MojaveComm: A Robust Group Communication Library

David Noblet [*], Cristian Ţăpuş, and Jason Hickey

Computer Science Department, California Institute of Technology

{dnoblet,crt,jyh}@cs.caltech.edu

This paper introduces a fault-tolerant group communication protocol that is aimed at grid and wide area environments. This protocol has three layers: at the bottom, the first layer provides basic reliability over regular IP-multicast; the second layer provides atomic multicast within a group of nodes; and the third layer, sitting on top, guarantees the causal ordering of messages sent in groups with overlapping membership. The protocol can be used to implement sequential consistency. To prove the correctness of our protocol we have used a combination of model checking and mathematical proofs.

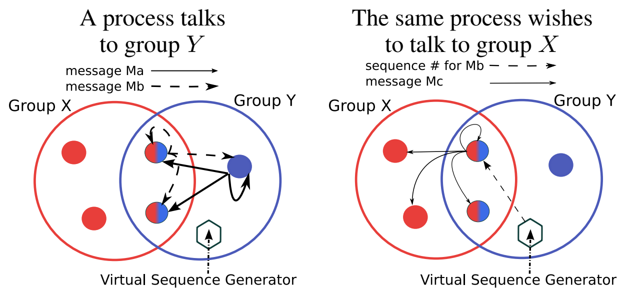
The major contributions of this work are: the design and implementation of a totally distributed (no central point of failure) group communication protocol with guarantees for a total ordering of messages, and an associated demo.

In order for the protocol to provide these guarantees, it presents the following abstractions. Primarily, the protocol assumes that the system consists of some finite (though arbitrarily large) number of *processes*, where each process has a unique identifier that is persistent across failures. The protocol defines a *group* to be a set of processes and a *view* (of a group) to be some subset of a group membership. Processes may belong to several groups at the same time.

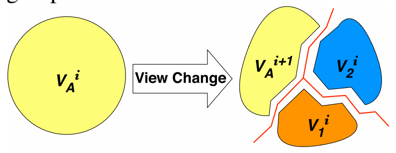
The upper layer of the protocol relies on the existence of the total ordering of the messages delivered within each group (provided by the middle layer). Additionally, it requires that messages sent by one process to different groups become part of the total order in each group in the same sequence in which the messages were issued. For example, if a process were to send two messages, m_1 and m_2, to groups g_1 and g_2 respectively, then message m_1 must become part of the order in group g_1 before m_2 becomes part of the order in group g_2. It is important to notice that we do not require that m_1 is delivered before we can send m_2; rather, we simply require that m_1 obtains a sequence number before m_2 does. While there is a penalty for implementing this restriction we minimize the cost by separating the message sequencing from the actual message delivery.

Each group may have several views and the views can overlap. Views are local to each process and they represent the local image of the membership of the group from the perspective of the process. In each group we implement the Virtual Synchrony model. When a request to join a group is received by a process or when a process leaves or is detected to have failed, the processes in the group trigger a *view change*. The view change is a synchronization point for the processes that survive it; it guarantees that membership changes within a process group are observed in the same order by all the group members that remain in the view. Moreover, view changes are totally ordered with respect to all regular messages that are sent in the system. This means that every two processes that observe the same two consecutive view changes, receive the same set of regular messages between the two view changes.

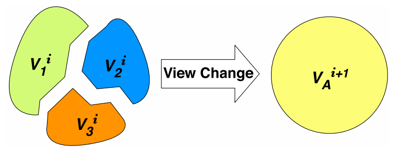
[*] student author


A process talks to group Y
message Ma
message Mb
Group X
Group Y
Virtual Sequence Generator
The same process wishes to talk to group X
sequence # for Mb
message Mc
Group X
Group Y
Virtual Sequence Generator

(a) Imposing an order on messages sent to different metadata groups.



(b) A 3-way split of a virtual server group due to a network failure.



(c) Three non-authoritative views merge to create a new authoritative view.

Each process that wants to join a group starts as a singleton view. It then contacts a directory service that provides hints as to the current membership of the group. These hints are used by the process to join other processes that belong to the same group. At this layer of the protocol we do not discriminate between different views of a group. It is only at the application level that one may treat separate views of the same group differenly based on some criteria. For example, one might want to allow only the members of a given view (like the one with highest cardinality) of a group to send or receive messages.

In order to provide an efficient communication mechanism for the underlying communication facilities of the group communication protocol, we have implemented a reliable multicast layer on top of the existing IP-multicast layer. In order to improve performance of the protocol, we opt to use a NACK-based approach, thus minimizing overhead (in the form of network traffic) in the absence of dropped messages or other failure.