstruction-free consensus (or variations of it) when only registers are available. Some of these works rely on a covering argument. In particular, using this technique Fich, Herlihy, and Shavit [85] show that $\Omega(\sqrt{n})$ registers are necessary for (non-deterministic solo terminating) consensus. Despite the existence of a Paxos-like algorithm that uses exactly n registers (see, e.g., [155]), this lower bound did not improve during two decades. In a breakthrough result, Gelashvili [278] proved that any consensus algorithm has to access $\Omega(n)$ registers in some execution. This bound is tightened a year later to $n-1$ registers by Zhu [299, 360]. The gap between the upper and lower bounds for consensus is closed in [317]. The authors prove that any x-obstruction-free protocol solving k-set agreement must use $\left\lfloor \frac{n-x}{k+1-x} \right\rfloor + 1$ or more registers. Departing from prior approaches that use a covering argument, this bound is established via a reduction to the impossibility of deterministic wait-free k-set agreement. (The lower bound also applies to non-deterministic solo terminating algorithms.) In **[314]**, the authors solves x-obstruction-free k-set agreement with just $(n-k+x)$ atomic registers when processes are anonymous. **(Q)** The authors of [317] conjecture that this value is tight. The algorithm developped in **[314]** can be seen as a generalization of the leaky register [251]. Intuitively, $n-k+1$ registers allows to store that many facts, leading to a k-leaky register. For agreement, each fact corresponds to a proposal, its round, and a flag indicating whether this round is conflicting or not.

The $\Omega(\sqrt{n})$ lower bound actually applies to all history-less objects, that is objects which do not

have a memory of past operations. Ovens [368] establishes a higher $\Omega(n)$ bound on all objects of this class. In particular, he shows that $n-1$ swap objects, that is objects which permit to read the content of a register while atomically writing to it, are necessary.

4.2.2 How fast can processes agree?

Modern computers are multi-core and it is not uncommon today to have 100+ hardware threads on a server machine.[1] At the scale of a socket, cores share various levels of cache (L1-L3) and contend for the access to the (distant) main memory. The memory model of these modern architectures is not atomic. Instead hardware designs implement a weaker form of consistency such as PRAM, cache consistency, and TSO [297]. At a higher level, programming languages also offers various forms of memory consistency (e.g., [134] for Java). When a program is compiled, the consistency model of the programming language is mapped to the target machine architecture.

On modern hardware, ensuring consistency of even a single memory word is expensive. In the worth case, it requires to synchronize all the caches (across one or more sockets) which hold a copy. This cost hundreds to thousands of CPU cycles [275]. As a consequence, synchronization may lead to serious bottlenecks in programs. For that reason, programmers tend to avoid where possible synchronization. This makes memory contention in many practical case a rare event, just like asynchrony is a somewhat rare event in today's networks.

As long as the workload exhibits a good locality and fits into the first levels of the cache ($\leq$ L2), computation takes pace at the speed of the CPU clock. For that reason, one commonly measures time complexity as a function of the number of non-local steps, that is steps which access the shared memory. The number of steps required to implement a given abstraction depends on both the adversary model and the progress condition.

For agreement, several authors look to identify how fast can processes agree when communicating with just MWMR registers. The seminal algorithm of Aspnes and Herlihy [40] uses a weak shared coin and runs in polynomial expected time. A bounded variation of this algorithm is presented in [37, 56]. These algorithms are followed by the works of Saks, Shavit, and Woll [52] then Bracha and Rachman [53], with respectively $O(n^3)$ and $O(n^2 log(n))$ total step complexity. Improving upon these results, [167] proves that the total step complexity of randomized consensus is $\theta(n^2)$ in the standard shared-memory model. Besides that, Attiya and Censor [166] show that for every integer k, the probability that an f-resilient randomized consensus algorithm does not terminate the agreement within $k(n-f)$ steps is at least $\frac{1}{c^k}$, for some constant c. The lower bound also holds for asynchronous systems, where processes communicate either by message passing or through shared memory, under a very weak adversary that determines the schedule in advance, without observing the algorithm's actions.

As mentioned above, contention in practice is rare. Thus, we can sidestep a (worth-case) lower bound on time complexity by adjusting the algorithm to the environment. In detail, we mean here *adapting* the algorithm to the actual contention on the (shared) memory locations. As an example, consider the seminal Bakery algorithm of Lamport [5] (or the Boulangerie variation [319]) which solves mutual exclusion using only registers. This algorithm requires to have an upper bound on the number of processes that may access the mutual exclusion object. First, one could adapt the algorithm to adjust on the actual number of processes using it. This is possible by using a snapshot object during the read busy phase of the algorithm. In a second refinement, we may

[1] In early 2020s, common general-purpose multiprocessors have up to a few dozen cores. A core embeds one or more hardware threads, each having its own registers, that share execution resources (e.g., the system bus interface).

add a fail-safe fast path. This is the idea developped by Lamport in [31] where a process running alone may enter in $O(1)$ steps the critical section. Constructing adaptive implementations of shared objects is a prolific topic in distributed computing (see, e.g., [102, 112, 141] for adaptive algorithms implementing respectively a mutual exclusion, a collect, and a lattice agreement object). In that case, the time complexity becomes a function of the actual contention encountered by the caller of the object. In some sense, this notion is similar to the one of fast path in SMR (see Chapter 2).

The adopt-commit object is introduced by Gafni [87] in his work on round-by-round fault detectors. This object models an attempt of the processes to agree on some common value, and precisely captures the cost of the fast path a process takes during a solo run. Aspnes [197] decomposes consensus into a sequence of adopt-commit objects and conciliators, an object that produces agreement with some probability. Fast paths are further explored in [261] with the notion of conflict detector, an object that indicates when a section of code is accessed concurrently. The fast path idea of Lamport is principled under the name of *splitter* in [69]. In this paper, Moir and Anderson [69] use the splitter object to solve the renaming problem. The work of Capdevielle et al. [304] generalizes the splitter to add a value parameter. **[270]** proposes the notion of grafarius as an abstraction for weak agreement, with a fast path when a process runs solo or a decision is already taken. Attiya, Guerraoui, and Kouznetsov [128] study the complexity of solo-fast algorithms, that is algorithms which execute only read/write operations in the absence of contention.

In this context, a central question is to measure precisely the cost of the fast path. This problem is partly answered in **[361]**, where the authors present an adopt-commit object that executes the minimal number of write operations. This work considers that the number of processes (n), their identities (c), as well as the size of the input set (m) may all vary. Two algorithms are proposed. One that executes three write operations, a value optimal in the general case. This can be reduced to two when either m is known and bounded, or n identities are available. In the corner case where $n = 2$, and either $c = 2$ or the input set is finite, a single write suffices. However, this lower bound is only achievable with large amount of reads. **(Q)** An interesting open question is how to make the algorithm adaptive and consume less memory. The second algorithm in **[361]**, called Janus, is an adopt-commit implementation that executes $O(n)$ operations, including $O(\sqrt{n})$ writes. This last value is tight when the number of registers in use is bounded and m is unknown—a result proved previously by Aspnes and Ellen [261].

Summary

Many authors study the time and space complexity of agreement in the shared-memory model. The conjecture that n registers are minimal for consensus was solved recently. For k-set agreement, there is still a gap between the best lower and upper bounds. As pointed above, modern machines provide synchronization primitives that have (often) an infinite consensus number and use a single memory location. Consequently, theoretical results on the space complexity of consensus are of little interest with current hardware. Nonetheless, they permit to map and better understand the problem and its many variations. Regarding time complexity, recent works on the topic focus on improving speed in the fast path. In practice, this is the common case as programmers avoid to synchronize threads/processes too often.—because this is just too costly. When contention occurs, processes fall back to the use of a hardware synchronization primitive. Future processors will always have more cores. In this context, cache coherency might be too costly and programmers may resort to hand-written synchronization (as seen e.g., with Intel Xeon Phi). All the theoretical work on the complexity of agreement can reveal useful to find efficient solutions for these new architectures.

Chapter 5

Conclusion and Perspectives

In what follows, we provide a summary of the content of this thesis, then we present directions for future research and close.

5.1 Summary

This habilitation thesis contains three chapters. For each chapter, we detail our scientific contributions, putting them in the perspective of past and present research for the considered topic. The content of these three chapters is summarized below:

Chapter 2 details the emergence of the leaderless approach to implement state-machine replication (SMR). This chapter first provides an historical perspective on SMR, covering key theoretical and practical results over the last 50 years. The notion of leaderless state-machine replication (LSMR) is then presented. In this approach, each replica in the system can contact a quorum of its peers to contribute to the ordering of state-machine commands. Ordering is computed with the help of a local (possibly cyclic) directed graph, named the dependency graph. This graph grows monotonically over time when the replica discovers new commands and new ordering dependencies. Chapter 2 explains how to understand LSMR as a conjunction of two services: a dependency discovery service and a consensus service. Some key performance properties are listed, and the ROLL trade-off is then established. Essentially, ROLL says that there is no free lunch and that designers must find a compromise between reliability, latency, and leaderless-ness. Chapter 2 closes with a survey of different LSMR protocols, detailing the choices that their designers have made wrt. the ROLL trade-off.

When data is partially replicated at each machine, that is part of the data stored in the system is copied locally, one talks about partial replication (PR). Chapter 3 starts by explaining the origins of PR in the first packet-switching distributed systems. PR is also linked to the first database systems as well as to the atomic multicast communication primitive. Two canonical approaches exist for PR: Execute-then-Order and Order-then-Execute. SMR belongs to the first category and it generalizes into partial state-machine replication (PSMR) when replicas do not hold all the data items. Chapter 3 presents PSMR in detail and underlines the interest of this intermediate abstraction (wrt. e.g., serializability theory). Several PSMR implementations are then covered. The chapter closes with a characterization of the minimal synchrony conditions under which PSMR is solvable in the unreliable failure detectors model.

The last chapter of this thesis investigates the agreement problem in shared memory. First, an historical perspective of the problem is provided with a focus on resolubility. Then, Chapter 4 covers the spectrum of solutions using various oracles (failure detectors, synchronization primitives, and randomness). When the problem is solvable, it has a cost measured in terms of space (memory consumption) and time (non-local steps) complexity. Chapter 4 presents several related lower and upper bounds and lists open questions in this area.

5.2 Future Directions

This habilitation thesis already mentions several interesting open problems in the fields of distributed computing and distributed systems. In the text, they appear with a sign **(Q)**. Page 87 recapitulates these open questions. Below, we list directions for future research on state-machine replication, concurrency control, and data consistency.

Modern programming languages all offer support for synchronization and sharing among threads. Generally, this comes as a library of parallel constructs, e.g., for bulk synchronous parallelism [103]), synchronization primitives, and (lock-free) linearizable objects [234]. To account for their many usages, these constructs offer a large application interface. From a practical point of view, applications only use a tiny part of the interface in a piece of code. An interesting question is to evaluate the actual usage of the interface in a program and where possible provide an efficient ad-hoc implementation of the construct. To a larger extend, this questions the ability to synthesize concurrency control and protocols to manage data consistency from high-level invariants. We note here that some efforts have been already made in this direction (e.g., [262, 288]).

In a near future, two paradigms might impact the way we build applications: *serverless computing* and *edge computing*. In serverless computing, an application is split into short-lived stateless functions. In edge computing, applications execute at the border of the network, atop low performance hardware. The infrastructures supporting these two paradigms are massively parallel. As a consequence, they need means to support internally synchronization and coordination. In the same manner as NewSQL was invented to supported OLTP (see §3.1.6), storage systems need to pivot to support efficiently these new paradigms. For serverless, some recent works investigate microsecond-scale consensus (e.g., [334]). Such a line of research needs to be further expanded, in particular to improve cold start, and scalability. Another interesting direction is the use of persistent memory in future storage systems **[355]**. For edge computing, a possible approach is to mix weak and strong consistency. Depending on local synchrony and the availability of a (more powerful) intermittent cloud, the application might opt for linearizable objects and state-machine replication, or resort to weakly-consistent objects such as CRDTs [224] that synchronize in the background. The work in **[359]** is an initial step in that direction, with the motivation of reducing metadata usage in weak consistency.

Starting with the pioneering works on lazy replication [46, 78], several researchers investigate mixing weak and strong consistency. Li et al. [235] invent the decomposition of an operation into a generator that reads data and computes a response value, and an effector that applies side-effects.. Serafini et al. [209] study if eventually linearizable shared objects are constructable, and under which conditions. More recently, Unistore [349] describes a fault-tolerant approach to implement a mix of consistency. To date, these works are mainly considering the problem from a binary perspective: either an operation is strongly consistent, or not. We believe that there is instead a rainbow of possibilities here. Of course, one of the difficulties is to expose them in an understandable manner

to the programmer (e.g., with different execution paths in the program). Another related perspective is to adjust on-the-fly an SMR protocol, stretching from weakly-consistent to strongly-consistent operations. Some modules of the protocol might run or halt, depending on the network conditions and failures. We believe that the k-set agreement problem and abstractions related to it (e.g., Σ_k [188] and k-BO-broadcast [308]) can play an interesting role here.

Replication protocols are complex machineries. Complexity arises from the fact that these algorithms need to cope with both asynchrony and failures. This leads to an exponentially large number of interleavings that need all to be considered when establishing correctness. As a consequence, many protocols have flaws and mistakes—see, e.g., [300], **[347]** and **[390]** for some recently discovered problems. To prevent them, some protocol designers resort to formal verification such as model checking [191] and theorem proving [1]. In this area, modularity and intermediary abstractions (such as PSMR) can help to simplify proofs. We note here that there is still a lot to accomplish on that topic because new protocols are often proved from the ground up. A related question is the gap between specification and implementation. Of course, some simplifications in the model are helpful when writing pseudo-code. On the other hand, coding a protocol from its specification is a tedious task that encounters many new challenges—without mentioning here the performance aspect. To address these challenges, some works (e.g., [324]) annotate the code with invariants and predicates that can be verified offline. An alternative path is the use of a domain-specific language that can be verified, and also compile to actual code. This provides confidence into an abstraction while offering already some elements of implementation. Many efforts already exist on that matter. For instance, Quint [388] is a specification language close to a programming one that transpiles to TLA$^+$ [68] and Apalache [366], a symbolic model checker. A Quint program is also executable, producing a trace that can be used for unit and randomized testing. Another example is PSync [287], a partially-synchronous language for fault-tolerant distributed algorithms.

Proving that a distributed protocol always makes progress is difficult. An example is the Raft consensus protocol for which a liveness bug was found recently [384]. Difficulty arises from the fact that partial synchrony is a low-level property. In addition, there are many definitions for partial synchrony, each having its pros and cons, which spreads the effort. As mentioned in §4.1.3, failure detectors are convenient from a modularity point of view and also to compare distributed problems. Unfortunately their asymptotical properties are not implementable in real systems. Communication-closed approaches offer a possible alternative to failure detectors. Yet, not all protocols are expressible this way. We believe that there is a need for new appropriate intermediary abstractions. An example of such is the recent notion of synchronizers for BFT SMR [362].

5.3 Closing Remarks

State-machine replication (SMR) is an essential building block for modern computing infrastructures. SMR permits to store durably critical data and to construct highly-available distributed services. In this habilitation thesis, we investigate SMR from both a theoretical and a practical perspective. We explore the new class of leaderless SMR protocols, and detail approaches when data is partially replicated. SMR is closely related to the agreement problem (consensus). We survey key computability and complexity results about agreement in the shared-memory and message-passing distributed models. For all these topics, we present our contributions with respect to the state of the art, list interesting open questions, as well as future research directions.

Bibliography

[1] Martin Davis, George Logemann, and Donald Loveland. "A Machine Program for Theorem-Proving". In: *Commun. ACM* 5.7 (July 1962), pp. 394–397. ISSN: 0001-0782. DOI: 10.1145/368273.368557. URL: https://doi.org/10.1145/368273.368557.

[2] Lawrence G. Roberts and Barry D. Wessler. "Computer Network Development to Achieve Resource Sharing". In: *Proceedings of the May 5-7, 1970, Spring Joint Computer Conference*. AFIPS '70 (Spring). Atlantic City, New Jersey: Association for Computing Machinery, 1970, pp. 543–549. ISBN: 9781450379038. DOI: 10.1145/1476936.1477020. URL: https://doi.org/10.1145/1476936.1477020.

[3] John H. Wensley. "SIFT: Software Implemented Fault Tolerance". In: *Proceedings of the December 5-7, 1972, Fall Joint Computer Conference, Part I*. AFIPS '72 (Fall, part I). Anaheim, California: Association for Computing Machinery, 1972, pp. 243–253. ISBN: 9781450379120. DOI: 10.1145/1479992.1480025. URL: https://doi.org/10.1145/1479992.1480025.

[4] Edsger W. Dijkstra. "Self-stabilizing Systems in Spite of Distributed Control". In: *Commun. ACM* 17.11 (1974), pp. 643–644. DOI: 10.1145/361179.361202. URL: https://doi.org/10.1145/361179.361202.

