Understanding the power of the virtually-synchronous model (extended abstract)*

André Schiper†, Alain Sandoz
Département d'Informatique
Ecole Polytechnique Fédérale de Lausanne
CH-1015 Lausanne (Switzerland)
e-mail: {schiper,sandoz}@lse.epfl.ch

1 Introduction

Dependability in a distributed system is achievable only by introducing redundancy of key system components, and in particular, replication of system services which require to be fault-tolerant [9]. The specification of functionality, performance requirements and failure semantics for replicated components implies the choice of a replication policy for the given services. However, and in general, the set of replicas implementing a system service can be viewed as a group of active entities collaborating towards a common objective. Hence the importance of group oriented models of communication in a dependable distributed system.

These considerations have lead to various models and implementations of group-oriented communication, based either on the synchronous model [4] (bounded communication delays and synchronized clocks), or on the asynchronous model [1, 16, 8] (no bound on communication delays and no synchronized clocks). One of the most interesting asynchronous models is the so-called *virtually-synchronous* model defined by the Isis system [1, 2]. The model incorporates a *failure detector*, also called *group membership protocol*, that enables to bypass the impossibility of detecting failures in the asynchronous model [7]. However a major problem with the Isis model is that it fails to define a clear semantics on one crucial point, related to reliable multicasts. This makes it difficult to understand which problems can be solved in the model and which cannot.

The purpose of this paper is to define a clear semantics of the virtually-synchronous model, and to show that distributed commit can be solved in the model. This is in a sense not surprising, as it has been shown that distributed consensus can be solved in the asynchronous model with a very weak failure detector [6]. Considering this result, the virtually-synchronous model becomes extremely powerful, and more basic than the transaction model, providing an interesting broader picture of the problem of building fault-tolerant applications.

Section 2 briefly introduces the notion of *failure detector* and presents the virtually-synchronous model, both informally with respect to the Isis model, and more formally concerning the definition of reliable multicasts. Section 3 then considers the problem of distributed commit in the virtually-synchronous model.

*Project funded by the "Fonds national suisse" under contracts number 21-29847.90 and 21-32210.91 (European Esprit BRA project "BROADCAST" number 6360).

†Current address: Cornell University, Department of Computer Science, 308 Upson Hall, Ithaca, NY 14853-7501; E-mail: schiper@cs.cornell.edu

2 Virtually-synchronous model

The virtually-synchronous model (*vs*-model) considers processes with fail-stop failure semantics and incorporates a *failure detector* FD. FD sends information on crashed and recovered processes in the form of *views*, which are sets of processes that are considered as *alive* by the failure detector. The failure detector is allowed to be imperfect, i.e to make incorrect failure detections. Building a failure detector is done by a so called GMP (Group Membership Problem) protocol. [10] describes an implementation, considering fail-stop processes and network partitions. The FD can be seen as sending views to processes, and we explicitly consider the important distinction between *reception* and *delivery* of a view by a process, which is common for protocols ordering communication-related events in a distributed system [12].

Considering the failure detector, the virtually-synchronous model (*vs*-model) can be defined informally as follows: all significant events (e.g. delivery of multicasts issued by the application layer, and delivery of views) appear as if each had occurred at the same logical instant on all processes. This can be expressed as an ordering property: all significant events (i.e. delivery of multicasts and of views) occur on each process in the same order. This informal definition however has two drawbacks: it doesn't consider the partial *causal* ordering [3, 11] of message delivery; also it incompletely characterizes the delivery conditions of *reliable* multicasts, with respect to delivery of views.

We prefer therefore to consider that the *vs*-model only defines some ordering property on the delivery of reliable multicasts (issued by the application layer), with respect to delivery of views (issued by the failure detector). In other words, total or causal ordering of application messages are outside the scope of the model, and can be realized by additional protocol layers built on top of the *vs*-model.

The following example illustrates the semantic issues related to the delivery of reliable multicasts: consider a group g of processes, the current view $v_i(g) = \{p_1, p_2, p_3, p_4\}$, and process p_1 reliably multicasting message m to $v_i(g)$. Reliability means that m should be received by all non-faulty processes in $v_i(g)$, or by none. Suppose that p_1 and p_2 both crash while the multicast is under way and that both crashes are detected by the failure detector, and assume the following scenario:

- failure of p_1 is detected prior to failure of p_2, leading FD to define and multicast first $v_{i+1}(g) = \{p_2, p_3, p_4\}$ and then $v_{i+2}(g) = \{p_3, p_4\}$;
- p_2 delivers m and then delivers view $v_{i+1}(g)$;
- p_3 and p_4 never receive m, but deliver $v_{i+1}(g)$ and then $v_{i+2}(g)$.

