マルチメディア通信と分散処理 91－14
(1999. 1. 22)

Quorum-Based Protocol for Group of Replicas

Mayumi Shindo, Tomoya Enokido, Hiroaki Higaki, and Makoto Takizawa
Tokyo Denki University
E-mail {mayumi, eno, hig, taki}@takilab.k.dendai.ac.jp

Distributed applications are realized by cooperation of a group of multiple objects. Only messages significant for the applications have to be causally ordered in order to reduce the computation and communication overhead while all messages transmitted in the network are ordered in traditional group protocols. In this paper, significant messages are defined in terms of object concepts. Objects support methods only by which the objects are manipulated. The significantly precedent relation among messages is defined in context of request and response messages. In this paper, we discuss a group protocol for a group of replicas of a simple object which supports read and write operations. Here, transactions issue read and write requests according to the quorum-based scheme.

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
東京電機大学理工学部経営工学科

現在の情報システムは、複数の計算機を相互接続した分散型のシステムとなっている。分散型アプリケーションは、複数のオブジェクトが互いにメッセージを交換し協調動作することで実現される。さらに、各オブジェクトは、システムの信頼性、可用性および性能を向上するため多重化される。このような複数のオブジェクト間の通信において、メッセージの因果順序配送やメッセージ紛失の無い通信を保証するグループ通信プロトコルが多数議論されている。しかし、従来のグループ通信プロトコルの多くは、ネットワークに送信された全てのメッセージに順序付けを行なうため、その通信と処理にかかる負荷が増大してしまう。本論文では、トランザクションが発行する write 演算と read 演算の関係から、順序付け配送すべきメッセージを定義する。これにより、アプリケーションにとって『重要な』メッセージのみを順序付けて配送を行うコーラム方式を用いたグループ通信プロトコルを提案する。

1 Introduction

In order to improve the reliability and availability of the system, objects are replicated in the system. The replicas of the objects are distributed on computers interconnected by communication networks. Transactions manipulate replicas by issuing requests to replicas in servers. The replicas of an object have to be mutually consistent in presence of multiple transactions. The authors [8] discuss a group protocol named OG (object-based group) protocol for a group of multiple abstract objects which support abstract methods. In this paper, we consider a simple object like a file, which supports only basic operations *read* and *write*. A transaction sends a *read* request message to only one replica and sends a *write* request message to all the replicas, i.e. *read-one-write-all* principle. Message transmitted in the network may be unexpectedly delayed due to congestions and message loss in the network. If some message is delayed to be delivered to a replica, the replica is required to wait for the message delayed while messages following the message are received. Raynal *et al.* [1] discuss a group protocol for replicas where some write request delayed can be omitted based on the write-write semantics. The authors [7] present a TBCO (transaction-based causally ordered) protocol which supports the ordered delivery of only messages which are required to be causally ordered from the application's point of view. The delivery order is defined in context of transactions. Here, the TBCO protocol orders only messages which are sent and received in a transaction and which are exchanged among conflicting transactions.

In the read-one-write-all scheme, a read request is not performed if a replica to which the request is destined is faulty. In addition, the read-one-write-all scheme implies larger computation and communication overheads for write dominating applications since *write* requests are sent to all the replicas. In the coterie scheme [10], 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 be a set of replicas $o^1, \ldots, o^m$ ($m \geq 1$) of an object o. The sets of replicas to which *read* and *write* are to be sent are referred to as *read* and *write quorums* Q_r and Q_w of o, respectively. The numbers of replicas in Q_r and Q_w are *quorum* members N_r and N_w of o, respectively. Here, there are constraints, $Q_r \cup Q_w = R$ and $Q_r \cap Q_w \neq \phi$, i.e. $N_r + N_w > m$ and $N_w + N_w > m$. In this paper, we discuss which messages transmitted in the network are required to be causally delivered [11] in the quorum-based scheme. We define *insignificant* messages received by the replica, which can be omitted and need not be ordered.

In sections 2 and 3, we present a system model and message precedency. In section 4, we discuss significant messages. In section 5, we present a quorum-based group (QG) protocol.