[5] Leslie Lamport. "A New Solution of Dijkstra's Concurrent Programming Problem". In: *Commun. ACM* 17.8 (Aug. 1974), pp. 453–455. ISSN: 0001-0782. DOI: 10.1145/361082.361093. URL: https://doi.org/10.1145/361082.361093.

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

[7] K. P. Eswaran et al. "The Notions of Consistency and Predicate Locks in a Database System". In: *Commun. ACM* 19.11 (Nov. 1976), pp. 624–633. ISSN: 0001-0782. DOI: 10.1145/360363.360369. URL: https://doi.org/10.1145/360363.360369.

[8] Clarence A. Ellis. "Consistency and Correctness of Duplicate Database Systems". In: *Proceedings of the Sixth ACM Symposium on Operating Systems Principles*. SOSP '77. West Lafayette, Indiana, USA: Association for Computing Machinery, 1977, pp. 67–84. ISBN: 9781450378673. DOI: 10.1145/800214.806548. URL: https://doi.org/10.1145/800214.806548.

[9] Jim Gray. "Notes on Data Base Operating Systems". In: *Operating Systems, An Advanced Course*. Berlin, Heidelberg: Springer-Verlag, 1978, pp. 393–481. ISBN: 3540087559.

[10] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Commun. ACM* 21.7 (1978), pp. 558–565. ISSN: 0001-0782. DOI: http://doi.acm.org/10.1145/359545.359563.

[11] Hector Garcia-Molina. "Performance of update algorithms for replicated data in a distributed database". PhD thesis. Standford Computer Science Laboratory, June 1979.

[12] David K. Gifford. "Weighted Voting for Replicated Data". In: *Proceedings of the Seventh ACM Symposium on Operating Systems Principles*. SOSP '79. Pacific Grove, California, USA: Association for Computing Machinery, 1979, pp. 150–162. ISBN: 0897910095. DOI: 10.1145/800215.806583. URL: https://doi.org/10.1145/800215.806583.

[13] Danny Dolev and Ruediger Reischuk. "Bounds on Information Exchange for Byzantine Agreement". In: *Proceedings of the First ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. PODC '82. Ottawa, Canada: Association for Computing Machinery, 1982, pp. 132–140. ISBN: 0897910818. DOI: 10.1145/800220.806690. URL: https://doi.org/10.1145/800220.806690.

[14] Fred B. Schneider. "Synchronization in Distributed Programs". In: *ACM Trans. Program. Lang. Syst.* 4.2 (Apr. 1982), pp. 125–148. ISSN: 0164-0925. DOI: 10.1145/357162.357163. URL: https://doi.org/10.1145/357162.357163.

[15] Michael Ben-Or. "Another Advantage of Free Choice (Extended Abstract): Completely Asynchronous Agreement Protocols". In: *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*. PODC '83. Montreal, Quebec, Canada: Association for Computing Machinery, 1983, pp. 27–30. ISBN: 0897911105. DOI: 10.1145/800221.806707. URL: https://doi.org/10.1145/800221.806707.

[16] Anita Borg, Jim Baumbach, and Sam Glazer. "A Message System Supporting Fault Tolerance". In: *Proceedings of the Ninth ACM Symposium on Operating Systems Principles*. SOSP '83. Bretton Woods, New Hampshire, USA: Association for Computing Machinery, 1983, pp. 90–99. ISBN: 0897911156. DOI: 10.1145/800217.806617. URL: https://doi.org/10.1145/800217.806617.

[17] Jo-Mei Chang and N. F. Maxemchuk. "Reliable Broadcast Protocols". In: *ACM Trans. Comput. Syst.* 2.3 (Aug. 1984), pp. 251–273. ISSN: 0734-2071. DOI: 10.1145/989.357400. URL: https://doi.org/10.1145/989.357400.

[18] Joseph Y. Halpern and Yoram Moses. "Knowledge and Common Knowledge in a Distributed Environment". In: *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*. Ed. by Tiko Kameda et al. ACM, 1984, pp. 50–61. DOI: 10.1145/800222.806735. URL: https://doi.org/10.1145/800222.806735.

[19] Leslie Lamport. "Using Time Instead of Timeout for Fault-Tolerant Distributed Systems." In: *ACM Trans. Program. Lang. Syst.* 6.2 (Apr. 1984), pp. 254–280. ISSN: 0164-0925. DOI: 10.1145/2993.2994. URL: https://doi.org/10.1145/2993.2994.

[20] Gabriel Bracha and Sam Toueg. "Asynchronous Consensus and Broadcast Protocols". In: *J. ACM* 32.4 (1985), pp. 824–840. DOI: 10.1145/4221.214134. URL: https://doi.org/10.1145/4221.214134.

[21] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. "Impossibility of Distributed Consensus with One Faulty Process". In: *J. ACM* 32.2 (Apr. 1985), pp. 374–382. ISSN: 0004-5411. DOI: 10.1145/3149.214121. URL: http://doi.acm.org/10.1145/3149.214121.

[22] Michael J. Carey et al. "The Architecture of the EXODUS Extensible DBMS". In: *Proceedings on the 1986 International Workshop on Object-Oriented Database Systems*. OODS '86. Pacific Grove, California, USA: IEEE Computer Society Press, 1986, pp. 52–65. ISBN: 0818607343.

[23] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. *Early Stopping in Byzantine Agreement*. Tech. rep. RJ5406. Dec. 1986.

[24] Clyde P Kruskal, Larry Rudolph, and Marc Snir. "Efficient Synchronization of Multiprocessors with Shared Memory". In: *Proceedings of the Fifth Annual ACM Symposium on Principles of Distributed Computing*. PODC '86. Calgary, Alberta, Canada: Association for Computing Machinery, 1986, pp. 218–228. ISBN: 0897911989. DOI: 10.1145/10590.10609. URL: https://doi.org/10.1145/10590.10609.

[25] C. Mohan, B. Lindsay, and R. Obermarck. "Transaction Management in the R* Distributed Database Management System". In: *ACM Trans. Database Syst.* 11.4 (Dec. 1986), pp. 378–396. ISSN: 0362-5915. DOI: 10.1145/7239.7266. URL: https://doi.org/10.1145/7239.7266.

[26] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. http://research.microsoft.com/pubs/ccontrol/. Addison-Wesley, 1987.

[27] Kenneth P. Birman and Thomas A. Joseph. "Exploiting virtual synchrony in distributed systems". In: ACM. Austin TX, USA, Nov. 1987, pp. 123–138.

[28] Kenneth P. Birman and Thomas A. Joseph. "Reliable Communication in the Presence of Failures". In: *ACM Transactions on Computers Systems* 5.1 (Jan. 1987), pp. 47–76. ISSN: 0734-2071. DOI: 10.1145/7351.7478. URL: http://doi.acm.org/10.1145/7351.7478.

[29] Gabriel Bracha. "Asynchronous Byzantine agreement protocols". In: *Information and Computation* 75.2 (1987), pp. 130–143. ISSN: 0890-5401. DOI: https://doi.org/10.1016/0890-5401(87)90054-X. URL: https://www.sciencedirect.com/science/article/pii/089054018790054X.

[30] Hector Garcia-Molina and Kenneth Salem. "Sagas". In: *Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data*. SIGMOD '87. San Francisco, California, USA: Association for Computing Machinery, 1987, pp. 249–259. ISBN: 0897912365. DOI: 10.1145/38713.38742. URL: https://doi.org/10.1145/38713.38742.

[31] Leslie Lamport. "A Fast Mutual Exclusion Algorithm". In: *ACM Trans. Comput. Syst.* 5.1 (Jan. 1987), pp. 1–11. ISSN: 0734-2071. DOI: 10.1145/7351.7352. URL: https://doi.org/10.1145/7351.7352.

[32] Michael C Loui and Hosame H Abu-Amara. "Memory requirements for agreement among unreliable asynchronous processes". In: *Advances in Computing research* 4 (31 1987), pp. 163–183.

[33] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony". In: *J. ACM* 35.2 (1988), pp. 288–323. ISSN: 0004-5411.

[34] Brian M. Oki and Barbara H. Liskov. "Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems". In: *PODC '88: Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*. Toronto, Ontario, Canada: ACM, 1988, pp. 8–17. ISBN: 0-89791-277-2. DOI: http://doi.acm.org/10.1145/62546.62549.

[35] W. E. Weihl. "Commutativity-Based Concurrency Control for Abstract Data Types". In: *Proceedings of the Twenty-First Annual Hawaii International Conference on Software Track*. Kailua-Kona, Hawaii, USA: IEEE Computer Society Press, 1988, pp. 205–214. ISBN: 0818608420.

[36] A. El Abbadi and S. Toueg. "Maintaining Availability in Partitioned Replicated Databases". In: *ACM Trans. Database Syst.* 14.2 (June 1989), pp. 264–290. ISSN: 0362-5915. DOI: 10.1145/63500.63501. URL: https://doi.org/10.1145/63500.63501.

[37] H. Attiya, D. Dolev, and N. Shavit. "Bounded Polynomial Randomized Consensus". In: *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*. PODC '89. Edmonton, Alberta, Canada: Association for Computing Machinery, 1989, pp. 281–293. ISBN: 0897913264. DOI: 10.1145/72981.73001. URL: https://doi.org/10.1145/72981.73001.

[38] Kerry Raymond. "A Tree-Based Algorithm for Distributed Mutual Exclusion". In: *ACM Trans. Comput. Syst.* 7.1 (Jan. 1989), pp. 61–77. ISSN: 0734-2071. DOI: 10.1145/58564.59295. URL: https://doi.org/10.1145/58564.59295.

[39] J. H. Wensley et al. "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control". In: *Tutorial: Hard Real-Time Systems*. Washington, DC, USA: IEEE Computer Society Press, 1989, pp. 560–575. ISBN: 0818608196.

[40] James Aspnes and M. Herlihy. "Fast Randomized Consensus Using Shared Memory". In: *J. Algorithms* 11.3 (Sept. 1990), pp. 441–461. ISSN: 0196-6774. DOI: 10.1016/0196-6774(90)90021-6. URL: https://doi.org/10.1016/0196-6774(90)90021-6.

[41] James Aspnes and Maurice Herlihy. "Fast Randomized Consensus Using Shared Memory". In: *J. Algorithms* 11.3 (1990), pp. 441–461. DOI: 10.1016/0196-6774(90)90021-6. URL: https://doi.org/10.1016/0196-6774(90)90021-6.

[42] Haran Boral et al. "Prototyping Bubba, A Highly Parallel Database System". In: *IEEE Trans. Knowl. Data Eng.* 2.1 (1990), pp. 4–24. DOI: 10.1109/69.50903. URL: https://doi.org/10.1109/69.50903.

[43] Soma Chaudhuri. "Agreement is Harder Than Consensus: Set Consensus Problems in Totally Asynchronous Systems". In: *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*. PODC '90. Quebec City, Quebec, Canada: ACM, 1990, pp. 311–324. ISBN: 0-89791-404-X. DOI: 10.1145/93385.93431. URL: http://doi.acm.org/10.1145/93385.93431.

[44] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. "Early Stopping in Byzantine Agreement". In: *J. ACM* 37.4 (1990), pp. 720–741. DOI: 10.1145/96559.96565. URL: https://doi.org/10.1145/96559.96565.

[45] Maurice Herlihy and Jeannette Wing. "Linearizability: a Correcteness Condition for Concurrent Objects". In: *ACM Transactions on Programming Languages and Systems* 12.3 (July 1990), pp. 463–492.

[46] Rivka Ladin, Barbara Liskov, and Liuba Shrira. "Lazy Replication: Exploiting the Semantics of Distributed Services". In: *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*. PODC '90. Quebec City, Quebec, Canada: Association for Computing Machinery, 1990, pp. 43–57. ISBN: 089791404X. DOI: 10.1145/93385.93399. URL: https://doi.org/10.1145/93385.93399.

[47] Fred B. Schneider. "Implementing fault-tolerant services using the state machine approach: a tutorial". In: *ACM Comput. Surv.* 22.4 (1990), pp. 299–319. ISSN: 0360-0300.

[48] Ofer Biran, Shlomo Moran, and Shmuel Zaks. "Deciding 1-Solvability of Distributed Task is NP-Hard (Extended Abstract)". In: *Proceedings of the 16th International Workshop on Graph-Theoretic Concepts in Computer Science*. WG '90. Berlin, Germany: Springer-Verlag, 1991, pp. 206–220. ISBN: 0387538321.

[49] Kenneth Birman, André Schiper, and Pat Stephenson. "Lightweight Causal and Atomic Group Multicast". In: *ACM Trans. Comput. Syst.* 9.3 (Aug. 1991), pp. 272–314. ISSN: 0734-2071. DOI: 10.1145/128738.128742. URL: https://doi.org/10.1145/128738.128742.

[50] Maurice Herlihy. "Wait-Free Synchronization". In: *ACM Trans. Program. Lang. Syst.* 13.1 (Jan. 1991), pp. 124–149. ISSN: 0164-0925. DOI: 10.1145/114005.102808. URL: https://doi.org/10.1145/114005.102808.

[51] Charles Lamb et al. "The ObjectStore Database System". In: *Commun. ACM* 34.10 (Oct. 1991), pp. 50–63. ISSN: 0001-0782. DOI: 10.1145/125223.125244. URL: https://doi.org/10.1145/125223.125244.

[52] Michael Saks, Nir Shavit, and Heather Woll. "Optimal Time Randomized Consensus—Making Resilient Algorithms Fast in Practice". In: *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*. SODA '91. San Francisco, California, USA: Society for Industrial and Applied Mathematics, 1991, pp. 351–362. ISBN: 0897913760.

[53] Gabi Bracha and Ophir Rachman. "Randomized consensus in expected O(n2log n) operations". In: *Distributed Algorithms*. Ed. by Sam Toueg, Paul G. Spirakis, and Lefteris Kirousis. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 143–150. ISBN: 978-3-540-46789-2.

[54] Prasad Jayanti and Sam Toueg. "Some results on the impossibility, universality, and decidability of consensus". In: *Distributed Algorithms*. Ed. by Adrian Segall and Shmuel Zaks. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 69–84. ISBN: 978-3-540-47484-5.

[55] Rivka Ladin, Barbara Liskov, and Sanjay Ghemawat. "Providing High Availability Using Lazy Replication". In: 10.4 (Nov. 1992), pp. 360–391.

[56] James Aspnes. "Time- and space-efficient randomized consensus". In: *Journal of Algorithms* 14.3 (May 1993), pp. 414–431.

[57] Elizabeth Borowsky and Eli Gafni. "Generalized FLP Impossibility Result for T-Resilient Asynchronous Computations". In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '93. San Diego, California, USA: Association for Computing Machinery, 1993, pp. 91–100. ISBN: 0897915917. DOI: 10.1145/167088.167119. URL: https://doi.org/10.1145/167088.167119.

[58] Elizabeth Borowsky and Eli Gafni. "Generalized FLP impossibility result for t-resilient asynchronous computations". In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*. Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. ACM, 1993, pp. 91–100. DOI: 10.1145/167088.167119. URL: https://doi.org/10.1145/167088.167119.

[59] James E. Burns and Nancy A. Lynch. "Bounds on Shared Memory for Mutual Exclusion". In: *Inf. Comput.* 107.2 (1993), pp. 171–184. DOI: 10.1006/inco.1993.1065. URL: https://doi.org/10.1006/inco.1993.1065.

[60] Danny Dolev, Shlomo Kramer, and Dalia Malki. "Early Delivery Totally Ordered Multicast in Asynchronous Environments". In: *Proceedings of the 23nd Annual International Symposium on Fault*. 1993, pp. 544–553.

[61] Maurice Herlihy and Nir Shavit. "The asynchronous computability theorem for t-resilient tasks". In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*. Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. ACM, 1993, pp. 111–120. DOI: 10.1145/167088.167125. URL: https://doi.org/10.1145/167088.167125.

[62] Prasad Jayanti. "On the Robustness of Herlihy's Hierarchy". In: *Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing*. PODC '93. Ithaca, New York, USA: Association for Computing Machinery, 1993, pp. 145–157. ISBN: 0897916131. DOI: 10.1145/164051.164070. URL: https://doi.org/10.1145/164051.164070.

[63] Michael E. Saks and Fotios Zaharoglou. "Wait-free k-set agreement is impossible: the topology of public knowledge". In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*. Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. ACM, 1993, pp. 101–110. DOI: 10.1145/167088.167122. URL: https://doi.org/10.1145/167088.167122.

[64] Hagit Attiya and Jennifer L. Welch. "Sequential consistency versus linearizability". In: *ACM Trans. Comput. Syst.* 12.2 (May 1994), pp. 91–122. ISSN: 0734-2071.

[65] Michael J. Carey et al. "Shoring up Persistent Applications". In: *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*. SIGMOD '94. Minneapolis, Minnesota, USA: Association for Computing Machinery, 1994, pp. 383–394. ISBN: 0897916395. DOI: 10.1145/191839.191915. URL: https://doi.org/10.1145/191839.191915.

