The Computability Hierarchy of Broadcast Abstractions

Matthieu Perrin
matthieu.perrin@univ-nantes.fr

Achour Mostéfaoui
achour.mostefaoui@univ-nantes.fr

Keywords: *Abstractions, Algorithms, Broadcast, Computability, Distributed computing, Shared Data Structures, Fault Tolerance, Reliability*

Context

Distributed systems are composed of a set of independant entities, be it decentralized networks of independant computing machines, multi-threads applications executed within a single multi-core machine, supercomputers composed of clusters of GPUs, or even complex human societies. They have recently been increaslingly present in our lives: In 2023, the world is now exceeding 5.3 billion Internet users, the simplest cellphone processor today is multi-core, cloud computing is now a well-established technology used by countless companies thanks to the explosion of services it offers, geo-replication is largely used to guarantee the persistence of sensitive data, etc.

Despite all the possible applications for distributed systems, distributed applications are notoriously difficult to program, on the one hand because each distributed system has its own specificities which make the algorithms dependent on the system considered, and on the other hand because it is currently very hard to dissociate the functional aspects specific to the application under development from those relating to distributed systems-specific synchronization issues.

Shared data structures are central to distributed applications as they encapsulate shared data and abstract synchronization mechanisms built upon communication primitives available to processes, such as shared memory or communication channels. Suppose, for example, that one needs to implement a new e-commerce service. A central data structure of the project would be a set of sellable objects, in which objects can be inserted when they are put up for sale, and deleted after they have been sold. From an functionnal perspective, the data structure implementation needs to focus on making the use of such data structure user-friendly, for example by providing a nice query interface, allowing to easily filter out elements based on user's previous purchases, or to optimize prices using some state-of-the-art machine learning.

However, in practice, a large portion of the code of the data structure will have to focus on low-level synchronization mechanisms required to make it safe to use in a distributed environment.

For example, when an item is sold at one data center, it needs to become invisible immediately at other data centers despite asynchrony and crashes, so a specific synchronization algorithm must be implemented to maintain consistency (this is similar to implemented a boolean atomic register to encode whether a specific object is available or not) that requires sending messages and waiting for acknowledgements, possibly twice [2]. Differently, evaluating a read query on a consistent state of the set would require an additional non-intuitive snapshot algorithms based on double collect strategies and complex helping mechanisms [1]... This methodology often results in very complicated algorithms that can only be designed by experts on several fields of distributed computing, and that

are very hard to maintain when the needs of the application evolve. Even worst, the synchronization mechanisms described above only work if the strong assumptions made in [2] are relevant in the system on which the e-commerce shop is built. Otherwise, for example if more types of failures are prone to happen, if the processes are unable to communicate directly with each other (e.g. in peer-to-peer systems), or if the scale of the system makes waiting for quorums impractical, completely different algorithms must be designed to solve the same problem.

Broadcast abstractions

When processes communicate by sending and receiving messages, replication is used to avoid a loss of information when crashes occur. A major difficulty resulting from replication is to maintain consistent data across all replicas when updates are performed concurrently. Reliable broadcast ensures all non-faulty processes receive exactly the same set of messages. It can be used to make sure all replicas will be noticed when a process performs an operation. However, reliable broadcast does not provide any form of ordering guaranties on the messages, so additional synchronization is required to maintain a consistent state across all replicas.

Reliable broadcast can be enriched by restricting the possible orders of message reception. For example, causal broadcast ensures that, if a process p first receives a message m and then broadcasts a message m', then no process receives m' before m. Causal broadcast can be used to solve some consistency issues [13], but not all. Another example is total-order broadcast, that ensures all replicas will receive all messages in the same order. Total-order broadcast allows to solve all the synchronization issues stated above, but at a much higher cost than causal consistency. The implementation of both reliable broadcast, causal broadcast and atomic broadcast has been widely studied in different systems [3, 4, 8, 12].

Total-order broadcast and causal broadcast are are not equivalent, because implementing total-order broadcast on top of causal broadcast requires strong assumptions on failure patterns and message routing delays, that may, or may not, be realistic depending on the specificities of the considered system. Similarly, total-order broadcast may be more costly than what is required for the e-commerce service described above (or any other shared system), or may provide guarantees that are only required infrequently.

