Strong, Weak and Hybrid Group Membership

Farnam Jahanian
William L. Moran, Jr.

IBM T. J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

keywords: group membership, failure semantics, distributed protocols

Abstract

The focus of this position paper is the group membership problem in distributed systems. The paper describes a set of group membership protocols that were developed as part of a toolkit for building distributed/parallel applications on a cluster of workstations. The group membership service is the lowest layer in the toolkit, and it is the glue which unifies all the other layers. The group membership layer supports three protocols: weak, strong, and hybrid membership. These protocols differ significantly in the level of consistency and the number of messages exchanged in reaching agreement. This paper briefly describes each protocol with a focus on what is new about them and where they might be used.

1 Introduction

In a distributed environment, a collection of processors (or processes) can be grouped together to provide a service. A server group may be formed to provide high-availability by replicating a function on several nodes or to provide load balancing by distributing a resource on multiple nodes. Agreement among members on group membership in distributed systems is the focus of this position paper. The membership of a group can change because of a member join, a member departure, or the perceived departure of a member due to random communication delays. A member may depart from a group due to a normal shutdown, such as scheduled maintenance, or due to a failure. Since it is impossible to detect correctly the crash of a processor in an asynchronous environment, in certain cases, a member departure is in fact an assumed failure by the other members. A join can occur when a new member is added to a system or when a failed member is restarted after being repaired.

This paper presents a suite of group membership protocols for asynchronous systems. This suite consists of three protocols: weak, strong and hybrid membership. These protocols were implemented as part of a toolkit for developing distributed/parallel servers on a cluster of workstations. These protocols are currently being used in several experimental projects within IBM in applications ranging from database servers to process control systems. The objective of this position paper is to describe the motivation for the proposed processor membership protocols and to present a short overview of each protocol with some insight into our experience in building a prototype. The properties of each protocol have been formally specified in a temporal logic notation [7]. However, the formal specification of these protocols is beyond the scope of this presentation.

Section 2 presents the motivation for the suite of membership protocols. After introducing the system model and its failure semantics in Section 3, the paper presents an informal overview of the three protocols in Sections 4. Section 5 contains the concluding remarks.

2 Motivation

The group membership problem has been widely studied in recent years both for synchronous and asynchronous systems [2, 9, 3, 4, 5, 10]. A number of experimental and commercial systems have implemented variations of the group membership service [8, 1, 6, 2]. A group membership protocol is an agreement protocol for achieving a consistent system-wide view of the operational processes in the presence of member departures, member joins and communication failures. The degree of consistency among the views of the members has been a primary motivating factor for supporting multiple membership protocols. We examined several systems that depended on membership-like services; we also looked at the use of membership services in managing replicate data and system availability for applications such as databases, cluster management and monitoring, and process control. It became evident that different levels of consistency were required in different applications of the group membership service. Most existing implementations of group membership protocols are variations to what we refer to as the strong membership in that processes see members leaving and joining in the same
order. A weaker notion of consistency is acceptable in certain cases, e.g., cluster management and monitoring tools, or a client's view of the members of a database server connected to the clients via redundant communication networks. In fact, as discussed in a later section, the presence of multiple communication adaptors (and hence IP addresses) in a processor has a significant impact on the processor membership protocol which has not been addressed by other implementations.

The group membership service in our system supports three protocols: weak, strong, and hybrid membership. The proposed protocols differ significantly in the level of consistency and the overhead in reaching agreement when a membership change occurs. These protocols allow the architect of a system to balance the degree of consistency and the overhead of the group membership service based on the semantics required by different applications. The consistency required of a membership protocol is determined by the the semantics of the layer that uses the service.

Informally, the weak processor membership guarantees eventual agreement on the group membership when the system stabilizes after departures or joins. The strong membership protocol ensures that the operational processors see the same sequence of group membership changes in the system. The hybrid membership is a compromise between the other two protocols. While it preserves a partial order on membership changes, it does not guarantee that the same sequence is observed by the operational members. However, it does ensure a unique group leader in the system.

