Group Membership in a Synchronous Distributed System

Gianluigi Alari*, Augusto Ciuffoletti**

*CRS4, Via N. Sauro 10, 09123 Cagliari - e.mail: alari@CRS4
**Dipartimento di Informatica, Corso Italia 40, 56100 Pisa - e.mail: augusto@di.unipi.it

Abstract

This paper presents a solution to the (processor) group membership problem.

The methodology followed in designing the algorithm is summarized by the option to optimize the performance of the algorithm under the assumption that no failure occurs during its run, and adding the appropriate level of fault tolerance by implementing the algorithm as periodic and self-stabilizing.

The performance indicators we consider are the number of message exchanges (between neighbor units) and the time needed to reach the agreement. The algorithm relies on the existence of three basic distributed services: clock synchronization, reliable diffusion, and local agreement between two neighbors. We do not assume the presence of a reliable datagram service, which is more complex to implement than the previous. The presence of a privileged unit (initiator) is avoided, so elections are not needed.

1. Introduction

A *group* is a subset of the system characterized by a certain property of the members; a *group membership algorithm* keeps each member's description of the group acceptably updated despite the dynamic change of the group.

It is possible to give two meanings to the group. In the first, the group is considered as the set of units that deliberately agree to participate in a certain activity: several groups can be active at the same time (see [1,4,7]). The second considers the group as the set of correct units that can cooperate (see [2]), and admits only one group. In this paper we adopt an hybrid view: the joining or departure from a group is a deliberate event, but we admit, at least in principle, the existence of only one group. Non-deterministic departures caused by failures are treated as recoverable transient faults.

Concerning the engineering of the algorithm, we have given priority to simplicity and efficiency in absence of failures, since *simplicity* implies design reliability and ease of test, while *efficiency in absence of failures* implies that under normal conditions the algorithm exhibits an optimal performance. Taking this approach, we opted for introducing a number of restrictions:

- every unit or link works properly;
- the clocks of the units are kept synchronized;
- the group membership algorithm is run at predetermined times;
- the group is unique, and it does not change in the while between two successive runs of the algorithm;
- a diffusion based broadcast service exists which is reliable and takes a fixed amount of time to complete;
- two neighbors always agree about the state of the link between them.

However, the intent of achieving simplicity and efficiency by restrictive assumptions is apparently conflicting with the reliability of the algorithm. To resolve this conflict we have designed the algorithm so that it is self-stabilizing [6] with respect to a number of possible "perturbations": the algorithm exhibits an anomalous behavior in case the above assumptions are not met, but spontaneously recovers as soon as they are re-established. In other words, transient failures in the support are reflected in transient failures of the algorithm. The last section is devoted to the discussion of the self-stabilizing properties of the algorithm

2. System Model

The system is represented as a non-directed graph *System*, where the nodes indicate the processing *units* (labelled with p, q, p_x, p_y, …) and the arcs indicate direct communication *links* among two units. Each unit p_X is directly connected to a small number of units, that are said *adjacent* or *neighbors* to p_X: we indicate by *deg* the maximum number of neighbors of a unit (the degree of the graph). Between two units there is at most one link, and the diameter of the system (the maximum length of the

This work has been carried out with the financial support of Sardinia Regional Authority

shortest path connecting two nodes) is indicated by *diam*. The software which controls the operation of the units is organized into *tasks*, which run concurrently.

2.1 Communication

The algorithm makes use of two communication facilities: a local service that enables a unit to agree with a neighbor whether the link between them is to be considered *in use* or not for what is concerning the group operation, and a global service that enables a unit to broadcast to all (reachable) units that a certain link is not *in use.*

The units do not directly request their exclusion or inclusion in the group, but the exclusion or inclusion in the group for each of the links they share: the fact that the unit will be or not in the group comes as a consequence, since the system can split into partitions when considering only the links *in use*. This option adds flexibility and improves the stability of the algorithm.

The function *InUse*, that renders the service defined above, takes a fixed amount of time to return the desired value: we indicate that time with ι.

Specification of Local Agreement:
Let p_X be a unit that calls InUse(p_Y), where p_Y is neighbor to p_X, at time t; by time [t, t+ι] both p_X and p_Y agree about whether the link between p_X and p_Y is in use or not.

The system offers a reliable broadcast facility ensuring that an information, originated by a unit, reaches every other unit in the same partition within a fixed time Δ.

