Building TMR-Based Reliable Servers Despite Bounded Input Lifetimes

Paul Ezhilchelvan[1], Jean-Michel Hélary[2], and Michel Raynal[2]

[1] Dept of Computing Science, University of Newcastle, NE1 7RU, UK
paul.ezhilchelvan@newcastle.ac.uk
[2] IRISA
Campus de Beaulieu, 35042 Rennes Cedex, France
{helary,raynal}@irisa.fr

1 Context

This paper comes from practical considerations [5]. It considers a client-server system made up of an arbitrary large number of clients that access a single server. To access the server, a client issues an input request and possibly waits for an answer. Then, the server processes the client input and sends back the result (if any) to the client.

The system is unreliable in the following way: clients can fail by crashing. This means a client behaves correctly until it possibly crashes. When a client fails, it stops its execution and never recovers. The implementation of the server relies on a TMR approach (Triple Modular Redundancy): it is made up of three server processes (in short *processes*). Each process processes every client input. Among the three processes, at most one of them can be faulty. Moreover, it can fail by exhibiting an arbitrary behavior. It is assumed that each process can sign the messages it sends, and authenticate the messages it receives from the other processes.

The system is synchronous in the following way. Local computations consume no time, and all communication delays are bounded. More precisely, D (resp. d) is an upper bound for the communication delays between any pair made up of a client and a correct process (resp. any pair of correct processes). Due to the geographic dispersion of clients and the locality of the processes implementing the server, we have $d << D$. Moreover, the clocks of the three processes implementing the TMR subsystem are assumed to be perfectly synchronized [6].

2 The Problem

The problem we address is the implementation of a correct TMR-based server. To this end, the correct processes (at least two) implementing the server must have mutually consistent copies of the state of the service they implement. This is obtained by requiring correct processes to order the client inputs in the same way before processing them according to this order [4]. This ordering is typically achieved by Atomic Broadcast protocols [1,3]. Moreover, every input has to be verified before processing, which involves accessing client-related data (such as authentication keys, input history, client privileges, credit limits, etc.). Owing to a large number of potential clients, this client-related data will be large and have to be stored in a disk. A process receiving input directly from a client must access this disk, and this can be time-consuming. A process receiving an input forwarded by a replica should also verify the input, since the forwarding process may be faulty: it may not have verified the input properly or, even worse, may be replaying an old input as a newly received one. To avoid other disk accesses, our approach verifies forwarded inputs through matching: the process waits for the ordered delivery of another identical input. So, if the forwarded input is replayed/invented or if it has been sent only to the forwarding process due to client crash, this input will remain unmatched but will not cause unwarranted verification. Let us also note that this approach requires a time bound so that waiting for the delivery of a matching input can be terminated even if no matching input occurs. Therefore, we introduce a duration bound (denoted Σ) as an essential requirement: every input (either directly received from a client or forwarded) is systematically discarded from a server process address space within at most Σ time units after it has been received. Note that, without this requirement, server processes should have an infinite memory: in fact, a faulty server process could invent infinitely many inputs and forward them to its peers. Such inputs would never match with another input and thus, without a time requirement, would never be discarded.

The novelty here comes from the fact that Atomic Broadcast has to be ensured despite the combined effects of the bounded lifetime Σ, the Byzantine server process, and the client crash failures. These constraints actually define a new problem that we call *Σ-constrained Atomic Broadcast*. A protocol solving the *Σ-constrained Atomic Broadcast* problem is called *Σ-ordering protocol*.

3 Results

The full paper [2] focuses on the family of Σ-ordering protocols. It presents two original results. First, a Σ-ordering protocol is presented; this protocol assumes $\Sigma > D + 3d$. Second, it is shown that it is impossible to design a Σ-ordering protocol when $\Sigma < D$.

3.1 Assumptions on Σ-Ordering Protocols

Let us first remark that, whatever the design of a Σ-ordering protocol, each input issued by a client is received by none, one, two or the three processes of the TMR subsystem (this depends on whether the client crashes or not when it issues the input). Then, the dissemination of copies of a client input among the processes depends on the protocol.

As far as the copies of a client input is concerned, a Σ-ordering protocol has the following three characteristics:

– A Σ-ordering protocol may force a process receiving an input (either directly from the client or from another process) to forward this input to none, one or the two other processes. This is the *forwarding policy* associated with the protocol. As a consequence of (1) this policy, (2) the client correctness, and (3) the correctness of the forwarding process, a process may have zero, one or more copies of an input, at the same time.
– When a Σ-ordering protocol requires a process to forward the client input it has received, it can permit some delay α (with $\alpha \leq \Sigma$) to elapse between the reception of an input and its forwarding. This is the *delay policy* of the protocol.
– A last feature of a Σ-ordering protocol lies in the number of input copies that a process must simultaneously have for ordering that input. This is the *decision policy* of the protocol.

