Quorum-Based Protocol for Group of Replicas *

1R−9
Mayumi Shindo, Tomoya Enokido, Hiroaki Higaki, and Makoto Takizawa †
Tokyo Denki University ‡
{mayumi,eno,hig,taki}@takilab.k.dendai.ac.jp

1 Introduction

Objects in distributed systems are replicated to make the systems fault-tolerant. Only messages significant for the applications, not all messages transmitted in the network, have to be causally ordered in order to reduce the computation and communication overheads while all messages transmitted in the network are ordered in traditional group protocols. In this paper, the significantly precedent relation among messages is defined in context of request and response messages of *read* and *write* operations on replicas. We discuss a group communication for the replicas where transactions issue *read* and *write* requests according to the quorum-based scheme.

In the quorum-based scheme [3], a *read* request may be sent to one or more than one replica and a *write* request may not be sent to all the replicas. Let $R(o)$ be a set of replicas $o^1, \ldots, o^q$ $(q \geq 1)$ of an object o. The sets of replicas to which *read* and *write* requests are sent are referred to as *read* and *write quorums* Q_r and Q_w of o, respectively. Quorum numbers N_r and N_w represent the numbers of replicas in Q_r and Q_w, respectively. Here, there are constraints, $Q_r \cup Q_w = R$ and $Q_r \cap Q_w \neq \phi$, i.e. $N_r + N_w > q$ and $N_w + N_w > q$. In this paper, we discuss which messages transmitted in the network are required to be causally delivered in the quorum-based scheme.

2 Quorums

2.1 Quorum-based scheme

A transaction T_i sends *read* requests to N_{ar} $(\leq q)$ replicas and *write* requests to N_{aw} $(\leq q)$ replicas of an object o_a in the quorum-based protocol [3]. A read quorum Q_{ar} and a write quorum Q_{aw} are subsets of $R(o_a)$. Here, $Q_{ar} \cup Q_{aw} = R(o_a)$. The transaction T_i sends write requests to the replicas in Q_{aw}. The data of the replicas in Q_{aw} are overwritten by a *write* request. Each time o_a^t in Q_{aw} receives a write request, v_a^t is incremented by one and the data, in o_a^t is overwritten. Here, T_i obtains the maximum version number from a set of v_a^t sent by o_a^t in Q_{aw}.

The transaction T_i has to read from the newest replica, i.e. replica whose version number is the maximum in $R(o)$. Since the *write* requests are sent to not all the replicas, some replicas to which the *write* request is not sent are still obsolete. T_i derives data from a replica o^t whose version number v^t is the maximum in Q_r.

2.2 Object fault

Let k be $N_{ar} + N_{aw} - q$ for an object o_a where q is a number of replicas. This means that every pair of *read* and *write* quorums include at least k (≥ 1) common replicas. As long as fewer number of the replicas than k are faulty, the transactions can continue the computation of o_a. We assume that the replicas suffer from stop-fault and the number of faulty replicas is smaller than or equal to k. If h $(\leq k)$ replicas are detected to be faulty, the quorum numbers N_{ar} and N_{aw} can be reduced because $R(o_a)$ includes $(q - h)$ operational replicas. In this paper, if h $(\leq k)$ replicas are detected to be faulty, N_{ar} and N_{aw} are updated as follows :

$N_{ar} := N_{ar} - h$;
$N_{aw} := N_{aw} - h$;

If h $(\leq k)$ replicas are faulty, at most $(k - h)$ replicas may get faulty out of $(q - h)$ operational replicas. Here, the summation of read and write quorum numbers is required to be larger than $q + k - 2h - 1$.

3 Message Precedency

Each transaction T_i initiated in a computer p_u is given a transaction identifier $tid(T_i)$. The transaction identifier $tid(T_i)$ is given a concatenation of a logical clock value when T_i is initiated and a computer identifier of p_u. The logical clock of p_u is realized by a vector clock. $V = \langle V_1, \ldots, V_n \rangle$ where n is the number of the computers. Initially, each $V_t = 0$ and is used for a computer p_t $(t = 1, \ldots, n)$. On receipt of a message m, by which a vecter clock $m.V = \langle m.V_1, \ldots, m.V_n \rangle$ is carried, from T_j in a computer p_t, p_t manipulates V as follows :

$V_v := \max(V_v, m.V_v)$ for $v = 1, \ldots, n$ $(v \neq t)$;

That is, T_i is initiated after p_u receives a message from another transaction T_j iff $tid(T_i) > tid(T_j)$. Suppose that the vector clocks of T_i and T_j are not comparable. $tid(T_i) > tid(T_j)$ if the identifier of the computer p_u initiating T_i is larger than the identifier of the computer initiating T_j. Therefore, for every pair of different transactions T_i and T_j, either $tid(T_i) > tid(T_j)$ or $tid(T_i) < tid(T_j)$.