Specification of Reliable Broadcast:
Let p be a unit that originates a broadcast at time t; at time t+Δ, all and only the units in the partition that contains p accept the broadcast.

It may be considered as similar to the timed interactive consistency described in [8], but we require that the broadcast is always received by all the units belonging to the same partition of the originator. The reliable broadcast primitive is easy to implement using a diffusion strategy (see [5] and [3]) through *in use* links.

2.2 Timing

The units have two ways to control the scheduling of a certain task:

- defining a timeout for the execution of the task: the task is interrupted if it is not completed when the timeout elapses.
- firing the task at a certain time. The request is skipped if the present time is higher than the firing time.

Both are based on a global time reference. Since units are distinct physical entities, the timing cannot be absolutely precise. However, in the next two sections, we assume the timing is exact; in section 5.3 we analyse the consequence of non null timing errors.

3. Formal statement of the problem

The group is defined as follows:

Group definition (static part):
The group G is a subgraph of the system such that two units in G are connected either directly or transitively, through at least one path composed of links that are in use *and:*
a) the group is a maximal element (with respect to $\subseteq$) among the sub-graphs having this property and
b) G contains more than half the units in the system.

We omit the straightforward proof that the system cannot contain more than one group.

We allow group changes to occur only during periodic time intervals:

Group definition (dynamic part):
Links can change their state from in use *to* not in use *or viceversa only at some time t such that $\exists$ integer $k \geq 0$, $t \in [t_0 + k \cdot T, t_0 + k \cdot T + \tau]$ where τ (group change window) and T (group change period) are appropriate constants such that $\tau < T$.*

Since the group changes dynamically, we need a function that yields the group at a certain time. According to [2] this function is called DSV (Distributed System View):

DSV: Time -> SubgraphsOf(System)

The local knowledge of the group membership is represented by the function MSV_p (Member System View):

MSV_p: Time -> SubgraphsOf(System)

The consistency requirement for the group membership algorithm is the following:

Membership consistency requirement:
Given a real time t_0, corresponding to the group initialization time
$\forall$ integer $k \geq 0$, at any time $t \in [t_0+k \cdot T+\delta, t_0+(k+1) \cdot T]$,
$\forall$ $p \in DSV(t_0+k \cdot T+\tau)$, $MSV_p(t)=DSV(t_0+k \cdot T+\tau)$
and
$\forall$ $p \notin DSV(t_0+k \cdot T+\tau)$, $MSV_p(t) = \emptyset$
where δ (group membership shift) is an appropriate constant such that $\tau < \delta < T$.

In other words, we want to implement a periodic group membership algorithm, that assigns to the MSV of the members of the group the value of the DSV; the update is run every T time units, and takes δ time units to terminate. During the first τ time units in the time interval dedicated to the group membership algorithm, the links are allowed to change their state, that will remain unaltered until the next run of the algorithm. The δ - τ remaining time units are reserved to establish a valid value

for the MSVs on each unit of the system.

4. The protocol

The group membership algorithm we propose starts from scratch each time it is run, without considering the MSVs used during the previous period. This option is motivated by the intent of designing a self-stabilizing algorithm.

The algorithm is divided into three phases (see Fig. 1).

During the *local agreement phase*, each unit agrees with each of the neighbors whether the link between them is *in use* or not: this is done using the *InUse* primitive that terminates within τ time units.

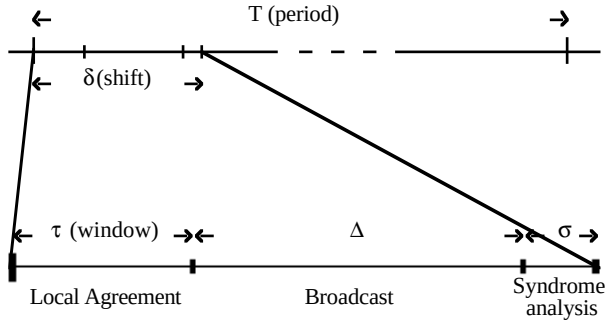

Fig. 1: Timing of the algorithm

During the next *broadcast phase*, each unit broadcasts a description of the outgoing links labeled as *not in use* during the previous phase, and collects the broadcasts coming from other units: the message (p_x,p_y) is broadcast by p_x to indicate that the link with p_y is *not in use*. After Δ time units each unit has certainly collected all the broadcasts originated within the partition it belongs to. Therefore the *Broadcast Phase* timeouts after Δ time units: we call *syndrome** *at p* or S_p the set of broadcasts received by unit p during this time interval.