2 Quorums

2.1 Quorum-based scheme

A system is composed of multiple computers interconnected by less reliable communication networks. Messages may be lost and the delay time is not bounded in the network. Clients and servers are realized in computers. A server supports clients with objects. An object o supports data and operations *read* and *write* to manipulate the data. The object o is replicated to tolerate the fault of o. Let o^t denote a replica of the object o ($t = 1, \ldots, m$). The replicas are stored in different computers. Let $R(o)$ be a *cluster* of the object o which is a set of the replicas $o^{r_1}, \ldots, o^{r_m}$ of the object o ($m \geq 1$).

A transaction in a client sends *read* and *write* requests to servers to read and write replicas. On receipt of the request from the transaction, the server p_t performs the request on the replica stored in p_t. Here, let $op_i^t(o_a^t)$ denote an operation op issued by a transaction T_i to manipulate a replica o_a^t, where op is either r (read) or w (write).

A transaction T_i sends *read* requests to N_r ($\leq m$) replicas and *write* requests to N_w ($\leq m$) replicas in the quorum-based protocol [10]. The transaction T_i sends write requests to the replicas in the write quorum set Q_w. The data of the replicas in Q_w are overwritten by a *write* request. Here, T_i obtains a version number v^t from a replica o^t which is the maximum in Q_w. v^t is incremented by one. Then, the version numbers of the replicas in Q_w are replaced with v^t.

The transaction T_i has to read the newest replica, i.e. replica whose version number is maximum in the cluster $R(o)$. 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 the read quorum set Q_r.

2.2 Object fault

Let k be $N_r + N_w - m$. It is guaranteed that every pair of *read* and *write* quorum sets 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. We assume that the replicas suffer from stop-fault and the number of faulty replicas is smaller than k. If some number h of replicas are detected to be faulty, the quorum numbers N_r and N_w can be reduced because the cluster $R(o)$ includes ($m - h$) operational replicas. In this paper, if h ($\leq k$) replicas are detected to be faulty, N_r and N_w are updated as follows :

$N_r := N_r - h$:
$N_w := N_w - h$:

The client and server computers detect that h replicas are faulty by using the time-out mechanism. A computer p_u considers a replica o^t to be faulty if p_u had not received any message in fixed time units. If p_u detects that o^t is faulty, p_u includes this information in messages which p_u sends. On receipt of the message from p_u, a computer p_v knows that o^t is faulty. If a computer p_u finds that there are h faulty replicas the p_u changes the *read* and *write* quorum numbers N_w and N_r to $N'_w = N_w - h$ and $N'_r = N_r - h$ since $h \leq k$, $N'_w > 0$ and $N'_r > 0$. Some computer p_v still does not detect the fault while p_u detects the fault. Here, p_v still uses N_r and N_w as the quorum numbers of o while p_u uses N'_r and N'_w. It is straightforward that the following properties hold :

1. $N'_r + N'_w = N_r + N_w - 2h > m + k - 2h$.
2. $N_r + N'_w > m + k - h > m + k - 2h$.
3. $N'_r + N_w > m + k - h > m + k - 2h$.
4. $(N_r - h) + (N_w - h) > m + k - 2h$.
5. $(N_r - h) + N'_w > m + k - \frac{3}{2} h > m + k - 2h$.
6. $N'_r + (N_w - h) > m + k - \frac{3}{2} h > m + k - 2h$.

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

If a faulty replica recovers from the fault, the quorum numbers N_r and N_w are incremented as $N_r := N_r + 1$ and $N_w := N_w + 1$. We assume that all operational computers are synchronized to get the same quorum numbers N_r and N_w if the faulty replicas are recovered.

3 Message Precedency

Each server in a computer stores replicas of objects. Here, let o_a^t denote a replica of an object o_a which is stored in a computer p_t. A transaction T_i in a computer p_u sends a *read* or *write* request m to server computers to manipulate replicas according to the quorum-based scheme. On receipt of a request message m from a transaction T_i, a computer p_t enqueues m into a receipt queue RQ_t. Here, let $m.op$ show an operation op, i.e. r or w and $m.o$ be an object o to be manipulated by op, which are carried by a request message m. Let $m.dst$ be a set of destination computers of m. Let $m.id$ be a transaction identifier of the transaction T_i which sends m. The computer p_t takes a top message m in RQ_t and then performs an operation $m.op$ on a replica $m.o$ stored in p_t, i.e. o^t. Here, let $op_i^t(o_a^t)$ denote an operation op on a replica o_a^t which is issued by a transaction T_i. Let $Q_r(o)$ and $Q_w(o)$ show the quorum sets of an object o, and $N_r(o)$ and $N_w(o)$ indicate the quorum number of o, respectively.

