On Group Communication in Large-Scale Distributed Systems*

Özalp Babaoğlu† André Schiper‡

Abstract

An increasing number of applications with reliability requirements are being deployed in distributed systems that span large geographic distances or manage large numbers of objects. We consider the process group mechanism as an appropriate application structuring paradigm in such large-scale distributed systems. We give a formal characterization for the attribute "large scale" as applied to distributed systems and examine the technical problems that need to be solved in making group technology scalable. Our design advocates multiple roles for group membership over a minimal set of abstractions and primitives. The design is currently being implemented on top of "off-the-shelf" technologies for both communication and computation.

1 Introduction

Very large-scale distributed systems, such as the Internet, present interesting opportunities and challenges as infrastructures for collaborative distributed applications. The two aspects of large scale — geographic separation and number of sites — combine to provide an environment where computations may tap a huge number of resources with independent failure modes. In this paper we consider the process group mechanism as an appropriate structuring paradigm in large-scale distributed systems. In particular, we examine the architectural issues that arise in designing group communication subsystems capable of scaling efficiently to very large dimensions.

Process groups were initially introduced by the V-Kernel as a convenient way of expressing one-to-many communication structures [7]. Subsequently, the paradigm has been extended (most notably by the Isis system [3]) to include certain consistency guarantees in the presence of failures [11, 1, 13]. Successful experience with these systems, combined with efficient reliable multicast primitives [5], have led to the consideration of groups as adequate structuring mechanisms for building fault-tolerant distributed applications.

Experience with applying existing group technologies to a wide range of problems has led to the identification of certain recurring structures [10, 4, 3]. While some of these structures reflect the intrinsic properties of applications, others are artifacts of scaling. Conventional group technology is not easily scalable to the dimensions we envision. The reasons for this limitation can be traced to: (i) inability to distinguish group membership roles, (ii) inability to cope with highly dynamic communication structures and partitions. At best, forced scaling of conventional groups to large-scale systems results in unnatural or inefficient solutions in that a single logical group typically has to be decomposed into multiple sub-groups with complex overlapping structures.

Groups have traditionally been used for achieving one of two distinct purposes: (i) maintaining shared state, and (ii) disseminating information. While the former use is typical for load balancing or fault tolerance

*This work has been supported in part by the Commission of European Communities under ESPRIT Programme Basic Research Project 6360 (BROADCAST).

†Dipartimento di Matematica, Università di Bologna, 40127 Bologna (Italy), E-mail: ozalp@dm.unibo.it.

‡Département d'Informatique, Ecole Polytechnique Fédérale de Lausanne, 1015 Lausanne (Switzerland), E-mail: schiper@lse.epfl.ch.

reasons, and thus involves relatively small group dimensions, the latter may require diffusion to an extremely large group. Furthermore, guarantees required from the group communication subsystem for shared state management are typically much more stringent than those necessary for information diffusion. Thus, any group communication design that does not distinguish between these two types of functions and treats group membership anonymously is bound to have severe scaling problems with respect to group dimensions. The other aspect of "large scale" has to do with geographic distances that groups may span and is a concern independent of the group size — even small groups may span multiple continents. As a consequence, group communication designs for large-scale distributed systems have to effectively address both issues.

In this paper, we propose a group communication architecture that distinguishes multiple roles for membership. Informally, *core members* are those that manage shared state and have strong guarantees with respect to message delivery and membership changes. *Client members* interact with core members by requesting updates to or queries of the shared state. Clients track core membership changes in order to direct their requests efficiently to them. The third role is defined by *sink members* that only receive diffused information regarding the shared state maintained by the core. As the name implies, sinks cannot issue requests and are only end-points to one-way communication emanating from the core. With respect to various costs, the three roles define a hierarchy with core members being "heavy-weight" and clients/sinks being "light-weight". In our design, the potential for scalability of groups rests not in artificially decomposing them into multiple groups, but in identifying the appropriate roles for each of their members. By limiting the higher costs to only those members that manage shared state, the total cost of the group structure could be made relatively independent of its overall dimension. Core members, client members and sink members have similarities with notions introduced by Isis [3] and Horus [13]. We discuss the differences in Section 4.

We propose a layered architecture for implementing the design outlined above. At the lowest layer, we rely on a *view-synchronous communication* (VSC) service [15, 1, 5] providing a minimal set of primitives for reliable multicasts and membership changes. The core members rely on VSC semantics for internal communication and the handling of request/reply interactions with clients. The client and sink member abstractions are layered on top of the core services. In contrast to many existing designs, our minimal group interface provides *no* guarantees with respect to relative ordering of messages delivered by different group members. For those applications that require ordering semantics beyond those of VSC, we provide *total-order delivery* and *uniform delivery* [14] multicast service as separate layers. This minimalist layered architecture is similar to other modern group communication proposals such as Horus [13]. We are currently engaged in an implementation of the lowest layer in the architecture [2, 8].