3 System Model & Failure Assumptions

This paper considers an asynchronous distributed system consisting of a set of processes \( P \) running on processors connected by a communication network. The dependency between the local process states is caused by message communication. No assumption is made about the network topology or the protocol for implementing the communication service. However, it is assumed that each process in \( P \) can send messages directly to the other processes. The communication subsystem can suffer a performance or an omission failure. The message delay between the moment a message is sent by a source process and the moment the message is received by its target process is potentially unbounded. A message delay depends on many random factors including the computation and communication load of the system, message retransmission due to corruption or loss, and other events such as page faults and context switching in an operating system.

There is no assumption about a global system clock or synchronized processor clocks in our computation model. However, we assume that the relative rate of change (the drift rate) between two clocks is insignificant for intervals of few seconds duration. (This assumption can be stated informally as follows: when a processor \( p_i \) observes 3 seconds on its local clock, another processor \( p_j \) does not see a 15 second interval. This does not imply that the processor clocks are approximately synchronized and see roughly the same absolute time. In fact, the deviation between processor clocks may grow without a known bound.) Without this assumption, the strong membership protocol would require an extra phase which would result in a 3-phase protocol instead of a 2-phase protocol, as described later.

In this model, processes are assumed to have crash failure semantics. Since the detection of a process (or processor) crash is done via timeouts, it is impossible to determine with absolute certainty whether a process has crashed in an asynchronous system. For example, when a number of heartbeat messages from a process are lost due to system overload or transmission error, the process may be seen to have failed by other processes. When a member is perceived to have crashed by the other members, the protocols ensure that the member exits and may rejoin the group in a consistent manner. Hence, a perceived crash of a group member is simulated as a crash failure in the system. The exact behavior of the group members varies from one protocol to another.

4 Overview of Protocols

The processor group membership problem is implemented by a set of daemon processes each running on a single processor. The failure of a daemon process is viewed as the failure of the processor. Informally, the objective is to provide agreement among group members in the presence of processor joins, processor departures and communication failures.

All three protocols: weak, strong and hybrid share many common features; these include the kinds of processor and group identifiers used, the way groups are structured, the way logical connections are maintained and mapped to real connections, the way bootstrapping is done, the way failures are detected, and to a large extent the kinds of messages exchanged. Of these, it is only necessary to understand a few to understand the protocols. Every processor gets assigned a unique identifier based on its IP address and an instance number that changes each time the processor changes state relative to the membership system. Using these identifiers, a group leader is chosen, and a logical ring structure is created. Every group has a unique group identifier; this identifier is simply the tuple identifying the group leader and its instance. Also, each group has a member sometimes referred to as a crown prince; this member would be the leader of the new group gotten by eliminating the leader from the old group. Every member has two neighbors in the ring, and it exchanges heartbeats with those neighbors. Failures are detected on the basis of missed heartbeats.
Weak Membership Protocol Description

The weak group membership protocol simply guarantees that if there are no failures for some period of time, there will be one consistent group in the system. Informally, the protocol ensures eventual agreement on the group membership when the system stabilizes after departures, joins or communication failures. As shown in Figure 1, it is possible that a membership change is not seen by a member, but eventually all operational members converge to the same view. The proposed weak membership protocol is a 1-phase protocol.

Given a domain G of all possible group identifiers, we define a binary relation &lt; on the elements in G. Informally, g1 &lt; g2 if a processor p joins the group g1 before joining g2. We define the binary relation &lt; to be the transitive closure of &lt; relation. The weak membership protocol ensures that the binary relation &lt; is a partial ordering relation: 1

i) irreflexive: g1 &lt; g1

ii) transitive: g1 &lt; g2 &amp; g2 &lt; g3 &rarr; g1 &lt; g3

iii) asymmetric: g1 &lt; g2 &rarr; g2 &lt; g1.

This is a very weak guarantee, and it is relatively easy to achieve. The major complication is to ensure the asymmetric property in a 1-phase protocol. Specifically, we need to ensure that if a processor first joins g1, and then joins g2, another processor does not join the two groups in the opposite order.