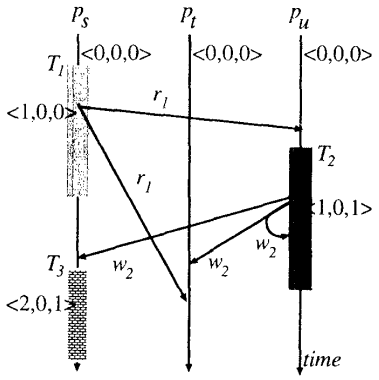

Figure 1: Causal precedence.

Figure 1 shows three computers p_s, p_t, and p_u. Initially, a transaction identifier is $\langle 0, 0, 0 \rangle$ in every computer. A transaction T_1 is initiated in p_s where $tid(T_1) = \langle 1, 0, 0 \rangle$. T_1 issues a read request r_1 to the computers p_t and p_u. Here, $r_1.id = \langle 1, 0, 0 \rangle$. After receiving

*コーラムを用いた因果順序配送プロトコル
†進藤 真由美　榎戸 智也　桧垣 博章　滝沢 誠
‡東京電機大学

r_1, a transaction T_2 is initiated where $tid(T_2) = \langle 1, 0, 1\rangle$. T_2 issues a write request w_2 to the computers p_s, p_t, and p_u. Here, $w_2.id = \langle 1, 0, 1\rangle$. p_t receives r_2 after w_2 since r_1 is delayed while r_1 precedes w_2.

4 Insignificant Messages

4.1 Insignificant messages on the precedency

In the quorum-based protocol, each transaction T_i issues a request message m to one or more than one replicas. Due to the unexpected communication delay in the network, some destination computers may not receive the message m although the other destination computers have received m already. The replicas have to wait for the message m delayed before delivering messages followed by the messages m delayed. The response time and throughput of the system can be improved if messages which need not be delivered are removed from the receipt queue and messages are delivered without waiting for every message causally preceding the messages. We discuss what messages a computer p_t can remove from a receipt queue RQ_t.

[**Definition**] A write request $w_i^t(o_a^t)$ is *current* for a *read* request $r_j^t(o_a^t)$ in a receipt queue RQ_t iff

1. $w_i^t(o_a^t) \Rightarrow_t r_j^t(o_a^t)$ and
2. there is no write request $w_i(o_a)$ such that $w_i^u(o_a^u) \rightarrow_u w_k^u(o_a^u)$ and $w_k^v(o_a^v) \rightarrow_v r_j^v(o_a^v)$. □

The read request $r_j^t(o_a^t)$ is *current* if $w_i^t(o_a^t)$ is current for $r_j^t(o_a^t)$. Otherwise, $r_j^t(o_a^t)$ is *obsolete*. If $r_j^t(o_a^t)$ is current, $r_j^t(o_a^t)$ reads the newest value of the object o_a in the computer p_t. Otherwise, $r_j^t(o_a^t)$ reads the obsolete value of o_a. If $w_i^t(o_a^t)$ is not current for $r_j^t(o_a^t)$, $w_i^t(o_a^t)$ is *obsolete* for $r_j^t(o_a^t)$. In Figure 2, $w_1^u(x)$ is current for $r_2^u(x)$ and $r_2^u(x)$ is current. However, $r_2^v(x)$ is obsolete.

In Figure 2, a computer p_t receives three write requests w_1^t, w_3^t, and w_4^t from the transactions T_1, T_3, and T_4 in this sequence. Other computers p_u, p_v, and p_w receive read and write requests as shown in Figure 2. The computer p_t performs a write request w_1^t before w_3^t, i.e. $w_1^t \rightarrow_t w_3^t$ but w_1^u does not directly precede w_3^u because r_2^u is performed after w_1^u before w_3^u in p_u. The read requests r_2^v and r_5^u are obsolete. w_4^v is current for r_5^v and w_1^u is current for r_2^u.

