Distributed Membership Management Protocol for Flexible Group Communication *

4O－6

Takayuki Tachikawa, Hiroaki Higaki, and Makoto Takizawa †
Tokyo Denki University ‡
e-mail{tachi, hig, taki}@takilab.k.dendai.ac.jp

1 Introduction

Distributed systems are composed of multiple computers connected by communication networks. In distributed applications like teleconferences and teleclassrooms, a group of multiple objects have to be cooperated. The group communication protocol is required to coordinate the cooperation of the objects in the group. In the group communication, the following services have to be supported: (G1) A message sent by the member object is received by one or multiple destination members in the group. (G2) A member object in the group receives messages in the causal order.

In the teleconferences, some new member joins the conference and a member leaves the conference. Furthermore, some object may be faulty. If the membership of the group is changed, every member object has to reach agreement on the membership. By the group membership protocol, only and all the member objects make agreement on the membership of the group. Reiter [2] discusses a centralized membership protocol where one coordinator object coordinates the cooperation among the objects and the data transmission is stopped during the execution of the membership protocol. In this paper, we would like to discuss how to support the services (G1) and (G2) without stopping the data transmission in the presence of the membership change.

2 System Model

A group G is composed of multiple objects O_1, ..., O_n ($n \geq 2$) interconnected by reliable high-speed networks [Figure 1]. We assume that (1) There is a reliable, synchronous communication link between every two objects. In addition, the transmission delay is bounded to be δ time units. (2) The objects may stop by fault.

3 Changes in Group

Each object O_i in the group G has a view $view_i(G)$ which denotes what objects O_i perceives are included in G. If G is not changed, every member object of G has the same view. If O_i is in G or would like to be in G, $O_i \in view_i(G)$. $view_i(G)$ is changed if O_i finds the membership change of G. If O_i finds that an object O_k leaves G, O_k is removed from $view_i(G)$. Even if O_i finds O_k's leaving, another O_j may not find it. Thus, every pair of views $view_i(G)$ and $view_j(G)$ are not always identical. If $view_i(G) \cup view_j(G) \neq \phi$, O_i and O_j are related. Let $rel(O_i)$ be a set of objects which are related with O_i.

[Complete group] For every pair of objects O_i and O_j, a collection of objects $G = rel(O_i)$ is referred to as *complete* group if $O_j \in rel(O_i)$, $rel(O_i) = rel(O_j)$, and $view_i(G) = view_j(G)$. □

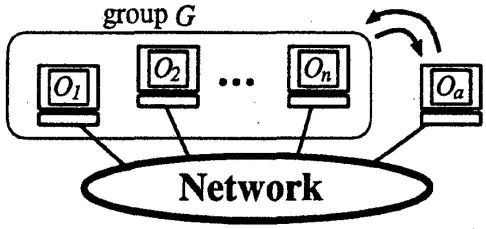

Figure 1: Group

The membership of the group G is changed if some member objects leave G, new objects join G, or member objects are faulty. In this paper, we assume that an object sends *join* and *leaving* requests to G if the object would like to join and leave G, respectively.

[Membership changes]
(1) An object O_{n+1} joins the group G, ({ O_1, ..., O_n }, O_{n+1}) = { O_1, ..., O_n, O_{n+1} }.
(2) An object O_i leaves G, ({ O_1, ..., O_n }, O_i) = { O_1, ..., O_{i-1}, O_{i+1}, ..., O_n }. □

4 Causally Ordered Delivery

Messages sent in G are required to be delivered in the causal order $\rightarrow$.

[Causal order] A message m_1 *causally precedes* m_2 ($m_1 \rightarrow m_2$) iff (1) an object sends m_1 before m_2, (2) an object sends m_2 after receiving m_1, or (3) there exists a message m_3 such that $m_1 \rightarrow m_3 \rightarrow m_2$. □

Messages can be ordered by using the vector clock [1]. In the system of the vector clocks, the time domain is represented by a set of n-dimensional vector.