During the *syndrome processing phase* each correct unit uses the syndrome to determine the members of the partition it belongs to. Let p be a generic unit and $\#_p$ the partition that contains p: for each link (p_x,p_y), p can draw the following conclusions based on the content of S_p:

a) if $(p_x,p_y) \notin S_p$, and $(p_y,p_x) \notin S_p$ then
   i) $p_x \notin \#_p$ and $p_y \notin \#_p$ or
   ii) $p_x \in \#_p$ and $p_y \in \#_p$ and the link is *in use*;
b) if $(p_x,p_y) \in S_p$ and $(p_y,p_x) \in S_p$ then $p_x \in \#_p$ and $p_y \in \#_p$ and the link is not *in use*;
c) if $(p_x,p_y) \in S_p$ and $(p_y,p_x) \notin S_p$ then $p_x \in \#_p$ and $p_y \notin \#_p$ and the link is not *in use*.

Case a) can be further discriminated: if there is a path from p to p_x or p_y that does not contain links in S_p, then p can conclude that a-ii) is true, otherwise a-i) holds.

In conclusion, every unit can compute the partition (both the units and the links *in use*) of which it is a part. If this counts more than half the units of the system, this is necessarily the DSV at that time. Otherwise, p concludes that its partition is not the group.

5. Analysis of the algorithm

Each time the algorithm is run, it consumes an amount of time and of channel bandwidth that is related to the number of links *not in use*: let X be the number of these links, and n the number of links in the system.

The amount of time depends on the length of each of the three phases: we will take as a unit the maximum time needed to perform a communication among adjacent units. The local agreement phase takes $\tau \in O(deg)$ time units if tests are performed sequentially. The broadcast phase requires $O(X)$ broadcasts where X is the number of not *in use* links. If the broadcast is by diffusion and if $O(X)$ broadcasts do not overload the system, then they all run in parallel, and the phase takes a time $\Delta \in O(diam)$. The syndrome analysis does not require any message: we assume that it takes a time that is $O(\Delta)$. The time required to complete the algorithm is therefore $O(deg+diam)$. In the most favorable case, the algorithm in [2] requires the circulation of the MSV among all the members arranged in a virtual ring supported by a datagram protocol: this takes a time $O(n \cdot diam)$, corresponding to n datagram communications among units arbitrarily placed in the system. Thus the protocol presented in this paper compares favorably with that presented by Cristian for what is concerning the time overhead.

The number of exchanged messages during the algorithm presented in this paper is $O(n \cdot X)$, i.e. the number of messages needed to perform X broadcasts. In Cristian's algorithm we have $O(n)$ datagram messages in the best case, that correspond to $O(n \cdot diam)$ communications on direct links. The number of messages exchanged by the two algorithms is therefore comparable.

5.1. Approximate timing

To extend the algorithm to imprecise timing, two sources of timing errors are considered: the clock reading error, which is kept within a limited value ε (see Schneider [1986], and Cristian [1989]), and the scheduling

*We have deliberately chosen the same term used in [9], since the concept is similar. However, the problem we solve is different: there the authors were interested in diagnosing the components of a system, given that the testing components could lie about the result of the test and that all the test results are shared by all the components. Here we assume that a component never lies (in the worst case it remains silent), but we do not assume the existence of a shared memory and we keep as low as possible the workload on the communication subsystem.

delay, which is kept lower than μ. The first is the kind of indeterminacy that is usually introduced when considering the local copies of a global value. The other is induced by some non-determinism in the scheduler. We omit the proof that any event scheduled at time T, occurs at a real time $t \in [T-\varepsilon, T+\varepsilon+\mu]$. All the correct units schedule the broadcast phase at time $T_0+kT+\tau$, and therefore each of them effectively enters this phase at some time within the interval $[T_0+kT+\tau-\varepsilon, T_0+kT+\tau+\varepsilon+\mu]$. Therefore the latest broadcast is sent at $T_0+kT+\tau+\varepsilon+\mu$, and we must ensure that units listen for broadcasts until $T_0+kT+\tau+\varepsilon+\mu+\Delta = T_0+kT+\tau+2\varepsilon+\mu+\Delta$. In other words, the broadcast phase must be extended $2\varepsilon+\mu$ time units over the time that is needed by the broadcast. This modification allows a limited timing error.