Group members exchange heartbeats in two directions around a logical ring. When a new member (or members) request to join an existing group, and the member failure is detected, a new group is formed. The leader forms a new group by simply sending out a NEW-GROUP message to all of the members. A processor that receives one of these messages can either join this group or not. Normally, it would join the group; however, if the message comes from a processor that was not a member of the old group, then it is necessary that this new leader should know which group each member is currently in. Specifically, this prevents interleaving of NEW-GROUP messages.

If a member fails to receive a new group message (because of message losses or long delay), then it will not exchange correct heartbeats with the members of the new group. The group will be reformed without this member. However, it is possible for one member to have a different membership view from others temporarily. The member with the inconsistent view will form a singleton group, and it will be allowed to join the others eventually.

Inconsistencies may occur when two groups merge, or when a member dies. All that happens when a member dies is that the member noticing this notifies the leader of this death. If the leader does not respond within a certain period of time, this member assumes that the group has failed and forms a new singleton group. If the member that has died is the group leader, the member noticing this notifies the crown prince. Once again, if a new group is not formed within a certain period of time, the member forms a new singleton group. This behavior is shown in Figure 1. The bootstrap process, group merge, and member death recovery are all that this protocol uses; this makes the protocol relative straightforward.

Strong Membership Protocol Description

The strong protocol differs from the other protocols mainly in that all group changes are acknowledged. In the weak protocol, the leader simply sends out the new group. As mentioned, the possibility arises that some of the recipients of the NEW-GROUP message will never get the message or the leader fails while forming a new group; if this happens, an inconsistent view of membership may exist among the operational members. In the strong protocol, instead of sending a NEW-GROUP message, the leader executes a 2-phase protocol. The leader sends a PTC (prepare to commit) message when a new group is being formed. A processor, upon receiving this message, if the message is from a valid leader, removes itself from its old group. At this point, the group of this processor is said to be IN-TRANSITION, i.e. it is a member in transition from one group to another. This processor then sends an ACK message to the leader. The leader, after collecting either ACKs or NAKs from all the members, or when it has timed out waiting, determines what the membership of the new group will be. It then sends out a COMMIT message containing the group membership to all the members. The important aspects of this protocol are that the group changes are acknowledged, and that for some period of time, all the members that will be in a new group are in transition. Specifically, the strong membership protocol ensures that:

- In a global system state, if a member p joins a group g locally, all other processors who are members of g either already joined g locally or are in the IN-TRANSITION state. (Figure 2 illustrates this property.)
- In the presence of network partitions (e.g., due to message loss or communication delay), the members of simultaneous groups are disjoint.

When a member dies, some neighbor notices the failure to send heartbeats, and it notifies the leader that its neighbor has died. Once again, it sets a timer to control how long it will wait for notification of a new group. In particular, the current implementation will notify the leader of the death several times before it finally abandons the group and forms its own singleton group. If the leader successfully forms a new group after the death of some member, it will perform the same steps as if two groups were being merged; the PTC &rarr; ACK &rarr; COMMIT sequence will occur normally. In the event that the leader

---

1The formal definition of the properties guaranteed by the protocols is beyond the scope of this paper.
has died, the neighbor notifies the crown prince as in the weak protocol. The crown prince then executes a 2-phase protocol forming a new group.

A possible inconsistency that might occur is that if the leader has not actually died, but has simply lost communications with one node, e.g., the next member in charge. The leader and the crown prince both would attempt to form a new group. This failure mode, unless avoided, would result in inconsistent views. The strong protocol allows the formation of two non-overlapping groups. This behavior is possible in a 2-phase protocol because of the assumption on the clock drifts mentioned in Section 3.