[66] Vassos Hadzilacos and Sam Toueg. *A Modular Approach to Fault-Tolerant Broadcasts and Related Problems*. Tech. rep. Cornell University, 1994.

[67] Amos Israeli and Lihu Rappoport. "Disjoint-Access-Parallel Implementations of Strong Shared Memory Primitives". In: *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '94. Los Angeles, California, USA: Association for Computing Machinery, 1994, pp. 151–160. ISBN: 0897916549. DOI: 10.1145/197917.198079. URL: https://doi.org/10.1145/197917.198079.

[68] Leslie Lamport. "The Temporal Logic of Actions". In: *ACM Trans. Program. Lang. Syst.* 16.3 (May 1994), pp. 872–923. ISSN: 0164-0925. DOI: 10.1145/177492.177726. URL: https://doi.org/10.1145/177492.177726.

[69] Mark Moir and James H. Anderson. "Fast, Long-Lived Renaming (Extended Abstract)". In: *Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 - October 1, 1994, Proceedings*. 1994, pp. 141–155. DOI: 10.1007/BFb0020430. URL: http://dx.doi.org/10.1007/BFb0020430.

[70] Douglas B. Terry et al. "Session Guarantees for Weakly Consistent Replicated Data". In: *Proceedings of the Third International Conference on on Parallel and Distributed Information Systems*. PDIS '94. Autin, Texas, USA: IEEE Computer Society Press, 1994, pp. 140–150. ISBN: 0818664010.

[71] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. "Sharing Memory Robustly in Message-Passing Systems". In: *J. ACM* 42.1 (Jan. 1995), pp. 124–142. ISSN: 0004-5411. DOI: 10.1145/200836.200869. URL: https://doi.org/10.1145/200836.200869.

[72] Z. Galil, A. Mayer, and Moti Yung. "Resolving message complexity of Byzantine Agreement and beyond". In: *Proceedings of IEEE 36th Annual Foundations of Computer Science*. 1995, pp. 724–733. DOI: 10.1109/SFCS.1995.492674.

[73] Rachid Guerraoui. "Revisiting the relationship between non-blocking atomic commitment and consensus". In: *Distributed Algorithms*. Ed. by Jean-Michel Hélary and Michel Raynal. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 87–100. ISBN: 978-3-540-44783-2.

[74] Gil Neiger. "Failure Detectors and the Wait-Free Hierarchy (Extended Abstract)". In: *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '95. Ottowa, Ontario, Canada: Association for Computing Machinery, 1995, pp. 100–109. ISBN: 0897917103. DOI: 10.1145/224964.224976. URL: https://doi.org/10.1145/224964.224976.

[75] Dennis Shasha et al. "Transaction Chopping: Algorithms and Performance Studies". In: *ACM Trans. Database Syst.* 20.3 (Sept. 1995), pp. 325–363. ISSN: 0362-5915. DOI: 10.1145/211414.211427. URL: https://doi.org/10.1145/211414.211427.

[76] Tushar Deepak Chandra, Vassos Hadzilacos, and Sam Toueg. "The Weakest Failure Detector for Solving Consensus". In: *J. ACM* 43.4 (July 1996), pp. 685–722. ISSN: 0004-5411. DOI: 10.1145/234533.234549. URL: http://doi.acm.org/10.1145/234533.234549.

[77] Tushar Deepak Chandra and Sam Toueg. "Unreliable Failure Detectors for Reliable Distributed Systems". In: *J. ACM* 43.2 (1996), pp. 225–267. DOI: 10.1145/226643.226647. URL: https://doi.org/10.1145/226643.226647.

[78] Alan Fekete et al. "Eventually-Serializable Data Services". In: *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 300–309. ISBN: 0897918002. DOI: 10.1145/248052.248113. URL: https://doi.org/10.1145/248052.248113.

[79] Jim Gray et al. "The dangers of replication and a solution". In: *SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data*. Montreal, Quebec, Canada: ACM Press, 1996, pp. 173–182. ISBN: 0-89791-794-4.

[80] M. Frans Kaashoek and Andrew S. Tanenbaum. "An Evaluation of the Amoeba Group Communication System". In: *Proceedings of the 16th International Conference on Distributed Computing Systems, Hong Kong, May 27-30, 1996*. IEEE Computer Society, 1996, pp. 436–448. DOI: 10.1109/ICDCS.1996.507992. URL: https://doi.org/10.1109/ICDCS.1996.507992.

[81] David Karger et al. "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web". In: *STOC*. 1997.

[82] Eric Ruppert. "Determining Consensus Numbers". In: *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '97. Santa Barbara, California, USA: Association for Computing Machinery, 1997, pp. 93–99. ISBN: 0897919521. DOI: 10.1145/259380.259427. URL: https://doi.org/10.1145/259380.259427.

[83] André Schiper. "Early Consensus in an Asynchronous System with a Weak Failure Detector". In: *Distributed Comput.* 10.3 (1997), pp. 149–157. DOI: 10.1007/s004460050032. URL: https://doi.org/10.1007/s004460050032.

[84] Eric Shenk. "The Consensus Hierarchy is Not Robust". In: *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '97. Santa Barbara, California, USA: Association for Computing Machinery, 1997, p. 279. ISBN: 0897919521. DOI: 10.1145/259380.259479. URL: https://doi.org/10.1145/259380.259479.

[85] Faith E. Fich, Maurice Herlihy, and Nir Shavit. "On the Space Complexity of Randomized Synchronization". In: *J. ACM* 45.5 (1998), pp. 843–862. DOI: 10.1145/290179.290183. URL: https://doi.org/10.1145/290179.290183.

[86] Eli Gafni. "Round-by-Round Fault Detectors (Extended Abstract): Unifying Synchrony and Asynchrony". In: *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '98. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998, pp. 143–152. ISBN: 0897919777. DOI: 10.1145/277697.277724. URL: https://doi.org/10.1145/277697.277724.

[87] Eli Gafni. "Round-by-round Fault Detectors (Extended Abstract): Unifying Synchrony and Asynchrony". In: *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*. PODC '98. Puerto Vallarta, Mexico: ACM, 1998, pp. 143–152. ISBN: 0-89791-977-7. DOI: 10.1145/277697.277724. URL: http://doi.acm.org/10.1145/277697.277724.

[88] Leslie Lamport. "The Part-Time Parliament". In: *ACM Trans. Comput. Syst.* (1998).

[89] Atul Adya. "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions". Ph.D. Cambridge, MA, USA: MIT, Mar. 1999.

[90] Maurice Herlihy and Sergio Rajsbaum. "New Perspectives in Distributed Computing". In: *Proceedings of the 24th International Symposium on Mathematical Foundations of Computer Science*. MFCS '99. Berlin, Heidelberg: Springer-Verlag, 1999, pp. 170–186. ISBN: 3540664084.

[91] Maurice Herlihy and Nir Shavit. "The topological structure of asynchronous computability". In: *J. ACM* 46.6 (1999), pp. 858–923. DOI: 10.1145/331524.331529. URL: https://doi.org/10.1145/331524.331529.

[92] Marcos Kawazoe Aguilera et al. "Thrifty Generic Broadcast". In: *Distributed Computing, 14th International Conference, DISC 2000, Toledo, Spain, October 4-6, 2000, Proceedings*. Ed. by Maurice Herlihy. Vol. 1914. Lecture Notes in Computer Science. Springer, 2000, pp. 268–282. DOI: 10.1007/3-540-40026-5_18. URL: https://doi.org/10.1007/3-540-40026-5%5C_18.

[93] Bernadette Charron-Bost, Rachid Guerraoui, and Andre Schiper. "Synchronous System and Perfect Failure Detector: Solvability and Efficiency Issue". In: *Proceedings of the 2000 International Conference on Dependable Systems and Networks (Formerly FTCS-30 and DCCA-8)*. DSN '00. USA: IEEE Computer Society, 2000, pp. 523–532. ISBN: 0769507077.

[94] Carole Delporte-Gallet and Hugues Fauconnier. "Fault-Tolerant Genuine Atomic Multicast to Multiple Groups". In: *Procedings of the 4th International Conference on Principles of Distributed Systems, OPODIS 2000, Paris, France, December 20-22, 2000*. Ed. by Franck Butelle. Studia Informatica Universalis. Suger, rue Catulienne Saint-Denis France, 2000, pp. 107–122.

[95] Rachid Guerraoui. "Indulgent algorithms (preliminary version)". In: *PODC '00*. Portland, Oregon, United States: ACM, 2000, pp. 289–297. ISBN: 1-58113-183-6.

[96] M. Herlihy and E. Ruppert. "On the existence of booster types". In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. 2000, pp. 653–663. DOI: 10.1109/SFCS.2000.892333.

[97] Bettina Kemme and Gustavo Alonso. "A new approach to developing and implementing eager database replication protocols". In: *ACM Transactions on Database Systems* 25.3 (2000), pp. 333–379. URL: citeseer.ist.psu.edu/kemme00new.html.

[98] Bettina Kemme and Gustavo Alonso. "Don't Be Lazy, Be Consistent: Postgres-R, A New Way to Implement Database Replication". In: *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*. Ed. by Amr El Abbadi et al. Morgan Kaufmann, 2000, pp. 134–143. URL: http://www.vldb.org/conf/2000/P134.pdf.

[99] Wai-Kau Lo and Vassos Hadzilacos. "All of Us Are Smarter than Any of Us: Nondeterministic Wait-Free Hierarchies Are Not Robust". In: *SIAM Journal on Computing* 30.3 (2000), pp. 689–728. DOI: 10.1137/S0097539798335766.

[100] M. Wiesmann et al. "Understanding replication in databases and distributed systems". In: *Proceedings of 20th International Conference on Distributed Computing Systems (ICDCS'2000)*. Taipei, Taiwan, R.O.C.: IEEE Computer Society Technical Commitee on Distributed Processing, 2000, pp. 264–274. URL: citeseer.ist.psu.edu/wiesmann00understanding.html.

[101] Matthias Wiesmann et al. "Database Replication Techniques: A Three Parameter Classification". In: *19th IEEE Symposium on Reliable Distributed Systems, SRDS'00, Nürnberg, Germany, October 16-18, 2000, Proceedings*. IEEE Computer Society, 2000, pp. 206–215. DOI: 10.1109/RELDI.2000.885408. URL: https://doi.org/10.1109/RELDI.2000.885408.

[102] Hagit Attiya and Arie Fouren. "Adaptive and Efficient Algorithms for Lattice Agreement and Renaming". In: *SIAM J. Comput.* 31.2 (2001), pp. 642–664. DOI: 10.1137/S0097539700366000. URL: https://doi.org/10.1137/S0097539700366000.

[103] Yan Gu, Bu-Sung Lee, and Wentong Cai. "JBSP: A BSP Programming Library in Java". In: *Journal of Parallel and Distributed Computing* 61.8 (2001), pp. 1126–1142. ISSN: 0743-7315. DOI: https://doi.org/10.1006/jpdc.2001.1735. URL: https://www.sciencedirect.com/science/article/pii/S0743731501917356.

[104] Rachid Guerraoui and André Schiper. "Genuine atomic multicast in asynchronous distributed systems". In: *Theor. Comput. Sci.* 254.1-2 (2001), pp. 297–316. DOI: 10.1016/S0304-3975(99)00161-9. URL: https://doi.org/10.1016/S0304-3975(99)00161-9.

[105] Joseph Y. Halpern, Yoram Moses, and Orli Waarts. "A Characterization of Eventual Byzantine Agreement". In: *SIAM J. Comput.* 31.3 (2001), pp. 838–865. DOI: 10.1137/S0097539798340217. URL: https://doi.org/10.1137/S0097539798340217.

[106] Udo Fritzke Jr. et al. "Consensus-Based Fault-Tolerant Total Order Multicast". In: *IEEE Trans. Parallel Distributed Syst.* 12.2 (2001), pp. 147–156. DOI: 10.1109/71.910870. URL: https://doi.org/10.1109/71.910870.

[107] Leslie Lamport. "Paxos Made Simple". In: *Sigact News - SIGACT* 32 (Jan. 2001).

[108] Butler Lampson. "The ABCD's of Paxos". In: *Proceedings of the Twentieth Annual ACM Symposium on Principles of Distributed Computing*. PODC '01. Newport, Rhode Island, USA: Association for Computing Machinery, 2001, p. 13. ISBN: 1581133839. DOI: 10.1145/383962.383969. URL: https://doi.org/10.1145/383962.383969.

[109] António Luís Pinto Ferreira de Sousa et al. "Partial Replication in the Database State Machine". In: *IEEE International Symposium on Network Computing and Applications (NCA 2001), October 8-10, 2001, Cambridge, MA, USA*. IEEE Computer Society, 2001, pp. 298–309. DOI: 10.1109/NCA.2001.962546. URL: https://doi.org/10.1109/NCA.2001.962546.

[110] Ion Stoica et al. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In: *SIGCOMM Comput. Commun. Rev.* 31.4 (Aug. 2001), pp. 149–160. ISSN: 0146-4833. DOI: 10.1145/964723.383071. URL: https://doi.org/10.1145/964723.383071.

[111] Ion Stoica et al. "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications". In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. SIGCOMM '01. San Diego, California, USA: Association for Computing Machinery, 2001, pp. 149–160. ISBN: 1581134118. DOI: 10.1145/383059.383071. URL: https://doi.org/10.1145/383059.383071.

[112] Hagit Attiya and Vita Bortnikov. "Adaptive and efficient mutual exclusion". In: *Distributed Comput.* 15.3 (2002), pp. 177–189. DOI: 10.1007/s004460100068. URL: https://doi.org/10.1007/s004460100068.

[113] Carole Delporte-Gallet, Hugues Fauconnier, and Rachid Guerraoui. "A Realistic Look At Failure Detectors". In: *2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings*. IEEE Computer Society, 2002, pp. 345–353. DOI: 10.1109/DSN.2002.1028919. URL: https://doi.org/10.1109/DSN.2002.1028919.

[114] Partha Dutta and Rachid Guerraoui. "Fast Indulgent Consensus with Zero Degradation". In: *Proceedings of the 4th European Dependable Computing Conference on Dependable Computing*. EDCC-4. Berlin, Heidelberg: Springer-Verlag, 2002, pp. 191–208. ISBN: 3540000127.

[115] Seth Gilbert and Nancy Lynch. "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services". In: *SIGACT News* 33.2 (2002), pp. 51–59.

[116] Ricardo Jiménez-Peris et al. "Improving the Scalability of Fault-Tolerant Database Clusters". In: *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2-5, 2002*. IEEE Computer Society, 2002, pp. 477–484. DOI: 10.1109/ICDCS.2002.1022297. URL: https://doi.org/10.1109/ICDCS.2002.1022297.

[117] F. Pedone and A. Schiper. "Handling message semantics with Generic Broadcast protocols". In: *Distributed Computing* 15 (2002), pp. 97–107.

[118] Bernadette Charron-Bost. "Comparing the Atomic Commitment and Consensus Problems". In: *Future Directions in Distributed Computing: Research and Position Papers*. Ed. by André Schiper et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 29–34. ISBN: 978-3-540-37795-5. DOI: 10.1007/3-540-37795-6_6. URL: https://doi.org/10.1007/3-540-37795-6_6.

[119] J. Holliday et al. "Epidemic algorithms for replicated databases". In: *IEEE Transactions on Knowledge and Data Engineering* 15.5 (2003), pp. 1218–1238. DOI: 10.1109/TKDE.2003.1232274.

[120] Fernando Pedone, Rachid Guerraoui, and André Schiper. "The Database State Machine Approach". In: *Distributed Parallel Databases* 14.1 (2003), pp. 71–98. DOI: 10.1023/A:1022887812188. URL: https://doi.org/10.1023/A:1022887812188.

[121] Emmanuel Cecchet, Julie Marguerite, and Willy Zwaenepole. "C-JDBC: Flexible Database Clustering Middleware". In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference*. ATEC '04. Boston, MA: USENIX Association, 2004, p. 26.

[122] Xavier Défago, André Schiper, and Péter Urbán. "Total Order Broadcast and Multicast Algorithms: Taxonomy and Survey". In: *ACM Comput. Surv.* 36.4 (Dec. 2004), pp. 372–421. ISSN: 0360-0300. DOI: 10.1145/1041680.1041682. URL: https://doi.org/10.1145/1041680.1041682.

[123] Xavier Défago, André Schiper, and Péter Urbán. "Total order broadcast and multicast algorithms: Taxonomy and survey". In: *ACM Comput. Surv.* 36.4 (2004), pp. 372–421. DOI: 10.1145/1041680.1041682. URL: https://doi.org/10.1145/1041680.1041682.

[124] Carole Delporte-Gallet et al. "The weakest failure detectors to solve certain fundamental problems in distributed computing". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing, PODC 2004, St. John's, Newfoundland, Canada, July 25-28, 2004*. 2004, pp. 338–346. DOI: 10.1145/1011767.1011818. URL: https://doi.org/10.1145/1011767.1011818.