3.2 Sketch of the Protocol

The proposed protocol uses an underlying Timed Atomic Broadcast protocol [1,3] (the timed atomic broadcast protocol described in [1] can be implemented in the TMR subsystem defined previously). Let TA-broadcast and TA-deliver be the two primitives defined by such a protocol. They ensure that all messages that are TA-broadcast by correct processes are TA-delivered to them. Moreover, the correct processes TA-deliver the same set of messages and these TA-deliveries occur in the same order. Finally, they satisfy the following timeliness property: a message that is TA-broadcast at time t is TA-delivered by every correct process before time $t + \Delta$. In the context of the TMR subsystem (with three processes, at most one faulty (Byzantine) process, message signatures, perfectly synchronized clocks, and an upper bound d on message transfer delays), we have $\Delta = 2d$.

The proposed Σ-ordering protocol is designed according to the following principles. When a server process p_i receives a *valid* input[1] directly from a client, it makes up a signed message m including this input, the current value of the clock and its identity. Then, it TA-broadcasts this message (forwarding policy) and does it immediately (delay policy $\alpha = 0$). A correct process orders a client input as soon as it has TA-delivered two messages including this input (decision policy). Note that, if a client is correct, each input μ of that client is received exactly once by each process. Since there are at least two correct processes, there will be at least two TA-broadcast invocations (by two different senders) with messages having μ as input field. So, each correct process will TA-deliver at least two messages containing μ as input field.

Due to the "Σ (bounded lifetime) constraint", every input stored in a process local memory is discarded Σ time units after it has been received. So, a crucial point in the design of the protocol is to ensure that, for each input μ from a correct client, at least two messages with an input field equal to μ will *simultaneously* be present in the local memory of at least two correct processes.

[1] i.e., an input verified by accessing the disk where client related data is stored.

This intuitively explains why those messages must not be removed too quickly due to the Σ bound, and why the protocol requires $\Sigma > D + \Delta + d = D + 3d$. The protocol ensures that two TA-broadcasts concerning the same input from a correct client, but issued by two different correct processes, will always have an "overlapping period" in the local memory.

The other important issue in the design of the protocol lies in ensuring that, despite the attempts from the faulty process to replay obsolete client inputs (or play false client inputs), any input ordered by correct processes is a client input and no client input is ordered more than once. The proposed protocol ensures this property by forcing processes to preventively discard messages (even before their lifetime Σ has elapsed) as soon as this discarding operation can be safely performed. The proof shows that no message is discarded too early (any input from a correct client is ordered at least once) or too late (any input is ordered at most once by a correct process). It also shows that the inputs from correct clients are actually ordered, and that the correct processes order the inputs in the same way.

3.3 The Impossibility Result

In the full paper [2], two lemmas are first proved. They show that, for every Σ-ordering protocol, only inputs received directly from clients may be forwarded (Lemma 1), and no process is allowed to order an input unless it simultaneously has at least two copies (from different sources) of this input (Lemma 2). Then, these Lemmas are used to construct a scenario showing no Σ-ordering protocol can be designed when $\Sigma < D$. The following theorem is proved by contradiction.

Theorem 1. *Let Σ be a constant. There is no Σ-ordering protocol if $\Sigma < D$.*

Proofs of the protocol and of the impossibility result can be found in [2].

References

1. Cristian F., Aghili H., Strong H. R. and Dolev D., Atomic Broadcast: From Simple Message Diffusion to Byzantine Agreement. *Proc. 15th Int. Symposium on Fault-Tolerant Computing*, Ann Arbor (MI), IEEE Computer Society Press, pp. 200-206, 1985. 482, 484
2. Ezhilchelvan P., Hélary J.-M. and Raynal M., Building TMR-Based Reliable Servers with Arbitrary Large Number of Clients. *Research Report # 1398*, IRISA-IFSIC, Université de Rennes 1, May 2001. http://www.irisa.fr/EXTERNE/bibli/pi/1398/1398.html. 483, 485
3. Hadzilacos V. and Toueg S., Reliable Broadcast and Related Problems. In *Distributed Systems*, ACM Press (S. Mullender Ed.), New-York, pp. 97-145, 1993. 482, 484
4. Schneider F. B., Implementing Fault-Tolerant Services Using the State Machine Approach: a Tutorial. *ACM Computing Surveys*, 22(4):299-319, 1990. 482
5. Shrivastava S., Ezhilchelvan P., Speirs N. A., Tao S. and Tully A., Principle Features of the *Voltan* Family of Reliable System Architectures for Distributed Systems. *IEEE Transactions on Computers*, 41(5):542-549, 1992. 482

6. Veríssimo P. and Raynal M., Time in Distributed Systems: Models and Algorithms. In *Advances in Distributed Systems*, Springer-Verlag, LNCS 1752, pp. 1-32, 2000. 482