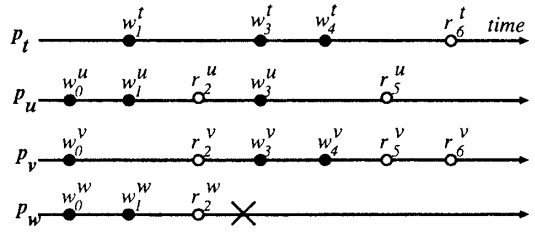

Figure 2: Insignificant requests.

We define *insignificant* messages which can be omitted in a receipt queue RQ_t of a computer p_t.

[**Definition**] A request m is *insignificant* in a receipt queue RQ_t iff one of the following conditions is satisfied,

- $w_i^t(o_a^t) \rightarrow_t w_j^t(o_a^t)$, and there is no read $r_i^k(o_a^t)$ such that $w_i^t(o_a^t) \rightarrow_t r_i^k(o_a^t) \rightarrow_t w_j^t(o_a^t)$ in RQ_t.
- m is a read request for $r_{ji}^t(o_a^t)$, $r_i^t(o_a^t) \rightarrow_t r_j^t(o_a^t)$, and $r_i^t(o_a^t) \rightarrow r_j^t(o_a^t)$ does not hold.
- m is an obsolete write request w_i^t in RQ_t.
- m is an obsolete read request $r_i^t(o_a^t)$ in RQ_t. □

In Figure 2, w_1^t and w_3^t are insignificant in the computer p_t. The values written by w_1^t and w_3^t are overwritten by w_4^t. r_6^v is insignificant in p_v because the requests r_5^v and r_6^v read the value written by w_4^v. Hence, after performing r_5^v, the computer p_v sends the response of r_5^v to not only the transaction T_5 but also T_6. w_3^u is insignificant since w_3^u is obsolete. r_2^v is also insignificant.

Insignificant messages can be removed from the receipt queues. The receipt queues RQ_t, RQ_u, and RQ_v are reduced by removing insignificant messages as shown in Figure 3. Here, r_{56}^v shows a read request where the response of r_5^v is sent to not only T_5 but also T_6. That is, a read request for the replica o_a^v is performed only one time.

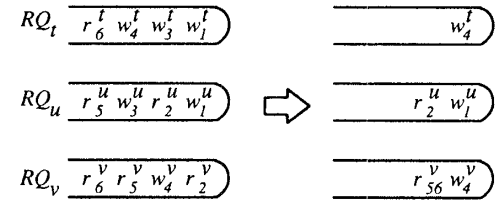

Figure 3: Omission of significant requests.

4.2 Insignificancy on object fault

As discussed in the previous section, some replicas may be faulty, e.g. due to the computer faults. In Figure 2, suppose that $N_{ar} = 3$ and $N_{aw} = 3$. Here, $k = 2$. If the computer p_w is faulty, the quorum numbers are changed to $N_{ar} = 2$ and $N_{aw} = 2$. After p_w gets faulty, a transaction T_3 sends a write request w_3 to p_t, p_u, and p_v by using $N_{aw} = 3$ since T_3 does not know of the fault of p_w. A transaction T_4 sends a read request r_4 to p_u and p_v by using $N_{ar} = 2$ since T_4 knows p_w is faulty. Here, there is no need that T_4 sends w_3 to three replicas o_a^t, o_a^u, and o_a^v in the computers p_t, p_u, and p_v. It is sufficient for T_4 to send w_3 to only two replicas. Hence, one of the three computers p_t, p_u, and p_v, say p_u, is not required to receive w_3. This is because w_3^u is insignificant.

5 Concluding Remarks

This paper has discussed a group protocol for replicas in the quorum-based scheme. A transaction sends read and write requests to one or more than one replica in the quorum-based one while a read request is sent to one replica and a write request is sent to all the replicas in the traditional scheme. We have defined insignificant messages which need not be ordered.

References

[1] Enokido, T., Tachikawa, T., and Takizawa, M., "Transaction-Based Causally Ordered Protocol for Distributed Replicated Objects," *Proc. of IEEE ICPADS'97*, 1997, pp.210–215.

[2] Enokido, T., Higaki, H., and Takizawa, M., "Group Protocol for Distributed Replicated Objects," *Proc. of ICPP'98*, 1998, pp.570–577.

[3] H. Garcia-Molina, H. and D. Barbara, D., "How to Assign Votes in a Distributed System," *Journal of ACM*, Vol.32, No.4, 1985, pp. 841-860.