[125] Rachid Guerraoui and Michel Raynal. "The Information Structure of Indulgent Consensus". In: *IEEE Trans. Computers* 53.4 (2004), pp. 453–466. DOI: 10.1109/TC.2004.1268403. URL: https://doi.org/10.1109/TC.2004.1268403.

[126] Robbert van Renesse and Fred B. Schneider. "Chain Replication for Supporting High Throughput and Availability". In: *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*. OSDI'04. San Francisco, CA: USENIX Association, 2004, p. 7.

[127] Amitanand S. Aiyer et al. “BAR fault tolerance for cooperative services”. In: *Proceedings of the 20th ACM Symposium on Operating Systems Principles 2005, SOSP 2005, Brighton, UK, October 23-26, 2005*. Ed. by Andrew Herbert and Kenneth P. Birman. ACM, 2005, pp. 45–58. DOI: 10.1145/1095810.1095816. URL: https://doi.org/10.1145/1095810.1095816.

[128] Hagit Attiya, Rachid Guerraoui, and Petr Kouznetsov. “Computing with Reads and Writes in the Absence of Step Contention”. In: *Distributed Computing: 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005. Proceedings*. Ed. by Pierre Fraigniaud. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 122–136. ISBN: 978-3-540-32075-3. DOI: 10.1007/11561927_11. URL: http://dx.doi.org/10.1007/11561927_11.

[129] Partha Dutta, Rachid Guerraoui, and Leslie Lamport. “How Fast Can Eventual Synchrony Lead to Consensus?” In: *2005 International Conference on Dependable Systems and Networks (DSN 2005), 28 June - 1 July 2005, Yokohama, Japan, Proceedings*. IEEE Computer Society, 2005, pp. 22–27. DOI: 10.1109/DSN.2005.54. URL: https://doi.org/10.1109/DSN.2005.54.

[130] John Goodacre and Andrew N. Sloss. “Parallelism and the ARM Instruction Set Architecture”. In: *Computer* 38.7 (July 2005), pp. 42–50. ISSN: 0018-9162. DOI: 10.1109/MC.2005.239. URL: https://doi.org/10.1109/MC.2005.239.

[131] Petr Kuznetsov. “Synchronization using failure detectors”. PhD thesis. EPFL, Jan. 2005. DOI: 10.5075/epfl-thesis-3262.

[132] Leslie Lamport. *Generalized Consensus and Paxos*. Tech. rep. MSR-TR-2005-33. Microsoft, Mar. 2005.

[133] Jeremy Manson, William Pugh, and Sarita V. Adve. “The Java Memory Model”. In: *SIGPLAN Not.* 40.1 (Jan. 2005), pp. 378–391. ISSN: 0362-1340. DOI: 10.1145/1047659.1040336. URL: https://doi.org/10.1145/1047659.1040336.

[134] Jeremy Manson, William Pugh, and Sarita V. Adve. “The Java Memory Model”. In: *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’05. Long Beach, California, USA: Association for Computing Machinery, 2005, pp. 378–391. ISBN: 158113830X. DOI: 10.1145/1040305.1040336. URL: https://doi.org/10.1145/1040305.1040336.

[135] Esther Pacitti et al. “Preventive Replication in a Database Cluster”. In: *Distributed Parallel Databases* 18.3 (2005), pp. 223–251. DOI: 10.1007/s10619-005-4257-4. URL: https://doi.org/10.1007/s10619-005-4257-4.

[136] Marta Patiño-Martínez et al. “MIDDLE-R: Consistent database replication at the middleware level”. In: *ACM Trans. Comput. Syst.* 23.4 (2005), pp. 375–423. DOI: 10.1145/1113574.1113576. URL: https://doi.org/10.1145/1113574.1113576.

[137] Matthias Wiesmann and André Schiper. “Comparison of Database Replication Techniques Based on Total Order Broadcast”. In: *IEEE Transactions on Knowledge and Data Engineering* 17.4 (2005), pp. 551–566. ISSN: 1041-4347.

[138] Shuqing Wu and Bettina Kemme. “Postgres-R(SI): Combining Replica Control with Concurrency Control Based on Snapshot Isolation”. In: *Proceedings of the 21st International Conference on Data Engineering*. ICDE ’05. USA: IEEE Computer Society, 2005, pp. 422–433. ISBN: 0769522858. DOI: 10.1109/ICDE.2005.108. URL: https://doi.org/10.1109/ICDE.2005.108.

[139] Piotr Zielinski. "Optimistic Generic Broadcast". In: *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*. Ed. by Pierre Fraigniaud. Vol. 3724. Lecture Notes in Computer Science. Springer, 2005, pp. 369–383. DOI: 10.1007/11561927_27. URL: https://doi.org/10.1007/11561927%5C_27.

[140] T. Anker et al. "Wire-speed total order". In: *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*. 2006. DOI: 10.1109/IPDPS.2006.1639381.

[141] Hagit Attiya et al. "Efficient adaptive collect using randomization". In: *Distributed Comput.* 18.3 (2006), pp. 179–188. DOI: 10.1007/s00446-005-0143-6. URL: https://doi.org/10.1007/s00446-005-0143-6.

[142] Mike Burrows. "The Chubby lock service for loosely-coupled distributed systems". In: *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*. Seattle, Washington: USENIX Association, 2006, pp. 335–350. ISBN: 1-931971-47-1.

[143] Fay Chang et al. "Bigtable: A Distributed Storage System for Structured Data". In: *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*. OSDI '06. Seattle, WA: USENIX Association, 2006, p. 15.

[144] Dan Dobre and Neeraj Suri. "One-step Consensus with Zero-Degradation". In: *2006 International Conference on Dependable Systems and Networks (DSN 2006), 25-28 June 2006, Philadelphia, Pennsylvania, USA, Proceedings*. IEEE Computer Society, 2006, pp. 137–146. DOI: 10.1109/DSN.2006.55. URL: https://doi.org/10.1109/DSN.2006.55.

[145] Jim Gray and Leslie Lamport. "Consensus on transaction commit". In: *ACM Trans. Database Syst.* 31.1 (2006), pp. 133–160. DOI: 10.1145/1132863.1132867. URL: https://doi.org/10.1145/1132863.1132867.

[146] Leslie Lamport. "Lower bounds for asynchronous consensus". In: *Distributed Computing* 19.2 (Oct. 2006), pp. 104–125. ISSN: 0178-2770.

[147] Jacob R. Lorch et al. "The SMART Way to Migrate Replicated Stateful Services". In: *Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006*. EuroSys '06. Leuven, Belgium: Association for Computing Machinery, 2006, pp. 103–115. ISBN: 1595933220. DOI: 10.1145/1217935.1217946. URL: https://doi.org/10.1145/1217935.1217946.

[148] Jacob R. Lorch et al. "The SMART Way to Migrate Replicated Stateful Services". In: *SIGOPS Oper. Syst. Rev.* 40.4 (Apr. 2006), pp. 103–115. ISSN: 0163-5980. DOI: 10.1145/1218063.1217946. URL: https://doi.org/10.1145/1218063.1217946.

[149] Michel Raynal and Corentin Travers. "In Search of the Holy Grail: Looking for the Weakest Failure Detector for Wait-Free Set Agreement". In: *Principles of Distributed Systems, 10th International Conference, OPODIS 2006, Bordeaux, France, December 12-15, 2006, Proceedings*. Ed. by Alexander A. Shvartsman. Vol. 4305. Lecture Notes in Computer Science. Springer, 2006, pp. 3–19. DOI: 10.1007/11945529_2. URL: https://doi.org/10.1007/11945529%5C_2.

[151] B. Ban. *JGroups: A Toolkit for Reliable Multicast Communication*. 2007. URL: http://www.jgroups.org.

[152] Lásaro J. Camargos, Rodrigo Schmidt, and Fernando Pedone. "Multicoordinated Paxos". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007*. Ed. by Indranil Gupta and Roger Wattenhofer. ACM, 2007, pp. 316–317. DOI: 10.1145/1281100.1281150. URL: https://doi.org/10.1145/1281100.1281150.

[153] Tushar D. Chandra, Robert Griesemer, and Joshua Redstone. "Paxos Made Live: An Engineering Perspective". In: *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing*. PODC '07. Portland, Oregon, USA: Association for Computing Machinery, 2007, pp. 398–407. ISBN: 9781595936165. DOI: 10.1145/1281100.1281103. URL: https://doi.org/10.1145/1281100.1281103.

[154] Giuseppe DeCandia et al. "Dynamo: amazon's highly available key-value store". In: *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*. Ed. by Thomas C. Bressoud and M. Frans Kaashoek. ACM, 2007, pp. 205–220. DOI: 10.1145/1294261.1294281. URL: https://doi.org/10.1145/1294261.1294281.

[155] Rachid Guerraoui and Michel Raynal. "The Alpha of Indulgent Consensus". In: *Comput. J.* 50.1 (2007), pp. 53–67. DOI: 10.1093/comjnl/bxl046. URL: https://doi.org/10.1093/comjnl/bxl046.

[156] Rachid Guerraoui and Michel Raynal. "The Alpha of Indulgent Consensus". In: *Comput. J.* 50.1 (2007), pp. 53–67. DOI: 10.1093/comjnl/bxl046. URL: https://doi.org/10.1093/comjnl/bxl046.

[157] Rachid Guerraoui et al. "On the Weakest Failure Detector Ever". In: *Proceedings of the Twenty-sixth Annual ACM Symposium on Principles of Distributed Computing*. PODC '07. Portland, Oregon, USA: ACM, 2007, pp. 235–243. ISBN: 978-1-59593-616-5. DOI: 10.1145/1281100.1281135. URL: http://doi.acm.org/10.1145/1281100.1281135.

[159] Murali Krishna Ramanathan et al. "Randomized leader election". In: *Distributed Computing* 19 (2007), pp. 403–418.

[160] Rodrigo Schmidt and Fernando Pedone. "A Formal Analysis of the Deferred Update Technique". In: *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings*. Ed. by Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal. Vol. 4878. Lecture Notes in Computer Science. Springer, 2007, pp. 16–30. DOI: 10.1007/978-3-540-77096-1_2. URL: https://doi.org/10.1007/978-3-540-77096-1%5C_2.

[161] Damián Serrano et al. "Boosting Database Replication Scalability through Partial Replication and 1-Copy-Snapshot-Isolation". In: *13th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC 2007), 17-19 December, 2007, Melbourne, Victoria, Australia*. IEEE Computer Society, 2007, pp. 290–297. DOI: 10.1109/PRDC.2007.39. URL: https://doi.org/10.1109/PRDC.2007.39.

[164] Amazon. *AWS Simple Storage Service*. https://aws.amazon.com/s3. 2008.

[165] J. E. Armendáriz-Iñigo et al. "SIPRe: A Partial Database Replication Protocol with SI Replicas". In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. SAC '08. Fortaleza, Ceara, Brazil: Association for Computing Machinery, 2008, pp. 2181–2185. ISBN: 9781595937537. DOI: 10.1145/1363686.1364207. URL: https://doi.org/10.1145/1363686.1364207.

[166] Hagit Attiya and Keren Censor. "Lower Bounds for Randomized Consensus under a Weak Adversary". In: *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*. PODC '08. Toronto, Canada: Association for Computing Machinery, 2008, pp. 315–324. ISBN: 9781595939890. DOI: 10.1145/1400751.1400793. URL: https://doi.org/10.1145/1400751.1400793.

[167] Hagit Attiya and Keren Censor. "Tight Bounds for Asynchronous Randomized Consensus". In: *J. ACM* 55.5 (Nov. 2008). ISSN: 0004-5411. DOI: 10.1145/1411509.1411510. URL: https://doi.org/10.1145/1411509.1411510.

[169] Brian F. Cooper et al. "PNUTS: Yahoo!'s Hosted Data Serving Platform". In: *Proc. VLDB Endow.* 1.2 (Aug. 2008), pp. 1277–1288. ISSN: 2150-8097. DOI: 10.14778/1454159.1454167. URL: https://doi.org/10.14778/1454159.1454167.

[170] Carole Delporte-Gallet et al. "The Weakest Failure Detector for Message Passing Set-Agreement". In: *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*. Ed. by Gadi Taubenfeld. Vol. 5218. Lecture Notes in Computer Science. Springer, 2008, pp. 109–120. DOI: 10.1007/978-3-540-87779-0_8. URL: https://doi.org/10.1007/978-3-540-87779-0%5C_8.

[171] Pascal Felber, Christof Fetzer, and Torvald Riegel. "Dynamic Performance Tuning of Word-Based Software Transactional Memory". In: *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '08. Salt Lake City, UT, USA: Association for Computing Machinery, 2008, pp. 237–246. ISBN: 9781595937957. DOI: 10.1145/1345206.1345241. URL: https://doi.org/10.1145/1345206.1345241.

[172] Rachid Guerraoui, Thomas A. Henzinger, and Vasu Singh. "Permissiveness in Transactional Memories". In: *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*. Ed. by Gadi Taubenfeld. Vol. 5218. Lecture Notes in Computer Science. Springer, 2008, pp. 305–319. DOI: 10.1007/978-3-540-87779-0_21. URL: https://doi.org/10.1007/978-3-540-87779-0%5C_21.

[173] Rachid Guerraoui and Michal Kapalka. "On Obstruction-Free Transactions". In: *Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures*. SPAA '08. Munich, Germany: Association for Computing Machinery, 2008, pp. 304–313. ISBN: 9781595939739. DOI: 10.1145/1378533.1378587. URL: https://doi.org/10.1145/1378533.1378587.

[174] Rachid Guerraoui and Michal Kapalka. "On the Correctness of Transactional Memory". In: *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*. PPoPP '08. Salt Lake City, UT, USA: ACM, 2008, pp. 175–184. ISBN: 978-1-59593-795-7. DOI: 10.1145/1345206.1345233. URL: http://doi.acm.org/10.1145/1345206.1345233.

[175] Stavros Harizopoulos et al. “OLTP through the looking glass, and what we found there”. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. Ed. by Jason Tsong-Li Wang. ACM, 2008, pp. 981–992. DOI: 10.1145/1376616.1376713. URL: https://doi.org/10.1145/1376616.1376713.

[176] Prasad Jayanti and Sam Toueg. “Every problem has a weakest failure detector”. In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*. Ed. by Rida A. Bazzi and Boaz Patt-Shamir. ACM, 2008, pp. 75–84. DOI: 10.1145/1400751.1400763. URL: https://doi.org/10.1145/1400751.1400763.

[177] Robert Kallman et al. “H-store: a high-performance, distributed main memory transaction processing system”. In: *Proc. VLDB Endow.* 1.2 (2008), pp. 1496–1499. DOI: 10.14778/1454159.1454211. URL: http://www.vldb.org/pvldb/vol1/1454211.pdf.

[178] Yanhua Mao, Flavio Paiva Junqueira, and Keith Marzullo. “Mencius: Building Efficient Replicated State Machine for WANs”. In: *Symposium on Operating Systems Design and Implementation (OSDI)*. 2008.

[179] Yoram Moses and Michel Raynal. “No Double Discount: Condition-Based Simultaneity Yields Limited Gain”. In: *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings*. Ed. by Gadi Taubenfeld. Vol. 5218. Lecture Notes in Computer Science. Springer, 2008, pp. 423–437. DOI: 10.1007/978-3-540-87779-0_29. URL: https://doi.org/10.1007/978-3-540-87779-0%5C_29.

[180] Nicolas Schiper and Fernando Pedone. “Solving Atomic Multicast When Groups Crash”. In: *Principles of Distributed Systems, 12th International Conference, OPODIS 2008, Luxor, Egypt, December 15-18, 2008. Proceedings*. Ed. by Theodore P. Baker, Alain Bui, and Sébastien Tixeuil. Vol. 5401. Lecture Notes in Computer Science. Springer, 2008, pp. 481–495. DOI: 10.1007/978-3-540-92221-6_30. URL: https://doi.org/10.1007/978-3-540-92221-6_30.

[181] Damián Serrano et al. “An Autonomic Approach for Replication of Internet-based Services”. In: *27th IEEE Symposium on Reliable Distributed Systems (SRDS 2008), Napoli, Italy, October 6-8, 2008*. IEEE Computer Society, 2008, pp. 127–136. DOI: 10.1109/SRDS.2008.22. URL: https://doi.org/10.1109/SRDS.2008.22.

[184] Piotr Zielinski. “Anti-Ω: The Weakest Failure Detector for Set Agreement”. In: *Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*. PODC ’08. Toronto, Canada: Association for Computing Machinery, 2008, pp. 55–64. ISBN: 9781595939890. DOI: 10.1145/1400751.1400761. URL: https://doi.org/10.1145/1400751.1400761.

[185] Marcos Kawazoe Aguilera et al. “Sinfonia: A new paradigm for building scalable distributed systems”. In: *ACM Trans. Comput. Syst.* 27.3 (2009), 5:1–5:48. DOI: 10.1145/1629087.1629088. URL: https://doi.org/10.1145/1629087.1629088.