5.2. Concerning Self-Stabilization

We require that the MSV of each member be equal to the DSV when the update algorithm is not running: this is the condition that characterizes a *legitimate* state.

In this view the leading idea of our algorithm is very simple: we periodically force the system into a *legitimate* state. This operation necessarily erases any possible effect of preceding transient failures. The algorithm that, after this point, preserves legitimacy in absence of failures has been presented in the previous sections.

To implement this purpose, the state that is periodically forced is decomposed into a part K_0 independent from previous computations (i.e. constant), and in a part C that is controlled by an underlying self-stabilizing protocol that serves to trigger the forcing. The self-stabilizing property of C guarantees that the forcing is executed consistently by every unit; K_0 should contain all the data needed to derive a state K_x that reflects on each unit the reality that we intend to represent, i.e. the DSV. In our case, C is the set of the clocks of the units, and K_0 consists in the values of the constants τ, Δ and in the static description of the system *System*, replicated on each unit.

Now it is evident the importance of the clock synchronization protocol, which has in charge the self-stabilization of C: the legitimacy condition it must ensure is that local clocks are sufficiently precise. Clock synchronization algorithms that meet strong reliability requirements have been studied during the 80's, and a survey is in [10].

5.3. Response in case of permanent failures

We consider first the case of the silent failure of a unit, and of the disconnection of a link. Both cases are rendered labeling as *not in use* either the failed link, or all the links connected to the failed unit: this can be obtained including in the code of the *InUse* function a timeout that returns false in the case the neighbor does not respond within the expected time.

A permanent failure of the services of broadcast or local agreement that cannot be reduced to a number of link or unit failures is not regarded as meaningful.

6. Conclusions

Despite the extremely restrictive assumptions, that exclude any form of failure of the hardware support and of the services offered by the system, the algorithm that we present offers a quite reliable group membership service: it responds to transient failures exhibiting in its turn transient failures. This result is obtained by designing the algorithm as self-stabilizing. A permanent link disconnection and a silent failure of a unit is also tolerated though exhibiting a transient failure.

The algorithm makes use of a global time reference, that is known to each unit with a limited error. The permanent failure of the time diffusion service disrupts the consistency of the group membership algorithm.

The algorithm takes a limited amount of time, comparable with the time required by a broadcast.

Bibliographical References.

[1] Cheriton D.R. and Zwaenepoel W. Distributed Process Groups in the V Kernel. *ACM ToCS;* 3(2): 77-107.

[2] Cristian F. Reaching agreement on processor-group membership in synchronous distributed systems. *Distributed Computing;* 1991; (4): 175-187.

[3] Cristian F.; Aghili H.; Strong R. Atomic broadcast: from simple message diffusion to Byzantine agreement. *15th FTCS;* June, 1985; Ann Arbor, USA; 1985. 200-206; ISBN: 0-8186-0618-5.

[4] Birman K.P.; Cooper R.; Gleeson B. Programming with Process Groups: Groups and Multicast Semantics. Cornell University Ithaca N.Y.; January 1991; TR 91-1185.

[5] Demers A.; Greene D.; Hauser C.; Irish W.; Larson J.; Shenker S.; Sturgis H.; Swinehart D.; Terry D. Epidemic Algorithm for replicated database maintenance. *ACM Symp. on Princ. of Distr. Comp.;* Vancouver (CANADA); 1987. 1-12.

[6] Dijkstra E.W. Self-stabilizing Systems in Spite of Distributed Control. *CACM;* 17(11): 643-644.

[7] Golding A.R. Weak-consistency Group Communication and membership. University of California - Santa Cruz; Phd Dissertation; December 1992; available as ucsc-crl-92-52 at ftp site ftp.cse.ucsc.edu.

[8] Lamport L. Using time instead of timeout for fault-tolerant distributed systems. *ACM ToPLaS;* 6(2): 254-280.

[9] Preparata F.P., Metze G., and R.T. Chien On the connection assignement problem of diagnosable stystems. *IEEE ToEC;* Vol. EC-16, No 6, 848-854.

[10] Schneider F. B. *A Paradigm for Reliable Clock Synchronization:* Cornell University; TR 86-735.