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 the computer p_u. The logical clock of p_u is realized by a vector clock [13]. The logical clock is a vector $V = \langle V_1, \ldots, V_n \rangle$ where n is the number. Initially, each $V_t = 0$ and is used for a computer p_t ($t = 1, \ldots, n$). For a pair of vector clocks $V_1 = \langle V_{11}, \ldots, V_{1n} \rangle$ and $V_2 = \langle V_{21}, \ldots, V_{2n} \rangle$, $V_1 \geq V_2$ if $V_{1t} \geq V_{2t}$ for $t = 1, \ldots, n$. If $V_1 \geq V_2$ or $V_1 \leq V_2$, V_1 and V_2 are *comparable*. Each time a transaction T_i is initiated in a computer p_u, V_u is incremented by one, i.e. $V_u := V_u + 1$. $tid(T_i) := V$. A message m sent by T_i carries the transaction identifier $tid(T_i)$ as $m.id = \langle m.V_1, \ldots, m.V_n \rangle$. On receipt of a message m from T_j in a computer p_t, V is manipulated as follows :

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

That is, if 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, $tid(T_i) > tid(T_j)$ or $tid(T_i) < tid(T_j)$.

A transaction T_i is a sequence of read and write requests. That is, T_i issues serially requests to servers. T_i does not send multiple requests in parallel.

[**Definition**] A request $op_i^t(o_a^t)$ *precedes* another request $op_i^u(o_b^u)$ in a transaction T_i ($op_i^t(o_a^t) \rightarrow_{T_i} op_i^u(o_b^u)$) if T_i issues $op_i^t(o_a^t)$ before $op_i^u(o_b^u)$. □

Each request message m has a sequence number $m.sq$. The sequence number is incremented by one each time p_t sends a message. For every pair of messages m_1 and m_2 sent by a computer p_t, $m_1.sq < m_2.sq$ iff p_t sends m_1 before m_2.

A receipt queue RQ_t of a computer p_t shows a sequence of *read* and *write* requests which p_t receives but does not yet compute. Messages in RQ_t are ordered by the following rule.

[**Ordering rule**] A request m_1 *precedes* another request m_2 in a receipt queue RQ_t of a computer p_t if one of the following conditions holds :

1. $m_1.id < m_2.id$ and $m_1.op$ conflicts with $m_2.op$.
2. $m_1.sq < m_2.sq$ if $m_1.id = m_2.id$, i.e. m_1 and m_2 are sent by a same transaction. □

If a pair of messages m_1 and m_2 cannot be ordered according to the ordering rule, m_1 and m_2 are stored in a receipt order. A pair of read requests m_1 and m_3 sent by different transactions are not ordered according to the ordering rule. The transaction identifier is generated by the vector clock and the computer identifier. Hence, it is straightforward to show that m_1 precedes m_2 in RQ_t if m_1 causally precedes m_2. In addition, $m_1.id > m_2.id$ or $m_1.id < m_2.id$ even if the transaction identifiers of the transactions sending m_1 and m_2 are not comparable. Hence, if $m_1.op$ and $m_2.op$ are write requests on a same replica, m_1 and m_2 are preceded in the same order in every pair of common destination replicas of m_1 and m_2.

[**Properties**] A request m_1 precedes another request m_2 in every destination computer of m_1 and m_2 if

1. m_1 causally precedes m_2.
2. m_1 and m_2 are not causally ordered and $m_1.op$ conflicts with $m_2.op$. □

Figure 1 shows there computer 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. After receiving 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 p_s, p_t, and p_u. p_t receives r_2 after w_2 since r_1 is delayed while r_1 causally precedes w_2. Here, $r_1.id = \langle 1, 0, 0\rangle > w_2.id = \langle 1, 0, 1\rangle$. Here, r_1 precedes w_2 in p_t and also in p_u by the ordering rule. A transaction T_3 is initiated at p_s, $tid(T_3) = \langle 2, 0, 1\rangle$.