[186] Hagit Attiya, Eshcar Hillel, and Alessia Milani. “Inherent Limitations on Disjoint-Access Parallel Implementations of Transactional Memory”. In: *Proceedings of the Twenty-First Annual Symposium on Parallelism in Algorithms and Architectures*. SPAA ’09. Calgary, AB, Canada: Association for Computing Machinery, 2009, pp. 69–78. ISBN: 9781605586069. DOI: 10.1145/1583991.1584015. URL: https://doi.org/10.1145/1583991.1584015.

[187] François Bonnet and Michel Raynal. "Looking for the Weakest Failure Detector for k-Set Agreement in Message-Passing Systems: Is {\$it\$Pi}_k{\it \Pi}_k the End of the Road?" In: *Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings*. Ed. by Rachid Guerraoui and Franck Petit. Vol. 5873. Lecture Notes in Computer Science. Springer, 2009, pp. 149–164. DOI: 10.1007/978-3-642-05118-0_11. URL: https://doi.org/10.1007/978-3-642-05118-0%5C_11.

[188] François Bonnet and Michel Raynal. "Looking for the Weakest Failure Detector for k-Set Agreement in Message-Passing Systems: Is {\$it\$Pi}_k{\it \Pi}_k the End of the Road?" In: *Stabilization, Safety, and Security of Distributed Systems, 11th International Symposium, SSS 2009, Lyon, France, November 3-6, 2009. Proceedings*. Ed. by Rachid Guerraoui and Franck Petit. Vol. 5873. Lecture Notes in Computer Science. Springer, 2009, pp. 149–164. DOI: 10.1007/978-3-642-05118-0_11. URL: https://doi.org/10.1007/978-3-642-05118-0%5C_11.

[189] Bernadette Charron-Bost and André Schiper. "The Heard-Of model: computing in distributed systems with benign faults". In: *Distributed Computing* 22.1 (2009), pp. 49–71.

[190] Bernadette Charron-Bost and André Schiper. "The Heard-Of model: computing in distributed systems with benign faults". In: *Distributed Comput.* 22.1 (2009), pp. 49–71. DOI: 10.1007/s00446-009-0084-6. URL: https://doi.org/10.1007/s00446-009-0084-6.

[191] Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis. "Model Checking: Algorithmic Verification and Debugging". In: *Commun. ACM* 52.11 (Nov. 2009), pp. 74–84. ISSN: 0001-0782. DOI: 10.1145/1592761.1592781. URL: https://doi.org/10.1145/1592761.1592781.

[192] Eli Gafni and Petr Kuznetsov. "The weakest failure detector for solving k-set agreement". In: *Proceedings of the 28th Annual ACM Symposium on Principles of Distributed Computing, PODC 2009, Calgary, Alberta, Canada, August 10-12, 2009*. Ed. by Srikanta Tirthapura and Lorenzo Alvisi. ACM, 2009, pp. 83–91. DOI: 10.1145/1582716.1582735. URL: https://doi.org/10.1145/1582716.1582735.

[193] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. "Vertical Paxos and Primary-Backup Replication". In: *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*. PODC '09. Calgary, AB, Canada: Association for Computing Machinery, 2009, pp. 312–313. ISBN: 9781605583969. DOI: 10.1145/1582716.1582783. URL: https://doi.org/10.1145/1582716.1582783.

[194] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. "Vertical Paxos and Primary-backup Replication". In: *Proceedings of the 28th ACM Symposium on Principles of Distributed Computing*. PODC '09. Calgary, AB, Canada: ACM, 2009, pp. 312–313. ISBN: 978-1-60558-396-9. DOI: 10.1145/1582716.1582783. URL: http://doi.acm.org/10.1145/1582716.1582783.

[196] Giuliana Santos Veronese et al. "Spin One's Wheels? Byzantine Fault Tolerance with a Spinning Primary". In: *28th IEEE Symposium on Reliable Distributed Systems (SRDS 2009), Niagara Falls, New York, USA, September 27-30, 2009*. IEEE Computer Society, 2009, pp. 135–144. DOI: 10.1109/SRDS.2009.36. URL: https://doi.org/10.1109/SRDS.2009.36.

[197] James Aspnes. "A Modular Approach to Shared-memory Consensus, with Applications to the Probabilistic-write Model". In: *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. PODC '10. Zurich, Switzerland: ACM, 2010, pp. 460–467. ISBN: 978-1-60558-888-9. DOI: 10.1145/1835698.1835802. URL: http://doi.acm.org/10.1145/1835698.1835802.

[198] François Bonnet and Michel Raynal. "A simple proof of the necessity of the failure detector Sigma to implement an atomic register in asynchronous message-passing systems". In: *Inf. Process. Lett.* 110.4 (2010), pp. 153–157. DOI: 10.1016/j.ipl.2009.11.011. URL: https://doi.org/10.1016/j.ipl.2009.11.011.

[199] Zohir Bouzid and Corentin Travers. "(anti-Omega^x $\times$Sigma$_z$)-Based k-Set Agreement Algorithms". In: *Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings*. Ed. by Chenyang Lu, Toshimitsu Masuzawa, and Mohamed Mosbah. Vol. 6490. Lecture Notes in Computer Science. Springer, 2010, pp. 189–204. DOI: 10.1007/978-3-642-17653-1_16. URL: https://doi.org/10.1007/978-3-642-17653-1%5C_16.

[200] Bernadette Charron-Bost, Martin Hutle, and Josef Widder. "In search of lost time". In: *Inf. Process. Lett.* 110.21 (2010), pp. 928–933. DOI: 10.1016/j.ipl.2010.07.017. URL: https://doi.org/10.1016/j.ipl.2010.07.017.

[201] Kristina Chodorow and Michael Dirolf. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly, 2010, pp. I–XVII, 1–193. ISBN: 978-1-449-38156-1.

[202] Brian F. Cooper et al. "Benchmarking Cloud Serving Systems with YCSB". In: *Symposium on Cloud Computing (SoCC)*. 2010.

[203] Patrick Hunt et al. "ZooKeeper: Wait-free Coordination for Internet-scale Systems". In: *USENIX Annual Technical Conference (USENIX ATC)*. 2010.

[204] Evan P.C. Jones, Daniel J. Abadi, and Samuel Madden. "Low Overhead Concurrency Control for Partitioned Main Memory Databases". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. SIGMOD '10. Indianapolis, Indiana, USA: Association for Computing Machinery, 2010, pp. 603–614. ISBN: 9781450300322. DOI: 10.1145/1807167.1807233. URL: https://doi.org/10.1145/1807167.1807233.

[205] Avinash Lakshman and Prashant Malik. "Cassandra: a decentralized structured storage system". In: *ACM SIGOPS Oper. Syst. Rev.* 44.2 (2010), pp. 35–40. DOI: 10.1145/1773912.1773922. URL: https://doi.org/10.1145/1773912.1773922.

[206] Parisa J. Marandi et al. "Ring Paxos: A High-Throughput Atomic Broadcast Protocol". In: (2010).

[207] Robbert van Renesse and Rachid Guerraoui. "Replication Techniques for Availability". In: *Replication: Theory and Practice*. Ed. by Bernadette Charron-Bost, Fernando Pedone, and André Schiper. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 19–40. ISBN: 978-3-642-11294-2. DOI: 10.1007/978-3-642-11294-2_2. URL: https://doi.org/10.1007/978-3-642-11294-2_2.

[209] Marco Serafini et al. "Eventually Linearizable Shared Objects". In: *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. PODC '10. Zurich, Switzerland: Association for Computing Machinery, 2010, pp. 95–104. ISBN: 9781605588889. DOI: 10.1145/1835698.1835723. URL: https://doi.org/10.1145/1835698.1835723.

[210] Tom Shanley. *X86 Instruction Set Architecture*. Mindshare Press, 2010. ISBN: 0977087859.

[211] Konstantin V. Shvachko. "HDFS Scalability: The Limits to Growth". In: *USENIX login* 35.2 (Apr. 2010).

[212] Alexander Thomson and Daniel J. Abadi. "The Case for Determinism in Database Systems". In: *Proc. VLDB Endow.* 3.1 (2010), pp. 70–80. DOI: 10.14778/1920841.1920855. URL: http://www.vldb.org/pvldb/vldb2010/pvldb%5C_vol3/R06.pdf.

[213] Marcos Kawazoe Aguilera et al. "Dynamic atomic storage without consensus". In: *J. ACM* 58.2 (2011), 7:1–7:32. DOI: 10.1145/1944345.1944348. URL: https://doi.org/10.1145/1944345.1944348.

[214] Jason Baker et al. "Megastore: Providing Scalable, Highly Available Storage for Interactive Services". In: *Proceedings of the Conference on Innovative Data system Research (CIDR)*. 2011, pp. 223–234. URL: http://www.cidrdb.org/cidr2011/Papers/CIDR11_Paper32.pdf.

[216] Felix C. Freiling, Rachid Guerraoui, and Petr Kuznetsov. "The Failure Detector Abstraction". In: *ACM Comput. Surv.* 43.2 (Feb. 2011). ISSN: 0360-0300. DOI: 10.1145/1883612.1883616. URL: https://doi.org/10.1145/1883612.1883616.

[217] Felix C. Freiling, Rachid Guerraoui, and Petr Kuznetsov. "The Failure Detector Abstraction". In: *ACM Comput. Surv.* 43.2 (Feb. 2011). ISSN: 0360-0300. DOI: 10.1145/1883612.1883616. URL: https://doi.org/10.1145/1883612.1883616.

[218] Felix C. Freiling, Rachid Guerraoui, and Petr Kuznetsov. "The failure detector abstraction". In: *ACM Comput. Surv.* 43.2 (2011), 9:1–9:40. DOI: 10.1145/1883612.1883616. URL: https://doi.org/10.1145/1883612.1883616.

[219] Lisa Glendenning et al. "Scalable Consistency in Scatter". In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. SOSP '11. Cascais, Portugal: Association for Computing Machinery, 2011, pp. 15–28. ISBN: 9781450309776. DOI: 10.1145/2043556.2043559. URL: https://doi.org/10.1145/2043556.2043559.

[220] Maurice Herlihy and Nir Shavit. "On the Nature of Progress". In: *Principles of Distributed Systems - 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings*. Ed. by Antonio Fernández Anta, Giuseppe Lipari, and Matthieu Roy. Vol. 7109. Lecture Notes in Computer Science. Springer, 2011, pp. 313–328. DOI: 10.1007/978-3-642-25873-2_22. URL: https://doi.org/10.1007/978-3-642-25873-2%5C_22.

[221] Flavio Paiva Junqueira, Benjamin C. Reed, and Marco Serafini. "Zab: High-performance broadcast for primary-backup systems". In: *Proceedings of the 2011 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2011, Hong Kong, China, June 27-30 2011*. IEEE Compute Society, 2011, pp. 245–256. DOI: 10.1109/DSN.2011.5958223. URL: https://doi.org/10.1109/DSN.2011.5958223.

[222] P.J. Marandi, M. Primi, and F. Pedone. "High performance state-machine replication". In: *IEEE/IFIP 41st International Conference on Dependable Systems and Networks*. DSN. 2011.

[224] Marc Shapiro et al. “Conflict-Free Replicated Data Types”. In: *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings*. Ed. by Xavier Défago, Franck Petit, and Vincent Villain. Vol. 6976. Lecture Notes in Computer Science. Springer, 2011, pp. 386–400. DOI: 10.1007/978-3-642-24550-3_29. URL: https://doi.org/10.1007/978-3-642-24550-3%5C_29.

[225] Yair Sovran et al. “Transactional storage for geo-replicated systems”. In: Cascais, Portugal, Oct. 2011, pp. 385–400. DOI: http://doi.acm.org/10.1145/2043556.2043592.

[227] Amazon. *AWS Dynamodb*. https://aws.amazon.com/dynamodb. 2012.

[229] James C. Corbett et al. “Spanner: Google’s Globally-Distributed Database”. In: *Symposium on Operating Systems Design and Implementation (OSDI)*. 2012.

[230] James A. Cowling and Barbara Liskov. “Granola: Low-Overhead Distributed Transaction Coordination”. In: *2012 USENIX Annual Technical Conference, Boston, MA, USA, June 13-15, 2012*. Ed. by Gernot Heiser and Wilson C. Hsieh. USENIX Association, 2012, pp. 223–235. URL: https://www.usenix.org/conference/atc12/technical-sessions/presentation/cowling.

[231] Faith Ellen et al. “Universal Constructions That Ensure Disjoint-Access Parallelism and Wait-Freedom”. In: *Proceedings of the 2012 ACM Symposium on Principles of Distributed Computing*. PODC ’12. Madeira, Portugal: Association for Computing Machinery, 2012, pp. 115–124. ISBN: 9781450314503. DOI: 10.1145/2332432.2332457. URL: https://doi.org/10.1145/2332432.2332457.

[232] Rachid Guerraoui et al. “The Weakest Failure Detectors to Solve Quittable Consensus and Nonblocking Atomic Commit”. In: *SIAM J. Comput.* 41.6 (2012), pp. 1343–1379. DOI: 10.1137/070698877. URL: https://doi.org/10.1137/070698877.

[233] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. Morgan Kaufmann Publishers Inc., 2012.

[234] Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming, Revised Reprint*. 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2012. ISBN: 9780123973375.

[235] Cheng Li et al. “Making Geo-Replicated Systems Fast as Possible, Consistent When Necessary”. In: *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation*. OSDI’12. Hollywood, CA, USA: USENIX Association, 2012, pp. 265–278. ISBN: 9781931971966.

[236] Dahlia Malkhi et al. “From paxos to CORFU: a flash-speed shared log”. In: *ACM SIGOPS Oper. Syst. Rev.* 46.1 (2012), pp. 47–51. DOI: 10.1145/2146382.2146391. URL: https://doi.org/10.1145/2146382.2146391.

[237] Parisa Jalili Marandi, Marco Primi, and Fernando Pedone. “Multi-Ring Paxos”. In: *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012*. Ed. by Robert S. Swarz, Philip Koopman, and Michel Cukier. IEEE Computer Society, 2012, pp. 1–12. DOI: 10.1109/DSN.2012.6263916. URL: https://doi.org/10.1109/DSN.2012.6263916.

[238] Parisa Jalili Marandi, Marco Primi, and Fernando Pedone. "Multi-Ring Paxos". In: *IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2012, Boston, MA, USA, June 25-28, 2012*. Ed. by Robert S. Swarz, Philip Koopman, and Michel Cukier. IEEE Computer Society, 2012, pp. 1–12. DOI: 10.1109/DSN.2012.6263916. URL: https://doi.org/10.1109/DSN.2012.6263916.

[239] Francesco Marchioni and Manik Surtani. *Infinispan Data Grid Platform*. Packt Publishing Ltd, 2012.

[240] Achour Mostéfaoui, Michel Raynal, and Julien Stainer. "Chasing the Weakest Failure Detector for k-Set Agreement in Message-Passing Systems". In: *11th IEEE International Symposium on Network Computing and Applications, NCA 2012, Cambridge, MA, USA, August 23-25, 2012*. IEEE Computer Society, 2012, pp. 44–51. DOI: 10.1109/NCA.2012.19. URL: https://doi.org/10.1109/NCA.2012.19.

[241] Sebastiano Peluso et al. "When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication". In: *Int. Conf. on Distributed Computing Systems*. Macau, China: IEEE, 2012, pp. 455–465.

[242] *ScyllaDB*. https://www.scylladb.com. 2012.

[243] Alexander Shraer et al. "Dynamic Reconfiguration of Primary/Backup Clusters". In: *USENIX Annual Technical Conference (USENIX ATC)*. 2012.

[244] The Apache Software Foundation. *The Hadoop Distributed File System*. 2012. URL: http://hadoop.apache.org.

[245] Alexander Thomson et al. "Calvin: fast distributed transactions for partitioned database systems". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*. Ed. by K. Selçuk Candan et al. ACM, 2012, pp. 1–12. DOI: 10.1145/2213836.2213838. URL: https://doi.org/10.1145/2213836.2213838.

[249] Mahesh Balakrishnan et al. "CORFU: A distributed shared log". In: *ACM Trans. Comput. Syst.* 31.4 (2013), p. 10. DOI: 10.1145/2535930. URL: https://doi.org/10.1145/2535930.

[250] Mahesh Balakrishnan et al. "Tango: distributed data structures over a shared log". In: *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*. Ed. by Michael Kaminsky and Mike Dahlin. ACM, 2013, pp. 325–340. DOI: 10.1145/2517349.2522732. URL: https://doi.org/10.1145/2517349.2522732.

[251] Carole Delporte-Gallet et al. "Adaptive Register Allocation with a Linear Number of Registers". In: *Proceedings of the 27th International Symposium on Distributed Computing - Volume 8205*. DISC 2013. Jerusalem, Israel: Springer-Verlag New York, Inc., 2013, pp. 269–283. ISBN: 978-3-642-41526-5. DOI: 10.1007/978-3-642-41527-2_19. URL: http://dx.doi.org/10.1007/978-3-642-41527-2_19.