Comments: (1) the scenario can arise using the current Isis *flush protocol* [3]; (2) it respects total ordering on reliable multicast delivery with respect to delivery of views; (3) the scenario should be prevented, in order to solve the distributed commit problem in the *vs*-model (see sect. 3). This leads to the following definition for reliable multicast in the *vs*-model [1]:

Definition D1. ***vs*-model**. Consider a group g, a view $v_i(g)$ and a message m multicast reliably to $v_i(g)$. The *vs*-model ensures the following property:
if $\exists p \in v_i(g)$ *which has delivered* m *in view* $v_i(g)$ *and has then delivered view* $v_{i+1}(g)$, *then all processes* $q \in v_i(g)$ *which have delivered view* $v_{i+1}(g)$ *have delivered* m *before* $v_{i+1}(g)$.

The preceding example clearly violates this definition: (1) process p_2 delivers m before $v_{i+1}(g)$;

[1] As reliable multicast is the only semantic issue considered here, we can without ambiguity consider that *semantics of reliable multicast in the vs-model* is identical to *semantics of the vs-model*.

(2) neither p_3 nor p_4 deliver m before delivering view $v_{i+1}(g)$.

Details of the implementation of D1 can be found in [13, 14]. The main idea is briefly described below:

- a global property $VSP(v_i(g), f)$ is identified, which is defined on view $v_i(g)$ and on failure information f delivered by the failure detector (f is a set of processes);
- if there exists f such that $VSP(v_i(g), f)$ is true, then definition D1 is fulfilled for multicasts in $v_i(g)$;
- the protocol (together with FD) ensures condition $VSP(v_i(g), f)$ for some f, and liveness on the detection of the property.

Section 3 shows how distributed commit can easily be solved in the *vs*-model satisfying definition D1.

3 Distributed commit in the *vs*-model

Consider the process group g and the current view $v_i(g)$. The distributed commit problem [15] can be solved in g using a *uniform reliable multicast* [5] primitive.

Definition D2. Uniform reliable multicast in the *vs*-model. Consider a group g, a view $v_i(g)$ and a message m multicast reliably to $v_i(g)$. The multicast is *uniform* iff:
if $\exists p \in v_i(g)$ *which has delivered* m *in view* $v_i(g)$ *then all processes* $q \in v_i(g)$ *which have delivered view* $v_{i+1}(g)$ *have delivered* m *before* $v_{i+1}(g)$.

This definition is clearly stronger than definition D1: definition D1 imposes conditions on delivery of a reliable multicast only on the processes which deliver view $v_{i+1}(g)$, whereas definition D2 considers delivery of message m by any process.

3.1 Distributed commit using uniform reliable multicast.

Distributed commit is trivially solved using a uniform reliable multicast. Consider view $v_i(g)$, and a function called *leader* on views, defining one privileged process in the view. Process $leader(v_i(g))$ is responsible for deciding *commit* or *abort*. Assume that $leader(v_i(g))$ has taken a decision D, and that this decision is multicast to $v_i(g)$ using an uniform reliable multicast. Consider any process $p \in v_i(g)$ not delivering the decision D before delivering a new view $v_{i+1}(g)$. The uniform reliable multicast ensures that *no process has delivered decision* D, i.e. no process has committed to any decision. Thus the processes in $v_{i+1}(g)$ can proceed in the same way as processes in view $v_i(g)$: process $leader(v_{i+1}(g))$ is responsible for taking a decision, and multicasting the decision to $v_{i+1}(g)$ using an uniform reliable multicast.

Note that the protocol does not overcome the impossibility result of non-blocking distributed commit [15]: the protocol can block, but if and only if (and as long as) the FD is unable to send new views.

3.2 Implementation of uniform reliable multicast.

Uniform reliable multicast of a message m to $v_i(g)$ can be implemented by a two phase protocol in the following way:

- the sender p starts by reliably multicasting m with an indication not to deliver m on reception;
- any process q receiving m acknowledges reception by sending $ack(id(m))$ to p;
- as soon as p has received $ack(id(m))$ from all processes in $v_i(g)$, p switches to the second phase;

- in the second phase p sends to each process in $v_i(g)$ a message $deliver(id(m))$. Upon reception of $deliver(id(m))$ by a process q, q delivers message m.

