Ensuring Causal Consistency in Internet–based Services with Arbitrary Failures

Daniela Tulone
ISTI, CNR
tulone@iei.pi.cnr.it

1 Introduction

The wide spread use of Internet has led to an explosion of highly decentralized systems and brought to light issues such as scalability, system performance for very large and dynamic populations, fault–tolerance also against attacks, and high availability of the service provided. Replication is a well–known technique for addressing both fault–tolerance and availability, and is employed in a number of real systems (i.e. the Swiss Stock Exchange, the French Air and Traffic Control, the FT–Corba and many Internet–based applications).
One of the primary challenges of replicating network services across the Internet is to maintain consistency among the replicas and guarantee the correct semantics of the objects replicated. Since providing strong consistency imposes performance overhead and limits system availability and scalability, protocols ensuring weaker consistencies are preferable when appropriate for the object semantics. A number of fault–tolerant protocols based on a client/server paradigm has been proposed to guarantee strong consistencies (i.e. linearizability) but little attention has been given to the design of protocols resilient to arbitrary failures and able to guarantee weaker consistencies. In fact, several Internet–based applications do not need a stringent requirement such as the linearizability of the operations invoked on the object replicas, but demand the causal order of the client invocations. For instance, in e-commerce applications such as an airline booking service, a user surfing the web in search of a cheap fare performs several reads before purchasing a ticket. The read operations can be performed concurrently but have to provide a system view that reflects the last completed update (i.e. if a user books a seat, then this change should be seen by the next client). In case of concurrent users competing for the same seat, the system can impose an order (i.e. lexicographic) to avoid inconsistencies in the replicas.
As malicious attacks and software errors are increasingly common and they can cause faulty nodes to exhibit Byzantine failures, it is important to include such failures in the protocol design.

Scalable consistency protocols suitable for the Internet have been proposed by Ahamad et al. [2], based on cached objects. Their solution guarantees a weaker consistency, *Local Consistency*, that is not suitable for interactive applications such as e–commerce where it is important for the user to have the most up–to–date system view. Causal memory has been analyzed by Ahamad et al. in [3] and several protocols have been proposed mostly based on vector timestamps, and assuming an *a priori* known number of processes. Notice that these protocols are not suitable for large and dynamic populations, that is the focus of our discussion. An adaptive protocol has been proposed by Ahamad et al. in [1] that explores causally consistent distributed services when multiple related services are replicated. Their system model classifies processes into three classes: *permanent servers*, *potential servers* and *clients*. Updates are performed only by permanent and potential servers, and they are responsible for propagating client updates. This approach is not suitable for Byzantine failures since a faulty server might corrupt or drop client updates.
In the next section we discuss the issues that arise when implementing causal consistency in a client/server paradigm in an asynchronous message system, in which read and write operations are performed remotely as the

object replicas reside at the servers.

2 Extending causal consistency to client/server paradigm

Ahamad et al. in [3] define the notion of *causal memory* for a set of processes interacting via shared memory through *read* and *write* operations and propose an implementation. The relation of causal order is consistent with the natural order of the operations issued by each process and with a *read–from* relation. According to this relation an operation o_1 causally precedes o_2 if o_1 is a write operation visible to the read operation o_2. Notice than in this model each process has a copy of the shared memory, therefore only write operations involve other processes, since a copy of the update is sent to the other processes. Notice that causal consistency increases the degree of concurrency since concurrent updates can be perceived in a different order at different processes. In a client/server paradigm both read and write operations are performed at the server side and therefore the meaning of the *read–from* relation changes.

In addition, most protocols that guarantee causal memory are based on vector timestamps: each process maintains a vector whose size is equal to the number of participating processes and whose entry reflects the local view of the updates issued by a process. This approach becomes infeasible when the number of processes is unbounded, since the vector timestamp is maintained at each server and transmitted along with the updates. Therefore variants of the vector timestamp or different approaches suitable for large and dynamic populations have to be studied. New issues arise also when considering attacks against the infrastructure: regarding the protocol requirements for ensuring causal consistency when b responses can be faulty and can even cooperate to fool the system, or its wait-free. We believe that the study of these issues deserve particular attention as they are significant for several widely used applications supporting large and dynamic populations.

References

[1] M. Ahamad, M. Raynal and G. Thia-Kime *An Adaptive Protocol for Implementing Causally Consistent Distributed Services.* In the Proc. of International Conference on Distributed Computing (ICDCS), May 1998.

[2] M. Ahamad and R. Kordale *Scalable Consistency Protocols for Distributed Services.* IEEE Transactions on Parallel and Distributed Systems.

[3] M.Ahamad, G.Neiger, P.Kohli, J.Burns, P.Hutto *Causal Memory: Definitions, Implementation, and Programming* Distributed Computing, Aug. 1995.