[252] Danny Dolev and Christoph Lenzen. "Early-Deciding Consensus is Expensive". In: *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*. PODC '13. Montréal, Québec, Canada: Association for Computing Machinery, 2013, pp. 270–279. ISBN: 9781450320658. DOI: 10.1145/2484239.2484269. URL: https://doi.org/10.1145/2484239.2484269.

[253] Vassos Hadzilacos and Sam Toueg. "On Deterministic Abortable Objects". In: *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*. PODC '13. Montréal, Québec, Canada: Association for Computing Machinery, 2013, pp. 4–12. ISBN: 9781450320658. DOI: 10.1145/2484239.2484241. URL: https://doi.org/10.1145/2484239.2484241.

[254] Tim Kraska et al. "MDCC: Multi-Data Center Consistency". In: *European Conference on Computer Systems (EuroSys)*. 2013.

[255] Christopher Mitchell, Yifeng Geng, and Jinyang Li. "Using One-Sided RDMA Reads to Build a Fast, CPU-Efficient Key-Value Store". In: *Proceedings of the 2013 USENIX Conference on Annual Technical Conference*. USENIX ATC'13. San Jose, CA: USENIX Association, 2013, pp. 103–114.

[256] Iulian Moraru, David G. Andersen, and Michael Kaminsky. "There Is More Consensus in Egalitarian Parliaments". In: *Symposium on Operating Systems Principles (SOSP)*. 2013.

[257] Oracle. *Java(TM) Transaction API (JTA)*. Tech. rep. 2013.

[261] James Aspnes and Faith Ellen. "Tight Bounds for Adopt-Commit Objects". English. In: *Theory of Computing Systems* 55.3 (2014), pp. 451–474. ISSN: 1432-4350. DOI: 10.1007/s00224-013-9448-1. URL: http://dx.doi.org/10.1007/s00224-013-9448-1.

[262] Peter Bailis et al. "Coordination Avoidance in Database Systems". In: *Proc. VLDB Endow.* 8.3 (2014), pp. 185–196. DOI: 10.14778/2735508.2735509. URL: http://www.vldb.org/pvldb/vol8/p185-bailis.pdf.

[263] Carlos Eduardo Benevides Bezerra, Fernando Pedone, and Robbert van Renesse. "Scalable State-Machine Replication". In: *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*. IEEE Computer Society, 2014, pp. 331–342. DOI: 10.1109/DSN.2014.41. URL: https://doi.org/10.1109/DSN.2014.41.

[264] Jiaqing Du et al. "Clock-RSM: Low-Latency Inter-datacenter State Machine Replication Using Loosely Synchronized Physical Clocks". In: *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*. 2014, pp. 343–354. DOI: 10.1109/DSN.2014.42. URL: https://doi.org/10.1109/DSN.2014.42.

[267] Shuai Mu et al. "Extracting More Concurrency from Distributed Transactions". In: *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*. OSDI'14. Broomfield, CO: USENIX Association, 2014, pp. 479–494. ISBN: 9781931971164.

[268] Arun C. Murthy et al. *Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2*. 1st. Addison-Wesley Professional, 2014. ISBN: 0321934504.

[269] Diego Ongaro and John K. Ousterhout. "In Search of an Understandable Consensus Algorithm". In: *USENIX Annual Technical Conference (USENIX ATC)*. 2014.

[271] Alexandru Turcu et al. "Be General and Don't Give Up Consistency in Geo-Replicated Transactional Systems". In: *International Conference on Principles of Distributed Systems (OPODIS)*. 2014, pp. 33–48.

[274] Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. "A Framework for Transactional Consistency Models with Atomic Visibility". In: *26th International Conference on Concurrency Theory (CONCUR 2015)*. Ed. by Luca Aceto and David de Frutos Escrig. Vol. 42. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, pp. 58–71. ISBN: 978-3-939897-91-0. DOI: 10.4230/LIPIcs.CONCUR.2015.58. URL: http://drops.dagstuhl.de/opus/volltexte/2015/5375.

[275] Austin T. Clements et al. "The Scalable Commutativity Rule: Designing Scalable Software for Multicore Processors". In: *ACM Trans. Comput. Syst.* 32.4 (Jan. 2015). ISSN: 0734-2071. DOI: 10.1145/2699681. URL: https://doi.org/10.1145/2699681.

[276] Huynh Tu Dang et al. "NetPaxos: Consensus at Network Speed". In: *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*. SOSR '15. Santa Clara, California: Association for Computing Machinery, 2015. ISBN: 9781450334518. DOI: 10.1145/2774993.2774999. URL: https://doi.org/10.1145/2774993.2774999.

[277] Swan Dubois et al. "The Weakest Failure Detector for Eventual Consistency". In: *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*. PODC '15. Donostia-San Sebastián, Spain: Association for Computing Machinery, 2015, pp. 375–384. ISBN: 9781450336178. DOI: 10.1145/2767386.2767404. URL: https://doi.org/10.1145/2767386.2767404.

[278] Rati Gelashvili. "On the Optimal Space Complexity of Consensus for Anonymous Processes". In: *Distributed Computing - 29th International Symposium, DISC 2015, Tokyo, Japan, October 7-9, 2015, Proceedings*. Ed. by Yoram Moses. Vol. 9363. Lecture Notes in Computer Science. Springer, 2015, pp. 452–466. DOI: 10.1007/978-3-662-48653-5_30. URL: https://doi.org/10.1007/978-3-662-48653-5%5C_30.

[280] Robbert van Renesse, Nicolas Schiper, and Fred B. Schneider. "Vive La Différence: Paxos vs. Viewstamped Replication vs. Zab". In: *IEEE Trans. Dependable Secur. Comput.* 12.4 (2015), pp. 472–484. DOI: 10.1109/TDSC.2014.2355848. URL: https://doi.org/10.1109/TDSC.2014.2355848.

[282] Irene Zhang et al. "Building Consistent Transactions with Inconsistent Replication". In: *Symposium on Operating Systems Principles (SOSP)*. 2015.

[283] Yehuda Afek, Faith Ellen, and Eli Gafni. "Deterministic Objects: Life Beyond Consensus". In: *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*. PODC '16. Chicago, Illinois, USA: Association for Computing Machinery, 2016, pp. 97–106. ISBN: 9781450339643. DOI: 10.1145/2933057.2933116. URL: https://doi.org/10.1145/2933057.2933116.

[285] Brendan Burns et al. "Borg, Omega, and Kubernetes". In: *Queue* 14.1 (Jan. 2016), 10:70–10:93. ISSN: 1542-7730.

[286] Huynh Tu Dang et al. "Paxos Made Switch-y". In: *SIGCOMM Comput. Commun. Rev.* 46.2 (May 2016), pp. 18–24. ISSN: 0146-4833. DOI: 10.1145/2935634.2935638. URL: https://doi.org/10.1145/2935634.2935638.

[287] Cezara Dragoi, Thomas A. Henzinger, and Damien Zufferey. "PSync: a partially synchronous language for fault-tolerant distributed algorithms". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by Rastislav Bodík and Rupak Majumdar. ACM, 2016, pp. 400–415. DOI: 10.1145/2837614.2837650. URL: https://doi.org/10.1145/2837614.2837650.

[288] Alexey Gotsman et al. "'Cause I'm strong enough: reasoning about consistency choices in distributed systems". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by Rastislav Bodík and Rupak Majumdar. ACM, 2016, pp. 371–384. DOI: 10.1145/2837614.2837625. URL: https://doi.org/10.1145/2837614.2837625.

[289] Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. "Flexible Paxos: Quorum Intersection Revisited". In: *International Conference on Principles of Distributed Systems (OPODIS)*. 2016.

[290] Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman. "Flexible Paxos: Quorum intersection revisited". In: *CoRR* abs/1608.06696 (2016). arXiv: 1608.06696. URL: http://arxiv.org/abs/1608.06696.

[291] Zsolt István et al. "Consensus in a Box: Inexpensive Coordination in Hardware". In: *Proceedings of the 13th Usenix Conference on Networked Systems Design and Implementation*. NSDI'16. Santa Clara, CA: USENIX Association, 2016, pp. 425–438. ISBN: 9781931971294.

[292] Kfir Lev-Ari et al. "Modular Composition of Coordination Services". In: *2016 USENIX Annual Technical Conference, USENIX ATC 2016, Denver, CO, USA, June 22-24, 2016*. Ed. by Ajay Gulati and Hakim Weatherspoon. USENIX Association, 2016, pp. 251–264. URL: https://www.usenix.org/conference/atc16/technical-sessions/presentation/lev-ari.

[293] Jialin Li et al. "Just Say No to Paxos Overhead: Replacing Consensus with Network Ordering". In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 467–483. ISBN: 9781931971331.

[294] Shengyun Liu et al. "XFT: Practical Fault Tolerance beyond Crashes". In: *Symposium on Operating Systems Design and Implementation (OSDI)*. 2016.

[295] Haonan Lu et al. "The SNOW Theorem and Latency-Optimal Read-Only Transactions". In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. Ed. by Kimberly Keeton and Timothy Roscoe. USENIX Association, 2016, pp. 135–150. URL: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/lu.

[296] Shuai Mu et al. "Consolidating Concurrency Control and Consensus for Commits under Conflicts". In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. Ed. by Kimberly Keeton and Timothy Roscoe. USENIX Association, 2016, pp. 517–532.

[297] Paolo Viotti and Marko Vukolić. "Consistency in Non-Transactional Distributed Storage Systems". In: *ACM Comput. Surv.* 49.1 (June 2016). ISSN: 0360-0300. DOI: 10.1145/2926965. URL: https://doi.org/10.1145/2926965.

[298] Nayuta Yanagisawa. "Wait-Free Solvability of Colorless Tasks in Anonymous Shared-Memory Model". In: *Stabilization, Safety, and Security of Distributed Systems - 18th International Symposium, SSS 2016, Lyon, France, November 7-10, 2016, Proceedings*. Ed. by Borzoo Bonakdarpour and Franck Petit. Vol. 10083. Lecture Notes in Computer Science. 2016, pp. 415–429. DOI: 10.1007/978-3-319-49259-9_32. URL: https://doi.org/10.1007/978-3-319-49259-9%5C_32.

[299] Leqi Zhu. "A tight space bound for consensus". In: *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*. Ed. by Daniel Wichs and Yishay Mansour. ACM, 2016, pp. 345–350. DOI: 10.1145/2897518.2897565. URL: https://doi.org/10.1145/2897518.2897565.

[300] Ittai Abraham et al. "Revisiting Fast Practical Byzantine Fault Tolerance". In: *CoRR* abs/1712.01367 (2017). arXiv: 1712.01367. URL: http://arxiv.org/abs/1712.01367.

[301] Ailidani Ailijiang et al. "Efficient Distributed Coordination at WAN-Scale". In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017, pp. 1575–1585. DOI: 10.1109/ICDCS.2017.274.

[302] Balaji Arun et al. "Speeding up Consensus by Chasing Fast Decisions". In: *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2017, pp. 49–60.

[303] Luiz Barroso et al. "Attack of the Killer Microseconds". In: *Commun. ACM* 60.4 (Mar. 2017), pp. 48–54. ISSN: 0001-0782. DOI: 10.1145/3015146. URL: https://doi.org/10.1145/3015146.

[304] Claire Capdevielle et al. "On the Uncontended Complexity of Anonymous Agreement". In: *Distrib. Comput.* 30.6 (Dec. 2017), pp. 459–468. ISSN: 0178-2770. DOI: 10.1007/s00446-017-0297-z. URL: https://doi.org/10.1007/s00446-017-0297-z.

[305] David Yu Cheng Chan, Vassos Hadzilacos, and Sam Toueg. "Life Beyond Set Agreement". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC '17. Washington, DC, USA: Association for Computing Machinery, 2017, pp. 345–354. ISBN: 9781450349925. DOI: 10.1145/3087801.3087822. URL: https://doi.org/10.1145/3087801.3087822.

[306] Paulo R. Coelho, Nicolas Schiper, and Fernando Pedone. "Fast Atomic Multicast". In: *47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2017, Denver, CO, USA, June 26-29, 2017*. IEEE Computer Society, 2017, pp. 37–48. DOI: 10.1109/DSN.2017.15. URL: https://doi.org/10.1109/DSN.2017.15.

[307] Natacha Crooks et al. "Seeing is Believing: A Client-Centric Specification of Database Isolation". In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. PODC '17. Washington, DC, USA: Association for Computing Machinery, 2017, pp. 73–82. ISBN: 9781450349925. DOI: 10.1145/3087801.3087802. URL: https://doi.org/10.1145/3087801.3087802.

[308] Damien Imbs et al. "Which Broadcast Abstraction Captures k-Set Agreement?" In: *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*. Ed. by Andréa W. Richa. Vol. 91. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 27:1–27:16. DOI: 10.4230/LIPICS.DISC.2017.27. URL: https://doi.org/10.4230/LIPIcs.DISC.2017.27.

[309] Marius Poke, Torsten Hoefler, and Colin W. Glass. "AllConcur: Leaderless Concurrent Atomic Broadcast". In: *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC '17. Washington, DC, USA: Association for Computing Machinery, 2017, pp. 205–218. ISBN: 9781450346993. DOI: 10.1145/3078597.3078598. URL: https://doi.org/10.1145/3078597.3078598.

[312] Gadi Taubenfeld. "Contention-sensitive data structures and algorithms". In: *Theoretical Computer Science* 677 (2017), pp. 41–55. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2017.03.017.

[313] Michael Wei et al. "vCorfu: A Cloud-Scale Object Store on a Shared Log". In: *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*. Ed. by Aditya Akella and Jon Howell. USENIX Association, 2017, pp. 35–49. URL: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/wei-michael.

[315] Victor Bushkov et al. "The PCL Theorem: Transactions Cannot Be Parallel, Consistent, and Live". In: *J. ACM* 66.1 (Dec. 2018). ISSN: 0004-5411. DOI: 10.1145/3266141. URL: https://doi.org/10.1145/3266141.

[316] Eli Daian et al. "A Wealth of Sub-Consensus Deterministic Objects". In: *32nd International Symposium on Distributed Computing, DISC 2018, New Orleans, LA, USA, October 15-19, 2018*. Ed. by Ulrich Schmid and Josef Widder. Vol. 121. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 17:1–17:17. DOI: 10.4230/LIPIcs.DISC.2018.17. URL: https://doi.org/10.4230/LIPIcs.DISC.2018.17.

[317] Faith Ellen, Rati Gelashvili, and Leqi Zhu. "Revisionist Simulations: A New Approach to Proving Space Lower Bounds". In: *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*. PODC '18. Egham, United Kingdom: Association for Computing Machinery, 2018, pp. 61–70. ISBN: 9781450357951. DOI: 10.1145/3212734.3212749. URL: https://doi.org/10.1145/3212734.3212749.

[318] Xin Jin et al. "Netchain: Scale-Free Sub-RTT Coordination". In: *Proceedings of the 15th USENIX Conference on Networked Systems Design and Implementation*. NSDI'18. Renton, WA, USA: USENIX Association, 2018, pp. 35–49. ISBN: 9781931971430.

[319] Yoram Moses and Katia Patkin. "Mutual exclusion as a matter of priority". In: *Theoretical Computer Science* 751 (2018). Structural Information and Communication Complexity, pp. 46–60. ISSN: 0304-3975. DOI: https://doi.org/10.1016/j.tcs.2016.12.015. URL: https://www.sciencedirect.com/science/article/pii/S0304397516307435.

[320] Achour Mostéfaoui, Matthieu Perrin, and Michel Raynal. "A Simple Object that Spans the Whole Consensus Hierarchy". In: *Parallel Process. Lett.* 28.2 (2018), 1850006:1–1850006:9. DOI: 10.1142/S0129626418500068. URL: https://doi.org/10.1142/S0129626418500068.

[323] Hanyu Zhao et al. "SDPaxos: Building Efficient Semi-Decentralized Geo-replicated State Machines". In: *Symposium on Cloud Computing (SoCC)*. 2018.

[324] Klaus v. Gleissenthall et al. "Pretend Synchrony: Synchronous Verification of Asynchronous Distributed Programs". In: *Proc. ACM Program. Lang.* 3.POPL (Jan. 2019). DOI: 10.1145/3290372. URL: https://doi.org/10.1145/3290372.

[325] Alexey Gotsman, Anatole Lefort, and Gregory V. Chockler. "White-Box Atomic Multicast". In: *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019*. IEEE, 2019, pp. 176–187. DOI: 10.1109/DSN.2019.00030. URL: https://doi.org/10.1109/DSN.2019.00030.

[326] Long Hoang Le et al. "DynaStar: Optimized Dynamic Partitioning for Scalable State Machine Replication". In: *39th IEEE International Conference on Distributed Computing Systems, ICDCS 2019, Dallas, TX, USA, July 7-10, 2019*. IEEE, 2019, pp. 1453–1465. DOI: 10.1109/ICDCS.2019.00145. URL: https://doi.org/10.1109/ICDCS.2019.00145.

[328] Sujaya Maiyya et al. "Unifying Consensus and Atomic Commitment for Effective Cloud Data Management". In: *Proc. VLDB Endow.* 12.5 (2019), pp. 611–623. DOI: 10.14778/3303753.3303765. URL: http://www.vldb.org/pvldb/vol12/p611-maiyya.pdf.

