COMMUNICATION-CLOSED LAYERS AS PARADIGM FOR DISTRIBUTED SYSTEMS: A MANIFESTO

Cezara Drăgoi[1],
Marijana Lazić[2],
Josef Widder[2]*

[1]INRIA / ENS Paris,
Paris, France
[2]TU Wien,
Vienna, Austria

Abstract:
Distributed computations are characterized by a partial order over events: two concurrent events at different processes may be re-ordered without changing the outcome of the computation. For systems that are composed of so-called *communication-closed layers*, this partial-order argument has been used by Elrad and Francez [13] to reduce the reasoning about distributed systems to a specific sequential form. We discuss existing techniques for communication-closed layers, and discuss applications to automated verification of state-of-the-art distributed systems.

Keywords:
computer-aided verification, reduction, fault-tolerant distributed systems, formal methods.

1. INTRODUCTION

As more and more people and institutions use services on the Internet on a daily basis, and as computers are increasingly used in embedded control systems of cars, aircraft, and medical devices, our society depends to a greater extent on the correct operation of distributed computing systems. Many of the applications require us to design fault-tolerant systems.

The classic application domain for fault-tolerant distributed systems were safety-critical systems [29] like cars and civil airplanes. In order to achieve very high reliability (that is, very low probability of system failure), components are replicated, and fault-tolerant distributed algorithms ensure that the collection of these components behave as one reliable component.

The advent of data centers and cloud computing over the Internet, led to design distributed systems that consists of thousands of commodity computers in clouds. Such systems typically pose less severe reliability requirements than safety-critical systems. However, the huge number of involved com-modity computers means that single components can fail very often, and fault tolerance is required as faults become part of the normal operation. Hence we see more and more implementations of fault-tolerant distributed algorithms [48],[30], [36], [5] for data centers.

Correspondence:
Josef Widder

e-mail:
widder@forsyte.tuwien.ac.at

Supported by: the Austrian Science Fund (FWF) through the National Research Network RiSE (S11403 and S11405), project PRAVDA (P27722), and Doctoral College LogiCS (W1255-N23); and by the Vienna Science and Technology Fund (WWTF) through project APALACHE (ICT15-103).

DOI: 10.15308/Sinteza-2018-131-138

Currently we observe increasing interest in blockchain technoloqy [37], [16], [22], [4]. Here the need for distributed systems comes from the fact that a large number of persons, companies, etc., are interested in participating in a market place. Also these systems have to ensure some sort of fault tolerance. For instance, if a majority of the participants are honest, the system should work as expected.

Due to the criticality of the mentioned application domain, it is crucial to design computer systems in a way that ensures that they do not fail. To do so, one has to address two challenges: on one hand, we have to design means that tolerate partial failure that is outside the control of a system designer (such as power outages or bit-flips due to radiation), and on the other hand, find design faults (bugs) in order to fix them. The former is classically addressed by means of replication and fault-tolerant distributed algorithms, while the latter is dealt with by rigorous system and software engineering methods.

It is well-understood that fault tolerance in distributed systems is a non-trivial challenge. Classic results in distributed computing theory show that under general assumptions on the environment, fault tolerance is not achievable [32], [18], [40]. Hence, fault-tolerant distributed algorithms are based on quite involved assumptions on the environment, such as type of fault behavior, message delays, processing speeds. Consequently, that a distributed algorithm actually is correct rests on quite intricate properties of the considered systems. There is a quite elaborate theory [34] on how to prove correctness math-ematically. However, mathematical proofs (including those involving interactive theorem provers) require huge manual effort and a lot of time. Thus, in order to keep pace with the development of new distributed systems these ideas should be transferred into rigorous software engineering tools.

