How to avoid the cost of causal communication in large-scale systems[*]

Luís Rodrigues
ler@inesc.pt

Paulo Veríssimo
paulov@inesc.pt

Technical University of Lisboa - IST - INESC [†]

Introduction

In a distributed system, consisting of a collection of processes that communicate by exchanging messages, the order in which messages are delivered to processes is of major relevance to the application design. With the aim of simplifying the design of distributed applications [14,4,11], several algorithms and protocols have been proposed to provide causal order delivery [4,6,9,7,10].

Despite its advantages, the use of causal communication has been somewhat limited by the overhead incurred by existing implementations. We can cite some disadvantages of existing causal communication services [5]: (i) potential large size of "history" information that needs to be stored and exchanged to maintain causality; (ii) little user control over message piggybacking policies; (iii) reliable communication is mandatory to avoid blocking of message delivery.

In the Navigators group at INESC, we are currently studying mechanisms to improve the efficiency of multicast communication over (geographically) large-scale networks. This research is complementary to the joint effort between U. of Bologna and EPFL that aims to provide reliable (virtually synchronous) communication for large scale-systems [2].

We claim that, to allow applications to take advantage of multicast communication, new services (based on weaker assumptions about the system) must be provided. To support this claim, this paper proposes and describes a new quality of service, referred to as *transparent causal messages*.

Related work

Many protocols have been presented to provide causal order delivery. However, despite the diversity of algorithms, most systems provide a single causal multicast communication primitive,

[*]A version of this report will be published in the Proceedings of the 6th SIGOPS European Workshop, Sept., 1994, Dagstuhl, Germany. This work was partially supported by the CEC, through Esprit Project BR 6360 (Broadcast).

[†]Instituto de Engenharia de Sistemas e Computadores, R. Alves Redol, 9 - 6° - 1000 Lisboa - Portugal, Tel.+351-1-3100000.

giving limited flexibility to the application designer. Taking into account the limitations of the early systems, recent research is defining alternative primitives that better match user requirements. Examples of such primitives are the virtually-synchronous communication defined in [13] (that only orders "bags" of messages in respect to group changes) or the global-flush protocol of [1] that orders messages in respect to special *flush* messages. In another report [12], we proposed a primitive that provides the users with explicit control over (message) piggybacking policies.

Transparent causal messages

One of the criticisms made of causal multicast communication systems [5] concerns the mandatory requirement for reliability. Once a message introduces a causal dependency, that message must be reliably delivered; otherwise, succeeding messages will be prevented from being delivered. In some cases the delivery of a causal message is delayed until there is a guarantee that the message will be delivered at all destinations. Additionally, the sender may be prevented from sending new messages before this guarantee is obtained.

Although the above is true, it is also true that one can find a number of distributed applications which take full advantage of causal ordering. Moreover, there is evidence that, when causal delivery properties are necessary, ad-hoc solutions to the problem are usually complex and hard to prove correct [3].

To solve this contradiction, we propose a scheme that distinguishes two types of messages: (normal) *opaque* causal messages and *transparent* causal messages. Transparent causal messages are messages that are delivered in causal order with respect to (normal) opaque messages but that do not introduce causal dependencies. Thus:

- no message is ever delayed by a transparent message;
- no reliability constraints are imposed on the transmission of transparent messages.

The delivery order for transparent messages with regard to opaque messages is summarized in the following table (where opaque messages are represented in capital, transparent messages in lower-case, and right-arrows, $\rightarrow$, represent transitive "happened-before" relation as defined in [8]).

relation	delivery order	relation	delivery order
$M_1 \rightarrow M_2$	M_2 after M_1	$m_1 \rightarrow M_2$	undefined
$M_1 \rightarrow m_2$	m_2 after M_1	$m_1 \rightarrow m_2$	undefined

The implementation of transparent messages is fairly simple and any causal communication protocol can be adapted to provide this service at *almost no cost*. We do not fully present the implementation here but we note that a transparent message only needs to carry the time-stamp of the sender without incrementing its clock.

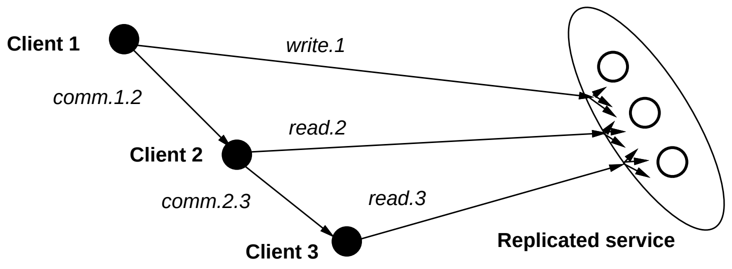

Figure 1: A simple example.