2 Large-Scale Distributed Systems

We give a formal characterization for the attribute "large scale" with respect to communication. By their very nature, communication networks in such systems are not fully connected and are typically quite sparse. Both processes and communication links may fail by crashing. Given that the computing and communication resources may be shared by large numbers of processes and messages, the load on the system will be highly variable and unpredictable. Thus it is not possible to place bounds on communication delays and relative speeds of processes. As such, large-scale distributed systems have to be considered asynchronous.

Asynchronous systems place fundamental limits on what can be achieved by distributed computations in the presence of failures [9, 3]. In particular, the absence of an expected message from some process p cannot be attributed to its real cause — p may have crashed, p may be slow, communication may have been interrupted or communication may be slow. Consequently, the state of the system can be represented through a presumed *reachability* relation between pairs of processes.

Given two processes p_1 and p_2, let $\rightsquigarrow$ be a binary relation such that $p_1 \rightsquigarrow p_2$ iff p_2 is *reachable* from p_1 in the sense that p_1 can effectively communicate with p_2. Reachability information is typically derived from a system service called the *failure suspector* [6, 15]. In an asynchronous system, conclusions regarding reachability derived by individual processes can never be totally accurate and may be mutually inconsistent.

What characterizes a distributed system as being "large scale" with respect to communication are the following properties of the reachability relation:

1. *Asymmetric:* $(p_1 \rightsquigarrow p_2) \not\Rightarrow (p_2 \rightsquigarrow p_1)$.
2. *Non-transitive:* $(p_1 \rightsquigarrow p_2) \wedge (p_2 \rightsquigarrow p_3) \not\Rightarrow (p_1 \rightsquigarrow p_3)$.
3. *Non-connected:* The graph representing the reachability relation need not be connected.

In a large-scale system, communication delays could be comparable to inter-failure times, which may result in significant periods during which symmetry and transitivity of reachability are not satisfied due to inconsistencies either among the failure suspectors or the network routing tables. Property 3, which corresponds to network partitions [12], is a consequence of the sparse network connectivity and may be provoked by a small number of failures and exist for extended periods.

For our purposes, we define a *large-scale distributed system* to be one where communication may exhibit asymmetric, non-transitive, non-connected reachability relations and where the number of group members is far greater than the number of sites. All three properties of the reachability relation lead to what are commonly called *network partitions* [12], although with varying degrees of severity.

3 Membership Roles and Group Structures

We now give a more detailed description of our proposed group communication architecture with respect to different classes of membership roles and structural constraints. Each role class is defined by the semantics of the available communication primitives and the interactions that are possible with other group members.

3.1 Core Members

This role roughly corresponds to the notion of group membership in traditional systems. Internal communication among the core members is through reliable multicasts. Furthermore, changes to the composition of this class have certain ordering guarantees with respect to message deliveries.

We believe that the ability to manage shared state in an asynchronous large-scale distributed system requires a minimum set of fault-tolerant abstractions similar to those defined by *view-synchronous communication* (VSC) [15, 1].[1] The basic primitive of VSC is the reliable multicast of a message to a group of processes. For the multicast primitive to be terminating in an asynchronous system with failures, VSC includes a *membership service* that provides consistent information regarding the components of the group that are currently believed to be reachable. At each process, in addition to delivering multicast messages, VSC also delivers *views* denoting the set of reachable processes. Being layered on top of an asynchronous system, views constructed by VSC may be inaccurate with respect to the actual set of reachable processes. What is guaranteed by VSC, however, is that views are mutually consistent in the sense that they are agreed upon by all of the processes in the view before being delivered.

View changes define consistent cuts of the computation and are triggered by process crashes and recoveries, communication failures and repairs, network partitions and mergers, or explicit requests to either join and leave the group. The particular VSC membership service we rely upon is called *strong-partial* in that multiple views are allowed to exist concurrently but their intersection is guaranteed to be empty.

The real power of VSC is not in its individual components — reliable multicasts and membership service — but in their integration. Informally, VSC guarantees that message deliveries are totally ordered with respect to view changes.

[1] In [15, 1], the abstraction is called *virtually*-synchronous communication. We prefer to call it *view*-synchronous communication since the term *virtually-synchronous* is loaded with semantics, including causal-order delivery and state transfers, associated with the Isis system.

Core members are those that define the view of a group and that can communicate among each other through multicasts with VSC semantics. They rely on VSC properties to ensure the consistency of the shared state being managed. Seen externally, the core can be thought of as providing an abstract service, which is distributed and reliable. In order to facilitate efficient interaction, core members track the composition of both clients and sinks, as will be discussed in the next section.