If failures occur during the multicast, a termination protocol is needed. The remarkable feature of the implementation based on the reliable multicast in the *vs*-model (sect. 2), is that the termination protocol does not require to issue any message! A process q having received m but not the authorization to deliver it, and ready to deliver view $v_{i+1}(g)$, can readily deliver m before $v_{i+1}(g)$. Details of the protocol and proofs can be found in [14].

4 Conclusion

We have shown that a careful definition of reliable multicast in the *vs*-model, greatly improves the power of the model, leading to an easy implementation of distributed commit. This should contribute to a better understanding of the *vs*-model, and perhaps to a wider recognition of its interest.

Moreover by showing that the transaction model can be built on top of the *vs*-model, the paper should clarify the position of the transaction model relatively to the *vs*-model, and lead to a better understanding of the field of fault-tolerance.

References

[1] K.Birman et al., *ISIS - A Distributed Programming Environment*, Cornell University, 1990.

[2] K.Birman, T.Joseph, *Exploiting Virtual Synchrony in Distributed Systems*, Proc 11th Symposium on Operating System Principles, Nov 1987, 123-187.

[3] K.Birman, A.Schiper, P.Stephenson, *Ligthweight Causal and Atomic Multicast*, ACM Trans. Comput. Syst. 9, 3 (Aug. 1991), 272-314.

[4] F.Christian, *Reaching agreement on processor-group membership in synchronous distributed systems* Distributed Computing, Vol.4, pp.175-187, 1991.

[5] T.D.Chandra, S.Toueg, *Time and Message Efficient Reliable Broadcast*, Proc. 4th Int Workshop on Distributed Algorithms, Bari, Sept 90, LNCS 486, Springer Verlag, 289-303.

[6] T.D.Chandra, S.Toueg, *Unreliable failure detectors for asynchronous systems*, Proc. 10th ACM Symposium on Principles of Distributed Computing, Montreal, Aug.1991, pp.325-340.

[7] M.J.Fisher, N.A.Lynch, M.S.Paterson, *Impossibility of Distributed Consensus with One Faulty Process*, Journal of the ACM, Vol 32, No2 (April 1985), 374-382.

[8] L.Peterson, N.Bucholz, R.Schlichting, *Preserving and using context information in interprocess communication*, ACM Trans. on Computer Systems, Vol 7, No 3 (Aug 1989), 217-246.

[9] D.Powel (ed.) *Delta-4: A Generic Architecture for Dependable Distributed Computing*, Esprit Research Report, Delta-4 Vol.1, Springer Verlag, 1991.

[10] A.Ricciardi, K.Birman, *Using process groups to implement failure detection in asynchronous environments*, Proc. 10th ACM Symp. on Principles of Distr. Computing, Montreal, Aug. 1991, 341-352.

[11] M.Raynal, A.Schiper and S.Toueg, *The causal ordering abstraction and a simple way to implement it*, Inf. Processing Letters 39 (1991), 343-350.

[12] A.Schiper, J.Eggli, A.Sandoz, *A New Algorithm to Implement Causal Ordering*, Proc 3rd Int Workshop on Distributed Algorithms, Nice, 1989, LNCS 392, Springer Verlag, 219-232.

[13] A.Schiper, A.Sandoz, *Termination detection in an asynchronous system subject to failures*, TR92-10, Computer Science Dept, EPFL, 1992.

[14] A.Schiper, A.Sandoz, *Uniform reliable multicast in a virtually synchronous environment*, LSE-TR92-13, Laboratoire de Systèmes d'Exploitation, Computer Science Dept, EPFL, 1992.

[15] D.Skeen, M.Stonebraker, *A Formal Model of Crash Recovery in a Distributed System* IEEE Trans. on Software Engineering, Vol.9/3, May 1983.

[16] P.Verissimo, L.Rodrigues, J.Rufino, *The Atomic Multicast Protocol (AMp), in* D.Powell (Ed), *Delta-4: A Generic Architecture for Dependable Distributed Computing*, Esprit Project 818/2252, Springer Verlag, 1991.