While methods such as static analysis, model checking, and SMT solving made impressive progress in the verification of sequential code, the required abstractions for distributed systems have not found their way into verification tools. The non-determinism due to faults, process step interleaving, and uncertain message delays lead to a combinatorial explosion of the state space and the execution space, which renders many existing verification methods a priori ineffective. At the same time, it is not well-understood in general what features of distributed systems would permit effective verification. The knowledge of some features would be an important guideline for "design for verification": From the beginning of the software life-cycle, a designer could follow guidelines that provide the promise that the resulting distributed system may be verified effectively.

In this paper we will discuss one such feature, which we find promising, namely designing processes with *communication-closed layers* [13]. We will show that for fault-tolerant systems, communication-closed layers allows us to do automated reasoning based on reduction theorems [33], [13], [6], which drastically reduces the execution space. We will discuss the peculiarities of distributed computations in Section II and the idea of reductions in Section III. In Section IV we review some existing results of reductions for communication-closed layers. Based on this, we motivate in Section V that in the context of fault-tolerant distributed systems, communication-closed layers should be a central design and verification paradigm in *design for verification*.

2. DISTRIBUTED COMPUTATIONS AND PARTIAL ORDERS

Given a sequential piece of code, once the input is fixed, the control flow of the code with the evaluation of variables defines *a single* execution (if we ignore a lot of details in modern compiler and processor design, such as, code optimization, caching, etc.). In sharp contrast, a distributed system consists of multiple processes each with its local control flow. In asynchronous systems, processes run independently, so that already all possible interleavings of steps of the distributed processes induce a typically huge execution space rather than a single execution. Additionally, these processes coordinate or communicate, either by means of shared memory or message passing. There exist effective verification techniques for shared memory. However, we are interested in message passing systems in this note. They require different verification techniques as they follow a different concurrency paradigm.

As already observed by Lamport [31] distributed compu-tations thus induce a partial order — the so-called happened before relation — of events in a distributed systems. Roughly speaking the send event of a message m happens before the receive event of m, and for each process p, local events at p are ordered according to the temporal order of their occurence. The happens before relation is the transitive closure of these relations for all messages and all processes. As a result, if in an execution events e_1 and e_2 happend directly one after the other at two distinct processes p and q, respectively, e_1 and e_2 may still be independent (not ordered according

to happened before). That is, neither the local control flows, nor the messages impose an order of e_1 and e_2 so that swapping e_1 and e_2 leads to a different execution, which (i) entails the same happened before relation, and (ii) is locally indistinguishable for processes p and q (and all other processes in the system).

We observe that asynchrony leads to a huge execution space. This, in turn, makes understanding and reasoning about the execution space hard, both for humans and computers. An important idea to handle this complexity is to structure the reasoning along the induced partial orders. Due to the mentioned partial order and indistinguishability arguments, the happened before relation can be understood as an equivalence relation between executions: all executions that have the same happened before relation over their events can be seen as falling into the same class. For many interesting specification, it is sufficient to check a representative execution from each class. Here we distinguish — very roughly — two approaches.

In so-called *partial-order reduction*, while searching the exe-cution space, executions are pruned that are "similar" to ones searched before [21], [46], [38]. In so-called *reduction* one proves a priori that every execution can be represented by an execution of specific form [33], [13]. Then verification procedures only need to consider executions of these specific forms.

3. REDUCTIONS

To the best of our knowledge, Lipton [33] was the first to highlight reduction as a proof method for concurrent systems. In his theory, processes execute sequences of statements, for instance, one process may be the sequence of statements *A, B, C*, and another process may be the sequence *X, Y* . Then, concurrent executions are interleaved sequences of state-ments. For example, *A, X, B, Y, C* is an execution, as well as *X, A, B, C, Y* . In the latter execution, the sequence *A, B, C* is said to be executed atomically.