Client-server interactions using transparent messages

In the context of large-scale systems, transparent messages can be extremely useful to implement replicated client-server interactions respecting causal order. The reason is that large-scale systems can be characterized by a number of attributes such as, among others, existence of partitions, non-transitive node-to-node connectivity, and large communication delays. To *always* enforce reliable *multicast* communication in such a setting can be prohibitively expensive. We illustrate the use of transparent messages in this context with the example of figure 1.

Consider a replicated service exporting *read* and *write* operations. Assume that clients interact with the replicated service using causal multicasts. The way replica consistency is maintained is orthogonal to this example. This could be achieved using additional inter-replica messages or by increasing the semantics of the multicasts, for instance requiring all writes to use a totally ordered multicast. Clients can also communicate among themselves (using either point-to-point or multicast communication). To illustrate our point, consider an execution where the following causal relations among messages are observed:

$$\textit{write.1} \rightarrow \textit{comm.1.2} \rightarrow \textit{read.2} \rightarrow \textit{comm.2.3} \rightarrow \textit{read.3}$$

If a single (opaque) causal primitive is offered, not only would all multicasts have to be reliable, but also any communication impairment with *read.2* would delay *read.3*. This is clearly too costly for some applications. Now assume that all reads and replies (not depicted in the figure) are executed using *transparent* messages. Clients 2 and 3 still “see” the update performed by client 1. However, reads can now be implemented using inexpensive best-effort multicasts and reliability ensured by end-to-end replies. Furthermore, communication delays in one read operation would not delay other read operations.

The above example is quite straightforward, but clearly illustrates the power of transparent messages. The example can be further expanded to make all interactions between clients and the replicated server via inexpensive transparent messages and restrict the use of (opaque) causal communication among replicas.

We illustrate this particular use of transparent causal messages with another example (see figure 2). Clients use point-to-point communication to contact a given replica. Replicas of the service communicate among themselves to ensure replica consistency. In this simple example, a

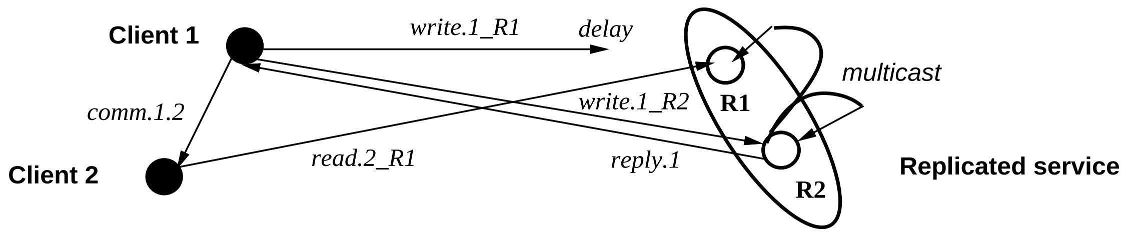

Figure 2: Another example.

replica that is contacted to execute a write operation issues a totally ordered multicast within the replica group before it replies to the client. Replicas keep track of which requests were executed such that if a client contacts more than one replica at-most-once semantics can be guaranteed. As before, causal delivery is globally enforced such that clients have a consistent view of the replicated service state. Consider the following execution:

$$\textit{write.1\_R1} \rightarrow \textit{write.1\_R2} \rightarrow \text{multicast} \rightarrow \textit{reply.1} \rightarrow \textit{comm.1.2} \rightarrow \textit{read.2\_R1}$$

In this execution, client 1 tries to contact replica R1 first. Since the communication link is slow, it tries to contact the other replica (R2). Replica R2 receives the request and issues a totally ordered multicast in the group of replicas. In a large scale system, and because communication delays are not transitive, this multicast can be received at R1 before the first *write.1_R1* message. With an appropriate retry detection mechanism (for instance, see [12]) it is possible to make immediate progress. Clearly, is not desirable that the multicast depends on the first write request, as this would block replica R1 until the request arrives. However, as in the previous example, it is useful that the precedence relation *multicast* → *read.2_R1* is preserved by the communication system. For instance, in the example, when accessing replica R1 client 2 should see the update performed by client 1. This problem can be circumvented by keeping inter-replica multicasts opaque but using transparent messages among clients and between clients and individual replicas.