3.2 Clients

Any process may interact with the core members of a group g (after resolving the group name) by issuing requests to it and receiving replies from it. In other words, a process need not join the group in order to interact with it through the abstract service interface. If, however, the interaction is frequent, a process may join the group g as a *client* for performance reasons. A process that has joined the group as a client not only informs the core of its existence, it also receives changes that occur to the core view. In this manner, deliveries of the request from the client to the core can be made more efficient (as the client has the most recent core view). If needed, delivery atomicity of a client request issued to the core members can be ensured by the core itself. Uniformity or total order of client requests will also be ensured by core members. Moreover, in most cases, these stronger semantic guarantees will have costs comparable to similar delivery guarantees within core members.

The number of clients is typically much larger than the number of core members. Since clients are not included in the group view, joins and leaves by them are relatively low-cost operations. Client composition is tracked by the core as part of its shared state information. Thus, join and leave operations executed by clients are implemented as requests to the core to update this information.

3.3 Sinks

In certain applications, it is common for processes to make a single request that provokes an arbitrary number of replies. This is typical for "information services" to which a process subscribes once and begins to receive a sequence of published data. Although this interaction fits in our client role, we prefer to distinguish it since it can lead to further optimization of the design with respect to scale.

A process wishing to receive information regarding the shared state managed by the core members joins the group as a *sink* member. Until it leaves the group, a sink member will receive state updates being disseminated by the core. The act of joining as a sink is implicitly a request for receiving such information. Since the core-sink interaction is one-way, the core tracks sink composition through its shared state but sinks are not informed of changes to the core view.

3.4 Structural Constraints

The only constraint we place on group structures is that different groups do not overlap in their core members. Our reasons for placing this limitation are rather pragmatic. Unless there are realistic applications with intrinsic needs for such overlapping structures, the additional complexity necessary to support them cannot be justified. There are also several technical issues that render the semantics of certain primitives not well defined when core members are allowed to overlap. Our design permits all other possibilities of overlap.

Since the lowest layers of our design provide VSC semantics only within the core, they cannot guarantee any properties for cross-group communication with or without overlaps. This is an explicit design choice in that appropriate semantics for cross-group communication cannot be determined by the infrastructure but can only be defined within the context of a particular application. Thus, we do not exclude the possibility of application-specific layers in the architecture that extend VSC or other ordering properties to multiple overlapping group structures.

4 Discussion

Core members communicate among themselves through multicasts that guarantee VSC. No guarantees are given with respect to uniformity [14], or with respect to the order in which messages are delivered. If uniformity or ordered delivery guarantees are required to ensure consistency of the service, the higher-layer software will have to build them on top of VSC. Similarly, the minimal interface has no notion of the particular replication or distribution strategy being employed by the core members. Obviously, primitives for certain common paradigms (e.g., primary-backup, active replication, etc.) will be available to applications through library functions. VSC can be understood as offering an exactly-once delivery guarantee for a message m delivered by a process p in a view v, among those processes that survive together with p into the same next view v'. For all other processes, only an at-most-once delivery guarantee can be ensured. Even for these processes, the at-most-once semantics can be turned into a stronger guarantee (e.g. exactly-once) through a mechanism such as state transfer. This allows us to make a clear distinction between core members on one side, and client/sinks on the other side, for which no exactly-once semantics is offered at the level of our minimal group interface.

At the level of our client/sink interface, both the requests and the replies are uninterpreted best-effort communication primitives, with an at-most-once delivery guarantee. That is, if the client does not fail and at least one core member in the current view is reachable, then the request will be received by it. Similarly, if the core member issuing the reply does not fail and the client is reachable, then the reply will be received by it. If the client request needs atomicity, uniformity or ordered delivery guarantees, then the client and the core members will have to cooperate to achieve it (e.g. either by having the client send the request to a single core member who then multicasts it internally, or by the client multicasting the request directly to the core). Similarly, as already mentioned, no exactly-once delivery guarantee is offered at the level of our client/core interface. To achieve this, retransmission and discarding of duplicates (if necessary) has to be handled above the level of the core/client interface. In other words, there are no exactly-once semantics attached to request-reply pairs. It is the application program implementing the service (core members) and the client who determine what semantics are actually achieved by the request-response interaction. The same observations apply also to sink members. The minimal interface provides only a simple best-effort multi-send primitive (with at-most-once delivery guarantee) from core members to a subset of the sink members in the current composition. As long as the core member does not fail, the message will be received by those sink members that are reachable. If stronger consistency guarantees are required, they will have to be programmed into the application being implemented by the core and sink members.