It is provably not possible to avoid all cases where the leader thinks some member has died, and that member thinks the leader died, all that can be done is to make this sort of partition not cause group inconsistencies. The way we avoid this problem is with the group identifiers. If we have a group \( G_1 = (P_{21}, P_{19}, P_{14}, P_{2}, P_1) \) we might have \( P_2 \) notifying \( P_{19} \) that \( P_{23} \) (the leader) had died. While \( P_{19} \) is taking over, we might have \( P_{23} \) also forming a new group. The end result might be two groups: \( G_2 = (P_{23}, P_{11}, P_1) \) and \( G_3 = (P_{19}, P_{2}, P_1) \) where \( P_1 \) is a member of both groups. This can happen if the messages to \( P_1 \) interleave in just the wrong way. The result is that \( G_2 \) will fail; these situations can result in both groups continuously failing and being restarted indefinitely. The worst side effect is that both groups now have a majority! This sort of oscillation between two groups must be avoided. The way this is avoided is with the group identifiers. In the above case, was in one of the two groups first, and it then moved to the other group. If every group transition is made dependent on the new leader correctly specifying what group the members were in, then this problem would be avoided. In the above example, both \( P_{19} \) and \( P_{23} \) would try to cause \( P_1 \) to make a transition from group \( G_1 \) to some new group. After one of them succeeds, the other will fail since \( P_1 \) would no longer be in \( G_1 \) but would be in some new group. This simple, low overhead solution avoids this problem altogether. The code to do this is implicit in the algorithm pseudo-code for all three protocols; a message that arrives with the wrong group is handled as an error condition, and an error message is returned to the sender.

Hybrid Protocol Description

The hybrid protocol is intended to provide a level of consistency between that of the strong and weak protocols. The protocol preserves the partial ordering of group membership changes. Furthermore, it guarantees that there will be a single leader for any group. The way this protocol insures this is by running a multi-phase commit whenever the leader of a group changes. Leader changes occur when two groups merge, or when the leader of a group dies. When two groups running the hybrid protocol merge, one of the two leader becomes the new group leader. The members of the (inferior) group with a new leader are forced to run a multi-phase commit. In other words, those members of the new group that are seeing a leader transition will do a PTC -> ACK -> COMMIT procedure. This insures that the members of the group that is being done away with will have acknowledged the new leader. The members of the group whose leader didn't change will be informed similar to the 1-phase protocol. This multiphase protocol is not necessary when a non-leader member dies, since the death of the member does not cause any leader changes. The sequence of steps necessary when a member dies under the hybrid protocol is identical to what happens under the weak protocol. Under the hybrid protocol, the sequence of steps performed when the leader dies is the same as under the strong protocol.

5 Concluding Remarks

This paper has described a set of group membership protocols with varying degrees of consistency and cost in reaching agreement. The current implementation of these protocols was done on AIX v3.1 \(^2\), but it runs on any BSD compatible system. The current implementation uses UDP datagram service on TCP/IP. We are currently experimenting with the use of these protocols on very fast point to point networks. The extended version of this paper addresses several implementation issues and optimization enhancements including: issues raised by the presence of multiple networks and multiple IP addresses for a processor; more detailed discussion on the handling of partitions; issues in adding a new membership protocol; our approach to the problem of simultaneous startup of multiple processors as it relates to group membership; protocol optimizations that reduce the number of messages exchanged or decrease latency in reaching agreement.

References


\(^2\)AIX is a trademark of IBM Corporation.


Figure 1: Membership Changes in Strong Protocol

Figure 2: Partial ordering of group changes.

Sequence of Events

T1: P1 gets JOIN_REQUEST from P2
T2: P3 and P4 get PTC from P1, exit their groups
T3: P3 gets PTC from P1, exits its group
T4: P1 has received all ACKs, forms new group G2
T5: P2 gets COMMIT, is now in group G3
T6: P4 gets COMMIT, is now in group G3
T7: P2 gets COMMIT, all members of G3 are now in G3
T8: P1 dies
T9: P2 notices P1 has died, exits G3, prepares to form group G4
T10: P3 and P4 get PTC messages from P2, leave G3
T11: P2 gets all ACKs, forms G4
T12: P2 gets commit, enters G4
T13: P4 gets commit, all members of G4 are now in G4

G1 = {P1,P2}  G4 = {P2,P3,P4}
G2 = {P4,P3}  G3 = {P1,P3,P3,P4}