He considered the classic semaphore operation *P (s)* and *V (s)*, for semaphore s. Then if *p*'s code is *P (s), B, V (s)*, he proves that all executions can be reduced to ones where *P (s), B, V (s)* occurs as uninterupted (atomic) block: Intu-itively, if an execution is interleaved with an event *A*'at a different process *p*', that is, *P (s), A*' , *B, V (s)*, then Lipton proves that *P (s)* can always be moved to the right, that is, *A*', *P (s), B, V (s)* is also an execution with the mentioned block. Similarly, all *V* operations are so-called *left movers*. Thus, Lipton's reduction consists in idetifying large blocks of local code between a *P* operation and its matching *V* operation that can be "moved" together. Then for verification (of reachability properties) it is sufficient to consider the executions where these blocks are executed atomically.

Lipton's reduction is widely used in verification of con-current systems. However, the idea of moving can also be used to generate other atomic steps. For instance, recently Konnov et al. [27], [28] considered symmetric systems of specific threshold automata, that is, processes that execute the same local code. Then if a given number *f* of processes, each performs event *A* one after the other, that is, the execution is *A, A,* . . . , *A*, this can be represented by a single (accelerated) transition $f \cdot A$. As a result, moving the As together leads to "shorter" executions, and indeed Konnov et al. prove that for safety and liveness it is sufficient to consider executions whose length can be bounded. While being very effective, this method a priori can be applied only to a restricted class of threshold automata.

In this paper we advocate for a classic reduction by Elrad and Francez [13], which they originally formulated in the context of CSP [23] (Communicating Sequential Processes). Consider a parallel composition of processes P_i, for $1 \le i \le n$, that is, $S = P_1 \,||\, P_2 \,||\, \ldots \,||\, P_n$. Further assume that each process P_i is a sequential composition of layers $L_i^1; L_i^2; \ldots; L_i^k$. Then they assume the following property: if for two processes *i* and *j*, the layers L_i^a and L_j^b communicate (have a synchronized event in CSP), then $a = b$. In other words, layers communicate only with layers of the same number, that is, are *communication-closed*. If we consider the parallel compositions of layers $L^k = L_1^k \,||\, L_2^k \,||\, \ldots \,||\, L_n^k$, then the central result — proved with a partial-order argument —is that instead of analyzing *S*, it is sufficient to analyze the sequential compositon of layers $S^l = L^1; L^2; \ldots L^k$. Observe that S^l has considerably fewer interleavings than *S*. For instance, in *S* events of layer 2 at process *p*, L_p^2 , might occur before events of layer 1 at process *q* (that is, L_q^1), while in S^l this cannot be the case.

4. COMMUNICATION-CLOSED LAYERS AS VERIFICATION PARADIGM

A. Reasoning about the sequential core

Chou and Gafni [10] introduce a design principle called *stratified decomposition* that is intimately related to communication-closed layers, but is formalized in

I/O automata, and targets distributed algorithms. They discuss in detail that complex distributed systems are designed according to an implicit sequential algorithm, whose executions are then tailored towards scenarios a designer has anticipated. How-ever, proving invariants of the asynchronous system typically ignores the sequential algorithm. To address this in [10], Chou and Gafni prove reduction theorems and apply them to the design of a minimum-weight spanning tree distributed algorithm. With a similar motivation, Stomp and Roever [42] formulate a principle (along with soundness proofs) for distributed programs that can be split up into subtasks that can be performed sequentially from a logical viewpointt, although in "reality" these tasks may be performed concurrently. In Section V we will discuss a similar idea for the verification of fault-tolerant distributed system implementations.

B. Communication-closure as proof strategy

We present a subjective selection of work that considers communication-closed layers that appears closest related to the verification approach we are advocating for.

a) Proving the closure: The result in [13] states that if a CSP program is communication-closed then the verification can be reduced to verification of a simpler CSP program. Given that, the first obvious question is how to check that a given CSP program is communication closed. This is ad-dressed in [20]. The paper shows that for a stronger notion of closure, the closure of the layer can be proved in terms of the layer itself. The basic strategy is to show that violations of communication closure are not reachable. We will discuss in Section V that the problem of proving closure also appears in verification of asynchronous code of fault-tolerant distributed systems.

b) Safe composition: Engelhardt and Moses [14], [15] analyze conditions under which distributed programs (or lay-ers) can be composed, if processes communicate by message passing. The considered composition is parallel for different processes, but every process executes the composed programs sequentially. The main goal is to find a binary relation between programs, such that if every two adjacent programs in a compositon are in this relation, then safety of all programs analyzed in isolation implies safety of the composition. They consider different communication semantics: communication over asynchronous order-preserving (FIFO) channels where messages might be duplicated or lost in [14], and reliable non-duplicating channels that are not order-preserving in [15].