[332] Zhaoguo Wang et al. "On the Parallels between Paxos and Raft, and How to Port Optimizations". In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. PODC '19. Toronto ON, Canada: Association for Computing Machinery, 2019, pp. 445–454. ISBN: 9781450362177. DOI: 10.1145/3293611.3331595. URL: https://doi.org/10.1145/3293611.3331595.

[333] Michael Zheludkov and Shamim Bhuiyan. *The Apache Ignite Book*. Lulu.com, 2019. ISBN: 0359439373.

[334] Marcos K. Aguilera et al. "Microsecond Consensus for Microsecond Applications". In: *CoRR* abs/2010.06288 (2020). arXiv: 2010.06288. URL: https://arxiv.org/abs/2010.06288.

[335] Matthew Burke, Audrey Cheng, and Wyatt Lloyd. "Gryff: Unifying Consensus and Shared Registers". In: *Symposium on Networked Systems Design and Implementation (NSDI)*. 2020.

[336] Faith Ellen et al. "A complexity-based classification for multiprocessor synchronization". In: *Distributed Comput.* 33.2 (2020), pp. 125–144. DOI: 10.1007/s00446-019-00361-3. URL: https://doi.org/10.1007/s00446-019-00361-3.

[339] Rachid Guerraoui et al. "Efficient Multi-Word Compare and Swap". In: *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. Ed. by Hagit Attiya. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 4:1–4:19. DOI: 10.4230/LIPIcs.DISC.2020.4. URL: https://doi.org/10.4230/LIPIcs.DISC.2020.4.

[340] Heidi Howard and Richard Mortier. "Paxos vs Raft: Have we reached consensus on distributed consensus?" In: *CoRR* abs/2004.05074 (2020). arXiv: 2004.05074. URL: https://arxiv.org/abs/2004.05074.

[341] Antonios Katsarakis et al. "Hermes: A Fast, Fault-Tolerant and Linearizable Replication Protocol". In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '20. Lausanne, Switzerland: Association for Computing Machinery, 2020, pp. 201–217. ISBN: 9781450371025. DOI: 10.1145/3373376.3378496. URL: https://doi.org/10.1145/3373376.3378496.

[342] Kyle Kingsbury and Peter Alvaro. "Elle: Inferring Isolation Anomalies from Experimental Observations". In: *Proc. VLDB Endow.* 14.3 (Nov. 2020), pp. 268–280. ISSN: 2150-8097. DOI: 10.14778/3430915.3430918. URL: https://doi.org/10.14778/3430915.3430918.

[343] Tanakorn Leesatapornwongsa et al. "Transactuations: Where Transactions Meet the Physical World". In: *ACM Trans. Comput. Syst.* 36.4 (May 2020). ISSN: 0734-2071. DOI: 10.1145/3380907. URL: https://doi.org/10.1145/3380907.

[344] Khiem Ngo, Siddhartha Sen, and Wyatt Lloyd. "Tolerating Slowdowns in Replicated State Machines using Copilots". In: *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 583–598. ISBN: 978-1-939133-19-9. URL: https://www.usenix.org/conference/osdi20/presentation/ngo.

[348] Rebecca Taft et al. "CockroachDB: The Resilient Geo-Distributed SQL Database". In: *International Conference on Management of Data (SIGMOD)*. 2020.

[349] Manuel Bravo et al. "UniStore: A fault-tolerant marriage of causal and strong consistency". In: *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. Ed. by Irina Calciu and Geoff Kuenning. USENIX Association, 2021, pp. 923–937. URL: https://www.usenix.org/conference/atc21/presentation/bravo.

[352] Idit Keidar et al. "All You Need is DAG". In: *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. PODC'21. Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 165–175. ISBN: 9781450385480. DOI: 10.1145/3465084.3467905. URL: https://doi.org/10.1145/3465084.3467905.

[353] Idit Keidar et al. "All You Need is DAG". In: *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. PODC'21. Virtual Event, Italy: Association for Computing Machinery, 2021, pp. 165–175. ISBN: 9781450385480. DOI: 10.1145/3465084.3467905. URL: https://doi.org/10.1145/3465084.3467905.

[354] Long Hoang Le et al. "RamCast: RDMA-based atomic multicast". In: *Middleware '21: 22nd International Middleware Conference, Québec City, Canada, December 6 - 10, 2021*. Ed. by Kaiwen Zhang et al. ACM, 2021, pp. 172–184. DOI: 10.1145/3464298.3493393. URL: https://doi.org/10.1145/3464298.3493393.

[357] Sujaya Maiyya et al. "Errata for "Unifying Consensus and Atomic Commitment for Effective Cloud Data Management"". In: *Proc. VLDB Endow.* 14.7 (2021), p. 1166. DOI: 10.14778/3450980.3450985. URL: http://www.vldb.org/pvldb/vol14/p1166-maiyya.pdf.

[358] Sarah Tollman, Seo Jin Park, and John K. Ousterhout. "EPaxos Revisited". In: *Symposium on Networked Systems Design and Implementation (NSDI)*. 2021.

[360] Leqi Zhu. "A Tight Space Bound for Consensus". In: *SIAM J. Comput.* 50.3 (2021). DOI: 10.1137/16M1096785. URL: https://doi.org/10.1137/16M1096785.

[362] Manuel Bravo, Gregory V. Chockler, and Alexey Gotsman. "Making Byzantine consensus live". In: *Distributed Comput.* 35.6 (2022), pp. 503–532. DOI: 10.1007/S00446-022-00432-Y. URL: https://doi.org/10.1007/s00446-022-00432-y.

[363] Armando Castañeda, Yannai A. Gonczarowski, and Yoram Moses. "Unbeatable consensus". In: *Distributed Comput.* 35.2 (2022), pp. 123–143. DOI: 10.1007/s00446-021-00417-3. URL: https://doi.org/10.1007/s00446-021-00417-3.

[364] Bogdan S. Chlebus, Dariusz R. Kowalski, and Jan Olkowski. "Brief Announcement: Deterministic Consensus and Checkpointing with Crashes: Time and Communication Efficiency". In: *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. PODC'22. Salerno, Italy: Association for Computing Machinery, 2022, pp. 106–108. ISBN: 9781450392624. DOI: 10.1145/3519270.3538471. URL: https://doi.org/10.1145/3519270.3538471.

[365] Vitor Enes. "Planet-scale leaderless consensus". PhD thesis. University of Minho, Portugal, 2022. URL: https://hdl.handle.net/1822/81307.

[366] Igor Konnov, Markus Kuppe, and Stephan Merz. "Specification and Verification with the TLA$^+$ Trifecta: TLC, Apalache, and TLAPS". In: *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles - 11th International Symposium, ISoLA 2022, Rhodes, Greece, October 22-30, 2022, Proceedings, Part I*. Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 13701. Lecture Notes in Computer Science. Springer, 2022, pp. 88–105. DOI: 10.1007/978-3-031-19849-6_6. URL: https://doi.org/10.1007/978-3-031-19849-6%5C_6.

[368] Sean Ovens. "The Space Complexity of Consensus from Swap". In: *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*. PODC'22. Salerno, Italy: Association for Computing Machinery, 2022, pp. 176–186. ISBN: 9781450392624. DOI: 10.1145/3519270.3538420. URL: https://doi.org/10.1145/3519270.3538420.

[373] Mojtaba Eslahi-Kelorazi, Long Hoang Le, and Fernando Pedone. "Heron: Scalable State Machine Replication on Shared Memory". In: *53rd Annual IEEE/IFIP International Conference on Dependable Systems and Network, DSN 2023, Porto, Portugal, June 27-30, 2023*. IEEE, 2023, pp. 138–150. DOI: 10.1109/DSN58367.2023.00025. URL: https://doi.org/10.1109/DSN58367.2023.00025.

[374] *A range of snapshot isolations*. URL: https://jepsen.io/consistency/models/snapshot-isolation#a-range-of-snapshot-isolations.

[375] *Apache Gora*. http://gora.apache.org.

[376] *Atomix GitHub repository*. URL: https://github.com/atomix/atomix.

[377] Cassandra. *Cassandra Enhancement Proposal (CEP-15): General Purpose Transactions*. URL: https://cwiki.apache.org/confluence/display/CASSANDRA/CEP-15%3A+General+Purpose+Transactions.

[378] Brian A. Coan and Jennifer L. Welch. "Modular Construction of Nearly Optimal Byzantine Agreement Protocols". In: pp. 295–306.

[379] *Epaxos (fixed) GitHub repository*. URL: https://github.com/otrack/epaxos.

[380] *Epaxos GitHub repository*. URL: https://github.com/efficient/epaxos.

[381] *etcd GitHub repository*. URL: https://github.com/coreos/etcd.

[383] *H-Store*. URL: https://github.com/apavlo/h-store.

[384] Ittai Abraham Heidi Howard. *Raft does not Guarantee Liveness in the face of Network Faults*. URL: https://decentralizedthoughts.github.io/2020-12-12-raft-liveness-full-omission/.

[385] *Improved Bulk Write API and One-Shot Transactions Support*. URL: https://jira.mongodb.org/browse/DRIVERS-716.

[386] *Jepsen*. URL: https://jepsen.io.

[387] *MySQL Group Replication - Transaction life cycle explained*. URL: https://dev.mysql.com/blog-archive/mysql-group-replication-transaction-life-cycle-explained/.

[388] *Quint Lang*. URL: https://github.com/informalsystems/quint.

[389] *Raft GitHub repository*. URL: https://raft.github.io/.

[392] *Transaction Processing Performance Council (TPC) - Benchmark C*. URL: http://www.tpc.org/tpcc/.

[393] *Transactional Synchronization Extensions*. URL: https://en.wikipedia.org/wiki/Transactional_Synchronization_Extensions.

[394] *What is FaunaDB?* URL: https://docs.fauna.com/fauna/current/introduction.html.

[395] Wikipedia. *Double compare-and-swap*. URL: https://en.wikipedia.org/wiki/Double_compare-and-swap.

[396] *YugabyteDB*. https://www.yugabyte.com/.

[397] ZooKeeper. *ZooKeeper GitHub repository*. URL: https://github.com/apache/zookeeper.

Personnal Bibliography

[150] Pierre Sutra, Marc Shapiro, and João Pedro Barreto. "An asynchronous, decentralised commitment protocol for semantic optimistic replication". In: *CoRR* abs/cs/0612086 (2006). arXiv: cs/0612086. URL: http://arxiv.org/abs/cs/0612086.

[158] Claudia-Lavinia Ignat et al. "A comparison of optimistic approaches to collaborative editing of Wiki pages". In: *Proceedings of the 3rd International Conference on Collaborative Computing: Networking, Applications and Worksharing, White Plains, New York, USA, November 12-15, 2007*. Ed. by Juan Quemada and Tao Zhang. IEEE, 2007, pp. 474–483. DOI: 10.1109/COLCOM.2007.4553878. URL: https://doi.org/10.1109/COLCOM.2007.4553878.

[162] Pierre Sutra, João Pedro Barreto, and Marc Shapiro. "Decentralised Commitment for Optimistic Semantic Replication". In: *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*. Ed. by Robert Meersman and Zahir Tari. Vol. 4803. Lecture Notes in Computer Science. Springer, 2007, pp. 318–335. DOI: 10.1007/978-3-540-76848-7_21. URL: https://doi.org/10.1007/978-3-540-76848-7%5C_21.

[163] Pierre Sutra and Marc Shapiro. "Comparing Optimistic Database Replication Techniques". In: *23èmes Journées Bases de Données Avancées, BDA 2007, Marseille, France, 23-26 Octobre 2007, Actes (Informal Proceedings)*. Ed. by Omar Boucelma et al. 2007.

[168] Lamia Benmouffok et al. "Telex: Principled System Support for Write-Sharing in Collaborative Applications". In: *CoRR* abs/0805.4680 (2008). arXiv: 0805.4680. URL: http://arxiv.org/abs/0805.4680.

[182] Pierre Sutra and Marc Shapiro. "Fault-Tolerant Partial Replication in Large-Scale Database Systems". In: *Euro-Par 2008 - Parallel Processing, 14th International Euro-Par Conference, Las Palmas de Gran Canaria, Spain, August 26-29, 2008, Proceedings*. Ed. by Emilio Luque, Tomàs Margalef, and Domingo Benitez. Vol. 5168. Lecture Notes in Computer Science. Springer, 2008, pp. 404–413. DOI: 10.1007/978-3-540-85451-7_44. URL: https://doi.org/10.1007/978-3-540-85451-7%5C_44.

[183] Pierre Sutra and Marc Shapiro. "Fault-Tolerant Partial Replication in Large-Scale Database Systems". In: *CoRR* abs/0802.0137 (2008). arXiv: 0802.0137. URL: http://arxiv.org/abs/0802.0137.

[195] Nicolas Schiper, Pierre Sutra, and Fernando Pedone. "Genuine versus Non-Genuine Atomic Multicast Protocols for Wide Area Networks: An Empirical Study". In: *28th IEEE Symposium on Reliable Distributed Systems (SRDS 2009), Niagara Falls, New York, USA, September 27-30, 2009*. IEEE Computer Society, 2009, pp. 166–175. DOI: 10.1109/SRDS.2009.12. URL: https://doi.org/10.1109/SRDS.2009.12.

[208] Nicolas Schiper, Pierre Sutra, and Fernando Pedone. "P-Store: Genuine Partial Replication in Wide Area Networks". In: *29th IEEE Symposium on Reliable Distributed Systems (SRDS 2010), New Delhi, Punjab, India, October 31 - November 3, 2010*. IEEE Computer Society, 2010, pp. 214–224. DOI: 10.1109/SRDS.2010.32. URL: https://doi.org/10.1109/SRDS.2010.32.

[215] Zohir Bouzid, Pierre Sutra, and Corentin Travers. "Anonymous Agreement: The Janus Algorithm". In: *Principles of Distributed Systems - 15th International Conference, OPODIS 2011, Toulouse, France, December 13-16, 2011. Proceedings*. Ed. by Antonio Fernández Anta, Giuseppe Lipari, and Matthieu Roy. Vol. 7109. Lecture Notes in Computer Science. Springer, 2011, pp. 175–190. DOI: 10.1007/978-3-642-25873-2_13. URL: https://doi.org/10.1007/978-3-642-25873-2%5C_13.

[223] Jonathan Michaux et al. "A semantically rich approach for collaborative model edition". In: *Proceedings of the 2011 ACM Symposium on Applied Computing (SAC), TaiChung, Taiwan, March 21 - 24, 2011*. Ed. by William C. Chu et al. ACM, 2011, pp. 1470–1475. DOI: 10.1145/1982185.1982500. URL: https://doi.org/10.1145/1982185.1982500.

[226] Pierre Sutra and Marc Shapiro. "Fast Genuine Generalized Consensus". In: *30th IEEE Symposium on Reliable Distributed Systems (SRDS 2011), Madrid, Spain, October 4-7, 2011*. IEEE Computer Society, 2011, pp. 255–264. DOI: 10.1109/SRDS.2011.38. URL: https://doi.org/10.1109/SRDS.2011.38.

[228] Masoud Saeida Ardekani et al. "The space complexity of transactional interactive reads". In: *Proceedings of the 1st International Workshop on Hot Topics in Cloud Data Processing, HotCDP '12, Bern, Switzerland, April 10, 2012*. Ed. by Christof Fetzer, Flavio Junqueira, and Peter R. Pietzuch. ACM, 2012, 4:1–4:5. DOI: 10.1145/2169090.2169094. URL: https://doi.org/10.1145/2169090.2169094.

[246] Masoud Saeida Ardekani, Pierre Sutra, and Marc Shapiro. "Non-monotonic Snapshot Isolation: Scalable and Strong Consistency for Geo-replicated Transactional Systems". In: *IEEE 32nd Symposium on Reliable Distributed Systems, SRDS 2013, Braga, Portugal, 1-3 October 2013*. IEEE Computer Society, 2013, pp. 163–172. DOI: 10.1109/SRDS.2013.25. URL: https://doi.org/10.1109/SRDS.2013.25.

[247] Masoud Saeida Ardekani et al. "Non-Monotonic Snapshot Isolation". In: *CoRR* abs/1306.3906 (2013). arXiv: 1306.3906. URL: http://arxiv.org/abs/1306.3906.

[248] Masoud Saeida Ardekani et al. "On the Scalability of Snapshot Isolation". In: *Euro-Par 2013 Parallel Processing - 19th International Conference, Aachen, Germany, August 26-30, 2013. Proceedings*. Ed. by Felix Wolf, Bernd Mohr, and Dieter an Mey. Vol. 8097. Lecture Notes in Computer Science. Springer, 2013, pp. 369–381. DOI: 10.1007/978-3-642-40047-6_39. URL: https://doi.org/10.1007/978-3-642-40047-6%5C_39.