[Vector operations] For every pair of vectors $VC_1 = \langle VC_{11}, ..., VC_{1n} \rangle$ and $VC_2 = \langle VC_{21}, ..., VC_{2n} \rangle$, the following relation holds:
(1) $VC_1 = VC_2$ iff $VC_{1i} = VC_{2i}$ for $i = 1, ..., n$.
(2) $VC_1 < VC_2$ iff $VC_{1i} \leq VC_{2i}$ for $i = 1, ..., n$ and $VC_{1j} < VC_{2j}$ for some j.
(3) $max(VC_1, VC_2) = \langle VC_{31}, ..., VC_{3n} \rangle$. Here, $VC_{3i} = max(VC_{1i}, VC_{2i})$ for $i = 1, ..., n$. □

A vector time VC is given in a vector $\langle VC_1, ..., VC_n \rangle$ where each element VC_i represents an object O_i in a group $G = \langle O_1, ..., O_n \rangle$. O_i has a variable $VC_i = \langle VC_{i1}, ..., VC_{in} \rangle$ denoting a vector time. VC_{ij} is initially 0 for $j = 1, ..., n$. Each message m sent by O_i carries a timestamp $m.VC = \langle m.VC_1, ..., m.VC_n \rangle$. O_i sends and receives messages by the following rule.

[Vector clock rule]
(1) Each time O_i sends a message m, $VC_{ii} := VC_{ii} + 1$; $m.VC := VC_i$;
(2) Each time O_i receives a message m from O_j, $VC_i := max(VC_i, m.VC)$; □

[Proposition] For every pair of messages m_1 and m_2,

---

*やわらかいグループ通信のための分散型メンバ管理プロトコル
†立川 敬行　桧垣 博章　滝沢 誠
‡東京電機大学

$m_1 \rightarrow m_2$ iff $m_1.VC < m_2.VC$. □

5 Membership Management

Here, let O be a set of possible objects. For a group G, let G_k be a membership of G, i.e. $G_k \subseteq O$. The membership G_k of G is changed to G_{k+1} ($\subseteq$ O) if the membership is changed. If G_k is changed to G_{k+1}, all the objects in G_{k+1} have to agree on G_{k+1}. Here, G_k is referred to as the kth version of G. The scheme of the vector clock VC_i denotes the view $view_i(G)$ of O_i. If O_i detects the membership change, the vector clock scheme of VC_i is updated in O_i so that the new scheme represents the new membership. The dimension of the vector clock is changed according to the update of the vector clock scheme. If the scheme of the vector clock is changed, the version of the vector clock is said to be changed. Each object O_i has a variable ver_i denotes the version number of VC_i. ver_i is updated by the following procedure.

[Update of version number]

(1) $ver_i := ver_i + 1$.

(2) The vector clock scheme of VC_i is updated. □

If O_{n+1} would like to join G, O_{n+1} sends a *join* request to one object. Another object which would like to join G may send the *join request* to an object. Similarly, if O_j would like to leave G, O_j sends a *leaving* request to one object. The faulty object O_j is detected by an object if O_i had not received one message from O_j for some predetermined time units, say 2δ. Then, O_i initiates the membership procedure.

Each object O_i has two kinds of variables, L_i and J_i. L_i denotes a set of objects which are detected to leave G, and J_i denotes a set of objects which are detected to join G. Initially, $L_i = J_i = \phi$ and O_i is in a *normal* state. While the membership of G is not changed, $L_i = J_i = \phi$. If O_i detects O_j's joining and leaving G, O_j is added to L_i and J_i, respectively. $G - L_i \cup J_i$ denotes a view $view_i(G)$ of O_i in G.

[Membership procedure]

(1) If L_i or J_i is changed, O_i sends a membership message m with L_i and J_i to all objects in $G \cup J_i$. O_i is in an *updating* state.

(2) On receipt of the membership message m with L_i and J_i from O_i, O_j manipulates L_j and J_j as $L_j := L_j \cup L_i$ and $J_j := J_j \cup J_i$. O_j is in an updating state.

(3) If L_j and J_j are changed, O_j sends the membership message with L_i and J_j to all the objects.