A notion of a program Q *fitting after* a program P has been introduced in [14]. This means that the two programs do not interfere with each other in the composition. They give an efficient algorithm for deciding whether a program Q fits after P , and show that if every program in the composition fits after the previous one, then every program is communication closed. If Q does not fit after P , they introduce so-called separators that allow safe composition of P and Q, and describe their construction.

In [15] they introduce a notion of *sealing*. We say that a program Q seals a program P if neither Q nor any other following program can interfere with P. Usually, if Q seals P, then Q is composed of smaller programs among which each fits after the previous one. Efficient algorithms are given for (i) testing whether a program seals another one, and (ii) constructing seals of a class of programs. If in a composition every program seals the previous one, then every program is communication closed and therefore safety of the composition follows from analysis of individual programs.

c) *Probabilistic Systems:* A probabilistic version of communication-closed layers was introduced in [43] and later extended to abstract probabilistic automata in [41]. They show that executions of probabilistic automata can be reduced to ones structured in layers. By taking a specific randomized mutual exclusion algorithm as example, they show that signifi-cantly fewer locations have to be considered in the verification process, which speeds up probabilistic model checking.

C. Communication-closure as design principle

As discussed in Section IV-A, communication-closed layers consitute a link between the sequential core of a distributed algorithm, and the phenomena induced by concurrency and asynchrony. In the following approaches it has been used to design distributed systems.

a) Algebraic approach: Following the original work by Elrad and Francez [13] the concept of layering has been studied in the context of process calculi [39], [26], [19]. Fokkinga, Poel, and Zwiers [19] discuss completeness of the communication-closed layers law with respect to the layering operator $\bullet$, where $P \bullet Q$ can be described as parallel compo-sition of P and Q with the restriction that if action q of Q depends on an action p of P , then p must precede q. If all actions of Q depend on all actions of P, the layer operator degenerates to sequential composition $P\ ;\ Q$. Using algebraic methods, parallel programs are derived from sequential ones in [39] similar to the approach in [10]. The theory is extended to timed systems in [26]

b) Epistemic logic: Behavior of a distributed algorithm can be described using epistemic logic. Changes in systems, that is, transitions, correspond to changes in knowledge of processes. In [25] we see how every knowledge modification of a set of processes can be implemented as a layer. These layers are then composed using so-called layer composition. This means that if two actions belong to different layers and if the actions interfere with each other, then they have to be executed in the corresponding order of the layers.

If a specification is given in epistemic logic, and can be split into simpler knowledge transitions, then we can obtain layers for each of the transitions. Layer composition of these layers represents the implementation satisfying the behavior from the specification. Moreover, these layers are formulated carefully such that their layer composition can be transformed to a distributed implementation, using communication closed laws.

D. Distributed Algorithms in the Heard-of Model

The HO-Model [9] was originally introduced as a computational model, that is, to model and prove correct fault-tolerant consensus algorithms for benign faults. Later it has been generalized to value faults [2] (e.g., Byzantine). The central idea is to capture round-based distributed algorithms that basically evolve like lock-step synchronous systems; while faults and time-outs are modeled by messages not being received. In this way the central concept is the *HO* set, where $HO(p, r)$ contains the processes from which process p has heard of — has received messages from — in round r. While the original work was in the context of distributed algorithms theory, at the latest with the introduction of a reduction theorem [6] it showed its potential as verification framework for system implementations. The observation in [6] is that in asynchronous executions of HO Algorithms, order of message arrival does not influence the local state transition. For these round-based distributed algorithms it is only relevant which message for the current round have been received by the time a process does the computation step of that round. Thus, on top of a reduction similar to [13] (that brings together all events of a round), all send events can be merged into a global send event, and receive events into a global receive event, and all computation events into a global computation event.

