# Compilation schemes for high-level fault-tolerant distributed protocols

## 1 General information

- Advisor: Cezara Drăgoi
- Institution: Inria Paris, Antique Team
- Location: ENS, 45 rue d'Ulm, 75005 Paris
- Language: French or English
- Keywords: fault-tolerant implementations, programming abstractions, compilation schemes, state machine replication.

**Summary**

Fault-tolerant distributed data structures are at the core distributed systems. Due to the multiple sources of non-determinism, their development is challenging. The project aims to increase the confidence we have in distributed implementations of data structures. We think that the difficulty does not only come from the algorithms but from the way we think about distributed systems. We will investigate partially synchronous programming abstractions that reduce the number of interleavings, simplifying the reasoning about distributed systems and their proof arguments. We will use partial synchrony to define reduction theorems from asynchronous semantics to partially synchronous ones, enabling the transfer of proofs from the synchronous world to the asynchronous one. Moreover, we will define a domain specific language, that allows the programmer to focus on the algorithm task, it compiles into efficient asynchronous code, and it is equipped with automated verification engines. The internship is financed by the ANR project SAFTA - Static analysis of fault-tolerant algorithms.

## 2 Context

Highly available data storage systems and high processing power systems are available today due the massive development of fault-tolerant distributed systems. The standard way to make any applications fault-tolerant (available independently of network and hardware faults) is replication: the application is copied on different sites and all its clients are free to interact with any of the replicas deploying the application. The challenge posed by replication is to keep the replicas consistent, i.e., the application is in the same state at all replicas, despite any interaction with

the client. Notorious examples are Amazon Dynamo (highly-available key-value storage system), Apache Zookeeper (distributed hierarchical key-value store used to provide a configuration service and synchronization service), or the blockchain, the data structure bitcoin is built on top of.

The complexity of the design of distributed algorithms is rooted in the FLP theorem [6] which states that in asynchronous networks (there are no bounds on the message delay), in the presence of faults (messages are lost or processes crash) it impossible for all processes to agree on a value, i.e., to solve consensus. State machine replication can be roughly implemented by using consensus iteratively. To cope with impossibility results, existing algorithms make various assumptions on the network (e.g., bounding the number of messages that are dropped, or the number of processes that may crash) or on the provided consistency guarantees (e.g., processes might be allowed to agree on two different values instead of one). This leads to solutions with extremely complex flow of data, that simultaneously deal with the asynchronous nature of the network, the presence of byzantine (message corruption) or benign (message lost) faults, and updates of the processes local state.

Given the massive usage and the intricacy of distributed system, they are a prime candidate for automate verification. From a theoretical perspective, these are infinite state systems, communicating via unbounded channels, whose verification is in general undecidable. However, the state of the art shows that static analysis techniques like deductive verification and abstract interpretation, had a great impact on increasing correctness of sequential software, despite the fact that the verification of sequential programs is in general undecidable. These are incomplete verification tools, that either prove that the program satisfies the specification or they launch alarms, that is a collection of potential bugs in the program. We can cite the impact the Satisfiability Modulo Theory (SMT) solver Z3 [9] had for the verification of device drivers, or the static Analyzer Astrèe which was successfully applied to numeric embedded software. However, there are no automated verification tools to accompany the design of fault-tolerant protocols.

# 3 Objective

The goal of this project is to increase the confidence we have in replicated systems. We think that the difficulty does not only come from the algorithms but from the way we think about distributed systems. Therefore we propose to:

- **identify high-level programming abstractions** that focus on algorithmic aspects simplifying the reasoning about fault-tolerant implementations,
- build a **domain-specific language** based on the identified programming abstraction, that allows the programmer to focus on the algorithmic task, it is equipped with an automated verification engine, and it compiles into efficient asynchronous code.

Due to the complexity distributed systems have reached, and their continuous evolution, we believe it is no longer neither realistic nor efficient to assume that high level specifications can be proved when development and verification are two disconnected steps in the software production process.

One fundamental obstacle in achieving proved correctness of distributed systems is the lack of abstractions when reasoning about their behaviors. The standard programming paradigm for implementing fault-tolerant distributed algorithms is highly error-prone, providing only asynchronous communication primitives and timer constraints. The programmer has to reason separately about *the degree of synchrony* of the network and *the various types of faults*, e.g., process

crash, message loss, network partition. Therefore we aim to identify *programming abstractions* that simplify the representation of program behaviors, leading to simpler proof arguments, without sacrifying the explosiveness power.

We focus on programming abstraction that distinguish the high-level algorithmic task from the network assumptions. In order to reconcile the modeling of various network assumptions, the distributed algorithms community introduced **partially synchronous computational models** that unify reasoning about (a)synchrony and fault assumptions, separating it from the algorithmic task. We start out quest for programming abstraction from two such partially synchronous models *Round-by-round fault detectors* [7] by E. Gafny and *Heard-Of* [4] by B. Charron-Bost and A. Schiper.

The partially synchronous programming abstraction simplifies the proofs arguments of fault-tolerant distributed algorithms. Algorithms are structured in rounds executed in lock-step and the round number plays the role of an abstract clock. The lock-step semantics reduces the number of interleavings between actions performed by different processes, leading to simpler invariants and variant functions that describe the system only at the boundary between rounds.

We will define efficient compilation schemes from partial-synchronous abstraction to asynchronous code. To this, the most important step is the procedure that implements the round switch: this procedure decides when a process goes to the next round while allowing sufficiently many messages to be delivered. We plan to investigate algorithms for determining the round switch that (1) preserves the safety assumptions of the algorithm (if any) and (2) checks (under-approximations of) the network assumptions that ensure progress. In [5] we have defined a compilation technique parametrized by timeouts.

Synchronization services like Apache Zookeeper [1] or Chubby [2] (developed by Google) are based on the agreement algorithms whose modelization we directly target in our benchmarks, e.g., Zab [8], Viewstamped [10], Multi-Paxos [3].

## References


[1] Apache zookeeper.

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

[3] 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, pages 398–407, New York, NY, USA, 2007. ACM.

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

[5] 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*, pages 400–415, 2016.

[6] 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.

[7] Eli Gafni. Round-by-round fault detectors (extended abstract): unifying synchrony and asynchrony. In *PODC*, pages 143–152, 1998.

[8] 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*, pages 245–256. IEEE, 2011.

[9] Leonardo Moura and Nikolaj Bjorner. Z3: An efficient SMT solver, 2008.

[10] Brian M. Oki and Barbara Liskov. Viewstamped replication: A general primary copy. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, August 15-17, 1988*, pages 8–17, 1988.