(4) If O_k receives the membership message with L_h and J_h from every object O_h in $G - L_k \cup J_k$, and $L_h = L_h$ and $J_h = J_h$, then O_k updates the membership of G to $G - L_k \cup J_k$. The version number ver_k is incremented by one. O_k leaves the updating state and is in a normal state. □

Figure 2 shows an example of a group $G = \{ O_1, ..., O_5 \}$. O_3 would like to leave G and O_6 would like to join G. O_3 sends a leaving request r_1 to O_1. On receipt of r_1, $L_1 = \{O_3\}$ and $J_1 = \phi$. O_1 sends the membership message m_1 with L_1 and J_1 to all the objects, i.e. O_1, O_2, O_4, and O_5. O_6 sends a join request r_2 to O_5. On receipt of r_2, $L_5 = \phi$ and $J_5 = \phi$, and O_5 sends the membership message m_2 with L_5 and J_5 to all the objects. O_2 receives m_1 and m_2. Here, $L_2 = L_1 \cup L_5 = \{ O_6 \}$ and $J_2 = J_1 \cup J_5 = \{ O_3 \}$. Since L_2 and J_2 are changed, O_2 sends the membership message m_3 with L_2 and J_2 to all the objects in $G = G - L_2 \cup J_2 = \{ O_1, O_2, O_4, O_5, O_6 \}$. On receipt of m_3, L_6 and J_6 gets $\{ O_3 \}$ and $\{ O_6 \}$ in O_6, respectively and O_6 sends the membership message to all the objects. Every object in $\{ O_1, O_2, O_4, O_5, O_6 \}$ has the same view.

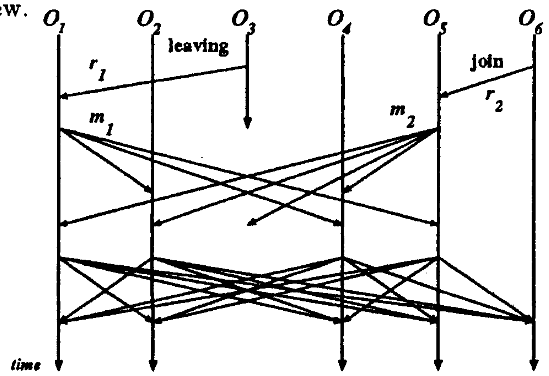

Figure 2: Membership change

6 Delivery of Messages

Messages sent in the group G are ordered by the following rule.

[Ordering (O) rule] For every pair of messages m_1 and m_2, m_1 *precedes* m_2 if the following condition holds: (1) if $m_1.ver = m_2.ver$, $m_1.VC \leq m_2.VC$, (2) otherwise $m_1.ver \leq m_2.ver$. □

We would like to present a protocol to causally deliver messages while the membership of G is being changed. Each object O_i has a variable ver_i denoting the current version number. Suppose that O_i receives a message m from O_j. There are following three cases: (1) $m.ver > ver_i$. (2) $m.ver < ver_i$. (3) $m.ver = ver_i$.

We would like to consider the first case (1) $m.ver > ver_i$. This means that O_j sends m to O_i after the version of the vector scheme is updated while O_i's version is not updated yet. O_i stores m in the buffer. m is stored in the buffer until ver_i is updated.

The second case (2) $m.ver < ver_i$ means that O_j sends m to O_i before updating the vector clock while O_i has updated the vector clock. Thus, O_i may receive messages with older version numbers than O_i. Here, O_i receives messages from the objects in the new membership. These messages have the same version number as ver_i. Suppose that O_i receives a message m_j from O_j where $m_j.ver = ver_i$. However, O_i does not deliver m_j by the O rule because O_i might still receive messages whose version number is smaller than ver_i. Here, O_i stores m_j in the buffer. If O_i receives a message with the new vector clock scheme from every object in G, the messages stored in the buffer are causally delivered according to the O rule.

7 Concluding Remarks

In this paper, we have presented the group communication protocol for maintaining the membership of the group G and supporting the causally ordered delivery of messages while the membership is being changed. We have adopted the distributed protocol where there is no centralized controller.

References

[1] Raynal, M. and Singhal, M., "Logical time: capturing causality in distributed systems," *IEEE Computer*, Vol. 29, No. 2, 1996, pp. 49–56.

[2] Reiter, M. K., "A Secure Group Membership Protocol," *IEEE Trans. on Software Engineering*, Vol.22, No.1, 1996, pp. 31–42.