This allows us to analyze sequential global executions where the non-determinism due to faults, timing, etc. is captured in the non-determinism in the *HO* sets of received messages. Such systems can be effectively verified with different methods. In the literature, we already find results using the following methods: a domain-specific consensus logic with decision procedures [11], [12], and methods to infer invariants [47], cut-off results [35], for model checking abstraction-based model checking [1], bounded model checking [45], [44], interactive theorem provers and finite state model checking [7], [8].

5. DESIGN FOR VERIFICATION

As discussed in Section IV-D, there is a selection of effective verification approaches using communication closed layers and the HO Model. In principle this can be exploited in two ways. First, the distributed system should be imple-mented using domain-specific languages [12], [3] that provide sufficient structure for verification. Second, an asynchronous implementation of a distributed system should be given as input to the verification process. For fault-tolerant distributed systems it is quite likely that (at least large parts of the asynchronous code) are communication closed. For instance, it is known [17] that to ensure reliable communication on top of lossy links, one needs to encode information equivalent to a unique message tag (in order to know whether the message is stale or should still be considered). As fault-tolerant distributed systems solve some coordination problems (e.g., atomic broadcast) that have reliable communication as sub-problem, these systems thus have unique message tags, which are in turn a strong indicator for communication-closed rounds. For instance, a message tag can be understood as the layer number of the layer it belongs to.

Typically, parts of the code that take care of recovering a crashed process are not communication-closed. We suggest that verification of such functionalities should be done inde-pendently from verification of the normal operation.

A. Design in Domain-specific Language

Psync is a domain-specific language [12] for expressing consensus algorithms in the HO model. The code of this language serves as input to (i) verification and (ii) compilation into an (asynchronous) runtime. Arguments similar to the reductions from [13], [6] ensure that the results of the verification apply to the asynchronous code. While the code is verified to be always safe, for liveness, the compiled code has to implement specific communication predicates. Adapting implementations

of communication predicates [24] to different networks, and measuring and improving the performance of these implementations are interesting research challenges.

B. Asynchronous Code

In general, asynchronous code for distributed systems needs not be communication closed. As mentioned above, for instance, the result of [17] suggests that if the code implements a fault-tolerant service (such as, state machine replication), it is quite likely that the code is communication closed. Automatically detecting whether this is the case, is a research challenge. However, with suitable user-provided code annotations, we conjecture that (i) it can be checked effectively, and (ii) the asynchronous code can be automatically translated into the HO framework, and verified with the methods discussed in Section IV-D. From this verification, we can then infer the communication predicates required for liveness. These predicates are implemented in asynchronous code (on top of networks that provide suitable timing guarantees). Verifying that the asynchronous code implements the required communication predicates in realistic settings, is also a challenge.

6. CONCLUSIONS

We discussed the concept communication-closed layers that has been originally introduced in the CSP framework by Elrad and Francez. They already observed that (i) it can be used for formal program verification as well as systematic construction of programs and (ii) their approach is not limited to CSP. Indeed, it applies to many distributed systems that are characterized by partial orders. For instance, Chou and Gafni [10] introduced stratified decomposition for message passing systems. While these concepts have been studied in the literature, many related questions are still research challenges:

- In Section IV-B we discussed several proof techniques for communication-closed layers. Some of them already come with efficient decision procedures. It is interesting to implement these techniques in verification tools and evaluate how well they perform on realistic benchmarks.
- The design principles from Section IV-C could give good guidelines how to design future systems. Moreover, they come with an interesting theory which may include ideas and concepts that can be used to design verification procedures.
- The Heard-of Model discussed in Section IV-D allows us to analyze fault-tolerant distributed algorithms with communication-closed layers. Extending this reasoning from theoretical consensus algorithms to practical distributed systems is a challenge, which can have huge impact on the correctness of critical computer systems.