Core members can be compared to heavy-weight groups in Isis. In Isis, there is also the notion of a client, that can interact with heavy-weight groups through an RPC-like interface. The semantics of this interaction, however, is not precisely defined in terms of exactly-once versus at-most-once guarantees. Also, since Isis does not offer a primitive with uniform delivery guarantees, the client-core interaction sometimes fails to offer the semantics required by the application. This may force the Isis programmer to transform a client into a core member, leading to large heavy-weight groups and increased cost for view changes. Our design should avoid this trend by limiting core membership strictly to those processes managing replication, which are typically a small number. Isis also allows a multicast to be directed to the clients of a group, in addition to the group members. While this possibility can be used for Isis clients to subsume our sink role, the result is less flexible. In our case, we could implement multicasts to clients by having each client be also a sink member and having core members forward every VSC multicast to the sink members. This would, however, result in every VSC multicast being delivered to the sinks, even if it is not relevant. For example, in an application where sinks implement second-class replicas of some data, not every update (resulting in a VSC multicast within the core) should be directed to the sinks. Finally, the client and sink roles are fundamentally distinct: a client is usually not concerned by state updates that are diffused by core members to sink members. Our group design makes this distinction between active clients and passive sinks explicit and, should thus, offer the programmer a better match for the application requirements.

We are currently engaged in two separate implemention efforts of the minimal group communication interface consisting of a VSC multicast and a best-effort multisend [2, 8]. Our goal is to demostrate that versatile and efficient group communication paradigms for large-scale systems can be based on this simple interface.

Acknowledgments The architecture presented in this paper has benefited from extensive discussions with R. Davoli, L.A. Giachini (Bologna), Uwe Wilhelm, Cristoph Malloth (Ecole Polytechnique Fédérale de Lausanne), Paulo Veríssimo, Luis Rodriguez (INESC, Lisbon) and K. Birman (Cornell).

References

[1] Y. Amir, D. Dolev, S. Kramer and D. Malki. Transis: A Communication Sub-System for High Availability. In *Proc. 22nd Annual International Symposium on Fault-Tolerant Computing*, pages 76–84, July 1992.

[2] Ö. Babaoğlu, M.G. Baker, R. Davoli, and L.A. Giachini. RELACS: A Communications Infrastructure for Constructing Reliable Applications in Large-Scale Distributed Systems. Technical Report UBLCS-94-15, Laboratory for Computer Science, University of Bologna, Italy, June 1994.

[3] K. Birman. The Process Group Approach to Reliable Distributed Computing, *Communication of the ACM*, 9(12):36–53, December 1993.

[4] K. Birman and R. Cooper. The ISIS Project: Real Experience with a Fault-Tolerant Programming System. *ACM SIGOPS Operating Systems Review*, 25(2):103–107, April 1991.

[5] K. Birman, A. Schiper and P. Stephenson. Lightweight Causal and Atomic Multicast. *ACM Trans. Computing Systems*, 9(3):272–314, August 1991.

[6] T.D. Chandra and S. Toueg. Unreliable Failure Detectors for Asynchronous Systems. In *Proceedings 10th ACM Symposium on Principles of Distributed Computing*, pages 325–340. ACM, August 1991.

[7] D.R. Cheriton and W. Zwaenepoel. Distributed Process Groups in the V Kernel. *ACM Trans. Comput. Syst.*. 3(2):77–107, May 1985.

[8] P. Felber, C. Malloth, A. Schiper and U. Wilhelm. Phoenix: A Group-Oriented Infrastructure for Large-Scale Distributed Systems. Technical Report, EPFL-LSE, Lausanne, Switzerland. In preparation.

[9] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of ACM*, 32(2):374–382, April 1985.

[10] L. Liang, S.T. Chanson and G.W. Neufeld Process Groups and Group Communications: Classifications and Requirements. *IEEE Computer*, 23(2):56–66, February 1990.

[11] L.L. Peterson, N.C. Bucholz, and R.D. Schlichting. Preserving and using context information in interprocess communication. *ACM Transactions on Computer Systems*, 7(3):217–246, August 1989.

[12] A. Ricciardi, A. Schiper and K. Birman, Understanding Partitions and the "No Partition" Assumption. In *Proc. 4th IEEE Workshop on Future Trends of Distributed Systems*, Lisboa, September 1993.

[13] R. van Renesse, K. Birman, R. Cooper, B. Glade and P. Stephenson. The Horus System. In *Reliable Distributed Computing with the Isis Toolkit*, K.P. Birman, R. van Renesse (Ed.), IEEE Computer Society Press, Los Alamitos, CA, pages 133–147, 1993.

[14] A. Schiper and A. Sandoz. Uniform Reliable Multicast in a Virtually Synchronous Environment. In *Proc. 13th Int. Conference on Distributed Computing Systems*, pages 501–568, May 1993.

[15] A. Schiper and A. Ricciardi. Virtually-Synchronous Communication Based on a Weak Failure Suspector. In *Proc. 23rd Int. Symp. on Fault-Tolerant Computing*, Toulouse, pages 534–543, June 1993.