In large-scale settings, using unreliable point-to-point communication to access remote (potentially replicated) services has many advantages over using reliable multicasts. One of the most important is that clients are not required to keep a fully accurate view of replica membership. This drastically reduces the required synchronization among clients and replicated services. Additionally, less reliable links between the client and the service can be easily tolerated. Reliable group communication can be restricted to inter-replica communication, for instance, by making replicas of a given service members of a virtually-synchronous group [13,12]. However, even when point-to-point messages are used, the propagation of causal precedence can strongly simplify the design of cooperating clients that access a common set of replicated services. Transparent causal messages provide the user with a simple way to express the duality between messages that introduce causal dependencies and messages that just propagate such dependencies. Finally, the definition of transparent causal messages is completely independent of the mechanisms used to implement causality. This permits its use over a large set of platforms.

Conclusions and future work

In the past years, research in the area of reliable communication has focused on algorithms and protocols to efficiently support causal communication. However, the acceptance of these services is far from expected. One reason for this is the relative lack of flexibility of most interfaces. In this paper we claim that new services, that better match application needs, should be sought and provided. However, not all of these services imply new or complex protocols. We illustrated our claim with a simple primitive, incurring low implementation cost, that can be extremely useful to support client-server interaction in large scale-systems.

We have recently developed a remote invocation protocol having large-scale in mind [12]. The protocol, called GRIP, provides flexible support for the construction of replication-transparent remote invocation of replicated services. Unlike the functionality provided by most existing systems, GRIP leaves the semantics of the replication protocol transparent to the remote invocation protocol and provides support for dynamic reconnection and client semantic control; moreover, it introduces explicit support for weakly consistent replication strategies and provides optional per-invocation distributed retry detection. In this protocol, clients use efficient point-to-point communication primitives to access replicated services. The protocol is now being adapted to take advantage of transparent causal messages in order to further reduce the amount of (unnecessary) synchronization between clients and server replicas.

References

[1] Mohan Ahuja and Michel Raynal. An implementation of global flush primitives using counters. Technical Report CS94-342, University of California, San Diego, January 1994.

[2] Ozalp Babaoglu and Andre Schiper. On Group communication in Large-Scale Distributed Systems. In *Proceedings of the 6th ACM-SIGOPS European Workshop*, Dagstuhl, Germany, 1994.

[3] Kenneth P. Birman. A Response to Cheriton and Skeen's Criticism of Causal and Totally Ordered Communication. Technical report, Cornell University, October 1993.

[4] K.P. Birman and T.A. Joseph. Exploiting replication in distributed systems. In Sape Mullender, editor, *Distributed Systems*, pages 319–366. ACM Press Frontier Series, 1989.

[5] D. Cheriton and D. Skeen. Understanding the Limitations of Causally and Totally Ordered Communication. In *Proceedings of the 14th Symposium on Operating Systems Principles*, Asheville, NC, USA, December 1993.

[6] C. Fidge. Timestamps in Message-Passing Systems that Preserve the Partial Ordering. In *Proceedings of the 11th Australian Computer Science Conference*, 1988.

[7] Rivka Ladin, Barbara Liskov, Liuba Shrira, and Sanjay Ghemawat. Lazy Replication: Exploiting the Semantics of Distributed Services. Technical Report MIT / LCS / TR-84, MIT Laboratory for Computer Science, 1990.

[8] Leslie Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *CACM*, 7(21), July 1978.

[9] Larry L. Peterson, Nick C. Buchholdz, and Richard D. Schlichting. Preserving and Using Context Information in Interprocess Communication. *ACM Transactions on Computer Systems*, 7(3), August 1989.

[10] D. Powell, editor. *Delta-4 - A Generic Architecture for Dependable Distributed Computing*. ESPRIT Research Reports. Springer Verlag, November 1991.

[11] Robbert van Renesse. Causal Controversy at Le Mont St.-Michel. *ACM Operating Systems Review*, 27(2):44–53, April 1993.

[12] L. Rodrigues, Ellen Siegel, and P. Veríssimo. A Replication-Transparent Remote Invocation Protocol. In *Proceedings of the 13th Symposium on Reliable Distributed Systems*, Dana Point, California, October 1994. (To Appear).

[13] Andre Schiper and Aleta Ricciardi. Virtually-synchronous communication based on a weak failure suspector. In *Digest of Papers, The 23th International Symposium on Fault-Tolerant Computing*, pages 534–543, Toulouse, France, June 1993. IEEE.

[14] F. B. Schneider. The state machine approach: a tutorial. In *Proceedings of the Workshop on Fault-tolerant Distributed Computing*, Lecture Notes in Computer Science. Springer-Verlag, 1988.