In the context of the Heard-of Model, we discussed in Section V our ideas for design and verification of distributed systems. From these ideas we obtain the following immediate questions:

- How to effectively provide the round structure of the Heard-of Model along with the required communication predicates in large distributed systems?
- Given asynchronous code, how to check whether it is communication closed?

Addressing these topics is crucial in order to exploit the important concept of communication-closed layers in practice. We conjecture that it will lead to powerful tools for automated verification of state-of-the-art distributed systems.

REFERENCES

[1] Benjamin Aminof, Sasha Rubin, Ilina Stoilkovska, Josef Widder, and Florian Zuleger. Parameterized model checking of synchronous dis-tributed algorithms by abstraction. In *VMCAI*, pages 1–24, 2018.

[2] Martin Biely, Bernadette Charron-Bost, Antoine Gaillard, Martin Hutle, Andr´e Schiper, and Josef Widder. Tolerating corrupted communication. In *PODC*, pages 244–253, 2007.

[3] Martin Biely, Pamela Delgado, Zarko Milosevic, and Andr´e Schiper. Distal: A framework for implementing fault-tolerant distributed algo-rithms. In *DSN*, pages 1–8, 2013.

[4] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of Blockchains. Master's thesis, University of Guelph, 2016.

[5] Mike Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI*, Berkeley, CA, USA, 2006. USENIX Association.

[6] Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In *RP*, volume 5797 of LNCS, pages 93–106, 2009.

[7] Bernadette Charron-Bost, Henri Debrat, and Stephan Merz. Formal verification of consensus algorithms tolerating malicious faults. In *SSS*, pages 120–134. Springer, 2011.

[8] Bernadette Charron-Bost and Stephan Merz. Formal verification of a consensus algorithm in the heard-of model. *Int. J. Software and Informatics,* 3(2-3):273–303, 2009.

[9] Bernadette Charron-Bost and Andre´ Schiper. The heard-of model: com-puting in distributed systems with benign faults. *Distributed Computing,* 22(1):49–71, 2009.

[10] Ching-Tsun Chou and Eli Gafni. Understanding and verifying distributed algorithms using stratified decomposition. In *PODC*, pages 44–65, 1988.

[11] Cezara Drăgoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A logic-based framework for verifying consensus algorithms. In Kenneth L. McMillan and Xavier Rival, editors, *VMCAI*, pages 161–181. Springer, 2014.

[12] Cezara Drăgoi, Thomas A. Henzinger, and Damien Zufferey. Psync: a partially synchronous language for fault-tolerant distributed algorithms. In *POPL*, pages 400–415, 2016.

[13] Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Sci. Comput. Program.*, 2(3):155–173, 1982.

[14] Kai Engelhardt and Yoram Moses. Safe composition of distributed programs communicating over order-preserving imperfect channels. In *IWDC*, pages 32–44, 2005.

[15] Kai Engelhardt and Yoram Moses. Causing communication closure: safe program composition with reliable non-fifo channels. *Distributed Computing*, 22(2):73–91, 2009.

[16] Andy Extance. The future of cryptocurrencies: Bitcoin and beyond. *Nature*, 526, 2015. http://dx.doi.org/10.1038/526021a.

[17] Alan Fekete and Nancy A. Lynch. The need for headers: An impossi-bility result for communication over unreliable channels. In *CONCUR*, pages 199–215, 1990.

[18] Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[19] Maarten Fokkinga, Mannes Poel, and Job Zwiers. Modular completeness for communication closed layers. In *CONCUR*, pages 50–65, 1993.

[20] Rob Gerth and Liuba Shrira. On proving communication closedness of distributed layers. In *FSTTCS*, pages 330–343, 1986.

