Synchronizability for distributed systems

Context. Most of the distributed systems we use nowadays are based on the message-passing paradigm where systems are structured into parties that interact only by sending/receiving messages. Message-passing programming is largely employed in high performance computing (MPI, OpenMP, etc), event-driven applications built on top of actor-based languages (Scala, Erlang, etc), service-oriented architectures, peer-to-peer applications, etc. Unfortunately, because of the variety of communication models (peer to peer, mailbox, etc.), of the ambiguities of the specifications of the communication primitives, and of the difficulty of running representative tests, etc., it is error prone and therefore often reserved to experts.

The majority of the issues above stem from the asynchronous nature of messages exchange. Indeed, the conception and the analysis of message-passing programs is greatly simplified if one can assume that all communications occur synchronously, i.e., that each message is received exactly at the same time it has been sent. As a matter of fact, some properties (e.g., reachability) becomes decidable if the underlying communication model is synchronous. We are thus interested in understanding under which conditions we could avoid using asynchronous models and being able to design and prove correct simpler but equally powerful systems. This is a property that goes under the name of synchronizability.

Synchronizability of a distributed system is the property that the observable behaviour of the system is the same whether or not the communications are synchronous. There are several notions in literature of synchronizability (sometimes also referred to as slack elasticity): for hardware design [1], with the goal of ensuring that some code transformations are semantic-preserving, for parallel programming in MPI [2, 3], for ensuring the absence of deadlocks and other bugs, for existentially bounded communicating machines [4], or more recently for web services and choreographies [5, 6], for verifying various properties, among which choreography realizability [7]. An algorithm for checking whether a distributed system is synchronizable has been proposed; it is now implemented in some verification tools like Verchon [10]. This algorithm has been applied to various communication models, including mailbox communications and peer-to-peer communications.

Recently, Finkel and Lozes observed that this algorithm contains an important flaw [8], and they proved that this flaw cannot be fixed for peer-to-peer communications, because the problem is undecidable in general. They also observed that the algorithm and its justification are correct under the extra assumption that the topology of communication channels is restricted to a ring.

Research project. The main topic of this PhD is to develop methods and tools that help designing safe distributed systems based on the notion of synchronizability.

- First, it will be explored how to effectively check for synchronizability. The PhD candidate will have to identify new subclasses of systems based on restrictions on the communication topology or on the model of communications, where synchronizability
checking becomes decidable. He will also have to look for semi-decision algorithms that may not always succeed to prove synchronizability but provide the correct answer when they terminate. Finally, we would like her/him to exploit these results with existing model-checking tools for checking synchronizability.

- Second, we want to ensure synchronizability by design. To this aim, the PhD candidate will have to understand whether behavioural type systems, like multiparty session types [9], ensure or not synchronizability. If an effective type system can be built, we would like to determine what is the relation (if any) with the subclass of systems identified in the previous point. In addition, we want to study to which extent synchronizability is linked to realisability and protocol conformance.

- Finally, the PhD candidate will have to exploit synchronizability in system verification. We would like to explore ways for *automatically repairing* synchronizability [10] in systems that are not synchronous. Moreover we would study how the synchronizability of a system affects the decidability of properties such as reachability, absence of deadlock, absence of orphan messages, etc.

*References*