We recently discovered several abstractions intermediate between reliable broadcast and atomic broadcast, but equivalent to various central problems of distributed algorithms, which allow very simple, elegant and efficient implementations of a variety of shared data structures. For example, mutual-broadcast [6] captures reading and writing in a shared variable, SCD-broadcast [9] is equivalent to the snapshot object defined above, and pair-broadcast [6] is exactly the abstraction requires to make sure that one, and only one, customer can buy the last item from the stock, when more that one try to do so concurrently.

This convinced us that broadcast primitives are "the right abstractions" to build a "standard model" of synchronization problems, that could tackle both challenges of allowing simple implementations of shared data structures that non-expert engineers can use to build efficient complex systems, as well as providing the fundational background to unify specific distributed systems. In other words, broadcast primitives can serve as building blocks that allow processes to synchronize their local copies of shared data (replicas), while providing a spark of homogeneity between systems.

Objectives

The recrutee will be required to perform advanced research, with the intention to publish about the following question: *Which distributed problems can be abstracted with broadcast abstractions, and how?* This general problem can be divided into many simpler ones, such as:

- Given a specific synchronization problem, (e.g. consensus between k processes or k-set agreement), is there a broadcast abstraction that captures it?
- Are there synchronization problems whose complexity cannot be captured exactly by a broadcast abstraction? How to prove it?
- How does the hierarchy of broadcast abstractions compare to other hierarchies of distributed computing, such as the minimal failure detector of consensus number?
- How to implement the newly-found abstractions efficiently in various distributed computing models?

After a phase of familiarization with the topic and the problems it raises, the recruitee will have to choose open questions and try to answer them, either by proposing an algorithm or by demonstrating a theorem of impossibility. This position requires advanced abstract reasoning skills, and advanced skills in distributed computing, especially regarding the theory of distributed computability.

It will take place in the *Laboratoire des Sciences du Numérique de Nantes* (LS2N) of the University of Nantes. The GDD (Gestion des Données Distribuées – Distributed Data Management) team is internationally recognized in the fields of distributed and parallel computing, the Web and databases.

Do not hesitate to contact us for more information!

References

[1] Afek, Y. Hagit A. Dolev D. *et al. Atomic snapshots of shared memory.* Journal of the ACM, 1993.

[2] Attiya H., Bar-Noy A. and Dolev D. *Sharing memory robustly in message-passing systems.* Journal of the ACM, 1995.

[3] Birman K. P. and Joseph, T. A. *Reliable communication in the presence of failures.* TOCS, 1987.

[4] Birman K. P., Hayden M., Ozkasap O *et al. Bimodal multicast.* TOCS, 1999.

[5] Demers, A., Greene, D., Houser *et al. Epidemic algorithms for replicated database maintenance.* ACM SIGOPS Operating Systems Review, 1988.

[6] Déprés M., Mostéfaoui A., Perrin M. and Raynal M. *Send/Receive Patterns Versus Read/Write Patterns in Crash-Prone Asynchronous Distributed Systems.* DISC, 2023.

[7] Gilbert S. and Lynch N. *Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services.* ACM SIGACT News, 2002.

[8] Hadzilacos V. and Toueg S. *Reliable Broadcast and Related Problems.* Journal of Distributed Systems, 2013.

[9] Imbs, D., Mostefaoui, A., Perrin, M. and Raynal, M. *Set-constrained delivery broadcast: Definition, abstraction power, and computability limits.* ICDCS, 2018.

[10] Kleinberg, J. *The small-world phenomenon: An algorithmic perspective.* STOC, 2000.

[11] Lamport, L. *The part-time parliament.* TOCS, 1998.

[12] Nédelec, B. Molli P. and Mostefaoui A. *Breaking the Scalability Barrier of Causal Broadcast for Large and Dynamic Systems.* SRDS, 2018.

[13] Perrin, M., Mostefaoui, A. and Jard, C. *Causal consistency: beyond memory.* PPOPP, 2016.

[14] Voulgaris, S., Gavidia, D. and Van Steen, M. *Cyclon: Inexpensive membership management for unstructured p2p overlays.* Journal of Network and systems Management, 2005.