[21] Patrice Godefroid. Using partial orders to improve automatic verification methods. In CAV, volume 531 of *LNCS*, pages 176–185, 1990. GOS.

[22] Distributed ledger technology: beyond block chain. A report by the UK Government Chief Scientific Adviser. GS/16/1, 2016. https://www.gov.uk/government/publications/distributed-ledger-technology-blackett-review.

[23] C. A. R. Hoare. Communicating sequential processes. *Commun. ACM*, 21(8):666–677, 1978.

[24] Martin Hutle and Andre´ Schiper. Communication predicates: A high-level abstraction for coping with transient and dynamic faults. In *DSN*, pages 92–101, 2007.

[25] Wil Janssen. Layers as knowledge transitions in the design of distributed systems. In *TACAS*, pages 238–263, 1995.

[26] Wil Janssen, Mannes Poel, Job Zwiers, and Qiwen Xu. Layering of real-time distributed processes. In *FTRTFT*, pages 393–417, 1994.

[27] Igor Konnov, Marijana Lazi´c, Helmut Veith, and Josef Widder. Para2: Parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms. *Formal Methods in System Design*, 51(2):270–307, 2017.

[28] Igor V. Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *POPL*, pages 719–734, 2017.

[29] Hermann Kopetz and G¨unter Gr¨unsteidl. TTP – a protocol for fault-tolerant real-time systems. *IEEE Computer*, 27(1):14–23, 1994.

[30] Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. *ACM Trans. Comput. Syst.*, 27(4):7:1–7:39, 2009.

[31] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM,* 21(7):558–565, 1978.

[32] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[33] Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Commun. ACM*, 18(12):717–721, 1975.

[34] Nancy Lynch. *Distributed Algorithms.* Morgan Kaufman, 1996.

[35] Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff Bounds for Consensus Algorithms. In *CAV*, pages 217–237, 2017.

[36] Iulian Moraru, David G. Andersen, and Michael Kaminsky. There is more consensus in egalitarian parliaments. In *SOSP*, pages 358–372, 2013. http://doi.acm.org/10.1145/2517349.2517350.

[37] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. https://bitcoin.org/bitcoin.pdf.

[38] Doron Peled. All from one, one for all: on model checking using representatives. In *CAV*, volume 697 of *LNCS*, pages 409–423, 1993.

[39] Mannes Poel and Job Zwiers. Layering techniques for development of parallel systems. In *CAV*, pages 16–29, 1993.

[40] Nicola Santoro and Peter Widmayer. Time is not a healer. In *STACS*, volume 349 of *LNCS*, pages 304–313, 1989.

[41] Arpit Sharma and Joost-Pieter Katoen. Layered reduction for abstract probabilistic automata. In *ACSD*, pages 21–31, 2014.

[42] Frank A. Stomp and Willem P. de Roever. A principle for sequential reasoning about distributed algorithms. *Formal Asp. Comput.*, 6(6):716–737, 1994.

[43] Mani Swaminathan, Joost-Pieter Katoen, and Ernst-R¨udiger Olderog. Layered reasoning for randomized distributed algorithms. *Formal Asp. Comput.,* 24(4-6):477–496, 2012.

[44] Tatsuhiro Tsuchiya and Andre´ Schiper. Using bounded model checking to verify consensus algorithms. In *Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings,* pages 466–480, 2008.

[45] Tatsuhiro Tsuchiya and Andre´ Schiper. Verification of consensus algorithms using satisfiability solving. *Distributed Computing,* 23(5-6):341–358, 2011.

[46] Antti Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets 1990*, volume 483 of *LNCS*, pages 491–515. Springer, 1991.

[47] Klaus von Gleissenthall, Nikolaj Bjørner, and Andrey Rybalchenko. Cardinalities and Universal Quantifiers for Verifying Parameterized Systems. In *PLDI*, pages 599–613, 2016.

[48] Apache ZooKeeper. Web page. http://zookeeper.apache.org/.