[258] Pierre Sutra, Etienne Rivière, and Pascal Felber. "An Efficient Distributed Obstruction-Free Compare-And-Swap Primitive". In: *CoRR* abs/1309.2772 (2013). arXiv: 1309.2772. URL: http://arxiv.org/abs/1309.2772.

[259] José Valerio et al. "Evaluating the Price of Consistency in Distributed File Storage Services". In: *Distributed Applications and Interoperable Systems - 13th IFIP WG 6.1 International Conference, DAIS 2013, Held as Part of the 8th International Federated Conference on Distributed Computing Techniques, DisCoTec 2013, Florence, Italy, June 3-5, 2013. Proceedings*. Ed. by Jim Dowling and François Taïani. Vol. 7891. Lecture Notes in Computer Science. Springer, 2013, pp. 141–154. DOI: 10.1007/978-3-642-38541-4_11. URL: https://doi.org/10.1007/978-3-642-38541-4%5C_11.

[260] Masoud Saeida Ardekani, Pierre Sutra, and Marc Shapiro. "G-DUR: a middleware for assembling, analyzing, and improving transactional protocols". In: *Proceedings of the 15th International Middleware Conference, Bordeaux, France, December 8-12, 2014*. Ed. by Laurent Réveillère, Lucy Cherkasova, and François Taïani. ACM, 2014, pp. 13–24. DOI: 10.1145/2663165.2663336. URL: https://doi.org/10.1145/2663165.2663336.

[265] Pascal Felber et al. "On the Support of Versioning in Distributed Key-Value Stores". In: *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014*. IEEE Computer Society, 2014, pp. 95–104. DOI: 10.1109/SRDS.2014.35. URL: https://doi.org/10.1109/SRDS.2014.35.

[266] Raluca Halalai et al. "ZooFence: Principled Service Partitioning and Application to the ZooKeeper Coordination Service". In: *33rd IEEE International Symposium on Reliable Distributed Systems, SRDS 2014, Nara, Japan, October 6-9, 2014*. IEEE Computer Society, 2014, pp. 67–78. DOI: 10.1109/SRDS.2014.41. URL: https://doi.org/10.1109/SRDS.2014.41.

[270] Pierre Sutra, Etienne Rivière, and Pascal Felber. "A Practical Distributed Universal Construction with Unknown Participants". In: *Principles of Distributed Systems - 18th International Conference, OPODIS 2014, Cortina d'Ampezzo, Italy, December 16-19, 2014. Proceedings*. Ed. by Marcos K. Aguilera, Leonardo Querzoni, and Marc Shapiro. Vol. 8878. Lecture Notes in Computer Science. Springer, 2014, pp. 485–500. DOI: 10.1007/978-3-319-14472-6_32. URL: https://doi.org/10.1007/978-3-319-14472-6%5C_32.

[272] Zohir Bouzid, Michel Raynal, and Pierre Sutra. "Anonymous Obstruction-Free (n, k)-Set Agreement with n-k+1 Atomic Read/Write Registers". In: *19th International Conference on Principles of Distributed Systems, OPODIS 2015, December 14-17, 2015, Rennes, France*. Ed. by Emmanuelle Anceaume, Christian Cachin, and Maria Gradinariu Potop-Butucaru. Vol. 46. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, 18:1–18:17. DOI: 10.4230/LIPIcs.OPODIS.2015.18. URL: https://doi.org/10.4230/LIPIcs.OPODIS.2015.18.

[273] Zohir Bouzid, Michel Raynal, and Pierre Sutra. "Anonymous Obstruction-free (n, k)-Set Agreement with n-k+1 Atomic Read/Write Registers". In: *CoRR* abs/1507.00474 (2015). arXiv: 1507.00474. URL: http://arxiv.org/abs/1507.00474.

[279] Do Le Quoc et al. "UniCrawl: A Practical Geographically Distributed Web Crawler". In: *8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015*. Ed. by Calton Pu and Ajay Mohindra. IEEE Computer Society, 2015, pp. 389–396. DOI: 10.1109/CLOUD.2015.59. URL: https://doi.org/10.1109/CLOUD.2015.59.

[281] Valerio Schiavoni et al. "TOPiCo: detecting most frequent items from multiple high-rate event streams". In: *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems, DEBS '15, Oslo, Norway, June 29 - July 3, 2015*. Ed. by Frank Eliassen and Roman Vitenberg. ACM, 2015, pp. 58–67. DOI: 10.1145/2675743.2771838. URL: https://doi.org/10.1145/2675743.2771838.

[284] Tarek Ahmed-Nacer, Pierre Sutra, and Denis Conan. "The Convoy Effect in Atomic Multicast". In: *35th IEEE Symposium on Reliable Distributed Systems Workshops, SRDS 2016 Workshop, Budapest, Hungary, September 26, 2016*. IEEE Computer Society, 2016, pp. 67–72. DOI: 10.1109/SRDSW.2016.22. URL: https://doi.org/10.1109/SRDSW.2016.22.

[310] Tuanir França Rezende et al. "On Making Generalized Paxos Practical". In: *31st IEEE International Conference on Advanced Information Networking and Applications, AINA 2017, Taipei, Taiwan, March 27-29, 2017*. Ed. by Leonard Barolli et al. IEEE Computer Society, 2017, pp. 347–354. DOI: 10.1109/AINA.2017.94. URL: https://doi.org/10.1109/AINA.2017.94.

[311] Pierre Sutra et al. "CRESON: Callable and Replicated Shared Objects over NoSQL". In: *37th IEEE International Conference on Distributed Computing Systems, ICDCS 2017, Atlanta, GA, USA, June 5-8, 2017*. Ed. by Kisung Lee and Ling Liu. IEEE Computer Society, 2017, pp. 115–128. DOI: 10.1109/ICDCS.2017.239. URL: https://doi.org/10.1109/ICDCS.2017.239.

[314] Zohir Bouzid, Michel Raynal, and Pierre Sutra. "Anonymous obstruction-free (n, k)-set agreement with n-k+1 atomic read/write registers". In: *Distributed Comput.* 31.2 (2018), pp. 99–117. DOI: 10.1007/s00446-017-0301-7. URL: https://doi.org/10.1007/s00446-017-0301-7.

[321] Marc Shapiro and Pierre Sutra. "Database Consistency Models". In: *CoRR* abs/1804.00914 (2018). arXiv: 1804.00914. URL: http://arxiv.org/abs/1804.00914.

[322] Pierre Sutra et al. "Boosting Transactional Memory with Stricter Serializability". In: *Coordination Models and Languages - 20th IFIP WG 6.1 International Conference, COORDINATION 2018, Held as Part of the 13th International Federated Conference on Distributed Computing Techniques, DisCoTec 2018, Madrid, Spain, June 18-21, 2018. Proceedings*. Ed. by Giovanna Di Marzo Serugendo and Michele Loreti. Vol. 10852. Lecture Notes in Computer Science. Springer, 2018, pp. 231–251. DOI: 10.1007/978-3-319-92408-3_11. URL: https://doi.org/10.1007/978-3-319-92408-3%5C_11.

[327] Pedro García López et al. "ServerMix: Tradeoffs and Challenges of Serverless Data Analytics". In: *CoRR* abs/1907.11465 (2019). arXiv: 1907.11465. URL: http://arxiv.org/abs/1907.11465.

[329] Daniel Barcelona Pons et al. "On the FaaS Track: Building Stateful Distributed Applications with Serverless Architectures". In: *Proceedings of the 20th International Middleware Conference, Middleware 2019, Davis, CA, USA, December 9-13, 2019*. ACM, 2019, pp. 41–54. DOI: 10.1145/3361525.3361535. URL: https://doi.org/10.1145/3361525.3361535.

[330] Marc Shapiro and Pierre Sutra. "Database Consistency Models". In: *Encyclopedia of Big Data Technologies*. Ed. by Sherif Sakr and Albert Y. Zomaya. Springer, 2019. DOI: 10.1007/978-3-319-63962-8_203-1. URL: https://doi.org/10.1007/978-3-319-63962-8%5C_203-1.

[331] Pierre Sutra. "On the correctness of Egalitarian Paxos". In: *CoRR* abs/1906.10917 (2019). arXiv: 1906.10917. URL: http://arxiv.org/abs/1906.10917.

[337] Vitor Enes et al. "State-machine replication for planet-scale systems". In: *EuroSys '20: Fifteenth EuroSys Conference 2020, Heraklion, Greece, April 27-30, 2020*. Ed. by Angelos Bilas et al. ACM, 2020, 24:1–24:15. DOI: 10.1145/3342195.3387543. URL: https://doi.org/10.1145/3342195.3387543.

[338] Vitor Enes et al. "State-Machine Replication for Planet-Scale Systems (Extended Version)". In: *CoRR* abs/2003.11789 (2020). arXiv: 2003.11789. URL: https://arxiv.org/abs/2003.11789.

[345] Tuanir França Rezende and Pierre Sutra. "Leaderless State-Machine Replication: Specification, Properties, Limits". In: *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*. Ed. by Hagit Attiya. Vol. 179. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 24:1–24:17. DOI: 10.4230/LIPIcs.DISC.2020.24. URL: https://doi.org/10.4230/LIPIcs.DISC.2020.24.

[346] Tuanir França Rezende and Pierre Sutra. "Leaderless State-Machine Replication: Specification, Properties, Limits (Extended Version)". In: *CoRR* abs/2008.02512 (2020). arXiv: 2008.02512. URL: https://arxiv.org/abs/2008.02512.

[347] Pierre Sutra. "On the correctness of Egalitarian Paxos". In: *Inf. Process. Lett.* 156 (2020), p. 105901. DOI: 10.1016/j.ipl.2019.105901. URL: https://doi.org/10.1016/j.ipl.2019.105901.

[350] Vitor Enes et al. "Efficient replication via timestamp stability". In: *EuroSys '21: Sixteenth European Conference on Computer Systems, Online Event, United Kingdom, April 26-28, 2021*. Ed. by Antonio Barbalace et al. ACM, 2021, pp. 178–193. DOI: 10.1145/3447786.3456236. URL: https://doi.org/10.1145/3447786.3456236.

[351] Vitor Enes et al. "Efficient Replication via Timestamp Stability (Extended Version)". In: *CoRR* abs/2104.01142 (2021). arXiv: 2104.01142. URL: https://arxiv.org/abs/2104.01142.

[355] Anatole Lefort et al. "J-NVM: Off-heap Persistent Objects in Java". In: *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*. Ed. by Robbert van Renesse and Nickolai Zeldovich. ACM, 2021, pp. 408–423. DOI: 10.1145/3477132.3483579. URL: https://doi.org/10.1145/3477132.3483579.

[356] Aurèle Mahéo, Pierre Sutra, and Tristan Tarrant. "The serverless shell". In: *Middleware '21: Proceedings of the 22nd International Middleware Conference: Industrial Track, Virtual Event / Québec City, Canada, December 6 - 10, 2021*. Ed. by Kaiwen Zhang et al. ACM, 2021, pp. 9–15. DOI: 10.1145/3491084.3491426. URL: https://doi.org/10.1145/3491084.3491426.

[359] Ilyas Toumlilt, Pierre Sutra, and Marc Shapiro. "Highly-available and consistent group collaboration at the edge with colony". In: *Middleware '21: 22nd International Middleware Conference, Québec City, Canada, December 6 - 10, 2021*. Ed. by Kaiwen Zhang et al. ACM, 2021, pp. 336–351. DOI: 10.1145/3464298.3493405. URL: https://doi.org/10.1145/3464298.3493405.

[361] Zohir Bouzid, Pierre Sutra, and Corentin Travers. "Agreeing within a few writes". In: *Theor. Comput. Sci.* 922 (2022), pp. 283–299. DOI: 10.1016/j.tcs.2022.04.030. URL: https://doi.org/10.1016/j.tcs.2022.04.030.

[367] Pedro García López et al. "Trade-Offs and Challenges of Serverless Data Analytics". In: *Technologies and Applications for Big Data Value*. Ed. by Edward Curry et al. Springer, 2022, pp. 41–61. DOI: 10.1007/978-3-030-78307-5_3. URL: https://doi.org/10.1007/978-3-030-78307-5%5C_3.

[369] Daniel Barcelona Pons et al. "Stateful Serverless Computing with Crucial". In: *ACM Trans. Softw. Eng. Methodol.* 31.3 (2022), 39:1–39:38. DOI: 10.1145/3490386. URL: https://doi.org/10.1145/3490386.

[370] Pierre Sutra. "Brief Announcement: The Weakest Failure Detector for Genuine Atomic Multicast". In: *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*. Ed. by Alessia Milani and Philipp Woelfel. ACM, 2022, pp. 268–270. DOI: 10.1145/3519270.3538467. URL: https://doi.org/10.1145/3519270.3538467.

[371] Pierre Sutra. "The Weakest Failure Detector for Genuine Atomic Multicast". In: *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*. Ed. by Christian Scheideler. Vol. 246. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 35:1–35:19. DOI: 10.4230/LIPIcs.DISC.2022.35. URL: https://doi.org/10.4230/LIPIcs.DISC.2022.35.

[372] Pierre Sutra. "The Weakest Failure Detector for Genuine Atomic Multicast (Extended Version)". In: *CoRR* abs/2208.07650 (2022). DOI: 10.48550/arXiv.2208.07650. arXiv: 2208.07650. URL: https://doi.org/10.48550/arXiv.2208.07650.

[382] Alexis Le Glaunec and Pierre Sutra. *Fixing Egalitarian Paxos*. unpublished research report.

[390] Fedor Ryabini, Alexey Gotsmam, and Pierre Sutra. *A problematic run in Apache Accord*. URL: https://issues.apache.org/jira/browse/CASSANDRA-18365.

[391] Fedor Ryabinin, Alexey Gotsman, and Pierre Sutra. "Swiftpaxos Fast Geo-Replicated State Machines". In: *NSDI '24: 21st USENIX Symposium on Networked Systems Design and Implementation (to appear)*.

Open Problems

ECOLE DOCTORALE

Titre : Contributions Pratiques et Théoriques à la Réplication de Machine à États

Mots clés : systèmes répartis, calcul réparti, mémoire partagée, consensus, réplication de machine à états, transactions

Résumé : La réplication de machine à états est un des piliers fondamentaux des infrastructures informatiques modernes. Cette technique assure la durabilité des données et offre une haute disponibilité aux services les plus essentiels. Elle est construite à l'aide de protocoles, tels que Paxos ou Raft, qui permettent de maintenir les répliques continûment à jour. Cette habilitation à diriger des recherches présente plusieurs contributions à *la théorie et la pratique de la réplication de machine à états*.

Dans les algorithmes usuels de la réplication de machine à états, les opérations qui accèdent au service répliqué sont ordonnées par une machine distincte, ou leader. Notre première contribution est l'identification d'une nouvelle classe de protocoles dits *sans leader*. Les protocoles sans leader permettent que chaque réplique progresse indépendamment en contactant un quorum de ses pairs. Nous introduisons plusieurs propriétés afin de caractériser cette classe de protocoles et nous étudions leurs limites. Par ailleurs, nous présentons des protocoles efficaces pour les systèmes géo-répartis s'appuyant sur l'indépendance des fautes.

Quand les données du service sont morcelées entre les répliques, on parle de réplication partielle. Notre seconde contribution est de définir la réplication *partielle* de machine à états pour de tels environnements. Là aussi, nous présentons des protocoles efficaces. Par ailleurs, nous identifions les conditions de synchronie minimales qui permette à une telle abstraction d'être mise en œuvre.

D'un point de vue calculabilité, les systèmes répartis sont équivalents à ceux à mémoire partagée dans de nombreux cas pratiques. Notre dernière contribution consiste en plusieurs résultats sur le coût et les conditions de résolubilité en mémoire partagée de l'accord et ses variantes (objet adopt-commit, k-accord, et par extension, la réplication de machine à états).

Title : Contributions to the Practice and Theory of State-Machine Replication

Keywords : distributed systems, distributed computing, shared memory, consensus, state-machine replication, transactions

Abstract : State-machine replication (SMR) is one of the cornerstones of modern computing infrastructures. It ensures the durability of data and provides high availability to essential services. SMR is built atop distributed protocols such as Paxos and Raft that keep in sync the service replicas. In this thesis, we present several *contributions to the theory and practice of SMR*.

Our first contribution is to identify a new class of leaderless protocols. In typical SMR protocols, all the operations accessing the service are ordered by a leader replica. On the contrary, *leaderless* protocols allow replicas to make progress independently by contacting a nearby quorum. We introduce several key properties for leaderless SMR and study some of its limits. We also contribute new protocols for geo-distributed systems that leverage failure independence in this setting.

When data is sharded across replicas, one talks about partial replication. Our second contribution is to define *partial* state-machine replication (PSMR) for such environments. We contribute several efficient PSMR protocols and identify the minimal synchrony assumptions under which such an abstraction is implementable.

Distributed systems are computationally equivalent to shared memory ones in many practical cases. Our last contribution consists in several results exploring the costs and conditions under which agreement (adopt-commit object, k-set agreement, and by extension SMR) is solvable in shared memory.

Institut Polytechnique de Paris
91120 Palaiseau, France