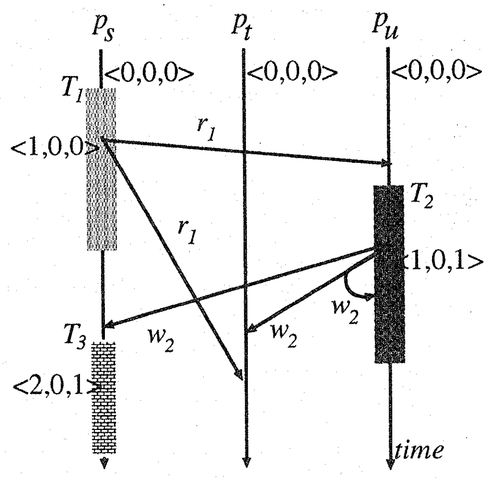

Figure 1: Causal precedence.

4 Insignificant Messages

4.1 Insignificant messages on the precedency

In the quorum-based scheme, each transaction T_i issues a request message m to one or more than one replica. Due to the unexpected communication delay in the network, some destination computer may not receive the message m although the other destination computers have received m already. The response time and throughput of the system can be improved if messages which need not be delivered are removed from the receipt queue. We discuss what messages a receipt queue RQ_t a computer p_t can remove from a receipt queue.

Suppose that there are two requests $op_i^t(o_a^t)$ and $op_j^t(o_b^t)$ in a receipt queue RQ_t of a computer p_t.

[**Definition**] A request $op_i^t(o_a^t)$ *locally precedes* another request $op_j^t(o_a^t)$ in a computer p_t ($op_i^t(o_a^t) \rightarrow_t op_j^t(o_a^t)$) iff $op_i^t(o_a^t)$ precedes $op_j^t(o_a^t)$ in RQ_t and $op_j^t(o_a^t)$ conflicts with $op_i^t(o_a^t)$. □

[**Definition**] A request $op_i^t(o_a^t)$ *precedes* another request $op_j^u(o_b^u)$ ($op_i^t(o_a^t) \rightarrow op_j^u(o_b^u)$) iff $op_i^t(o_a^t) \rightarrow_t op_j^u(o_b^u)$ for $u = t$, $op_i^t(o_a^t) \rightarrow_t op_j^t(o_b^t)$, $op_i^u(o_a^u) \rightarrow_u op_j^u(o_b^u)$, $op_i^t(o_a^t) \rightarrow_{T_i} op_j^u(o_b^u)$ for same transaction T_i and $j = i$, or there is same operation op such that $op_i^t(o_a^t) \rightarrow op \rightarrow op_j^u(o_b^u)$. □

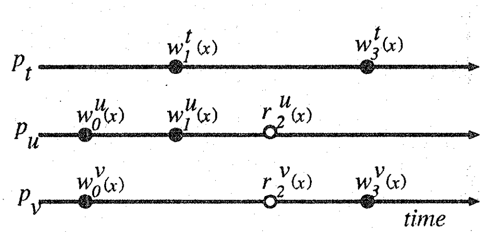

Figure 2: Precedency.

Figure 2 shows three computers p_t, p_u, and p_v each of which supports a replica x. $w_1^t(x) \rightarrow_t w_3^t(x)$, $w_1^u(x) \rightarrow_u r_2^u(x)$, and $r_2^v(x) \rightarrow_v w_3^v(x)$. Since $w_1^t(x) \rightarrow r_2^v(x)$, $w_1^t(x) \rightarrow w_3^v(x)$.

It is important to make clear what value each *read* request reads.

[Definition] A read request $r_j^t(o_a^t)$ *reads* o_a *from* a write request $w_i^t(o_a^t)$ in a computer p_t ($w_i^t(o_a^t) \Rightarrow_t r_j^t(o_a^t)$) iff $w_i^t(o_a^t) \rightarrow_t r_j^t(o_a^t)$ and there is no write request *op* such that $w_i^t(o_a^t) \rightarrow_t op \rightarrow_t r_j^t(o_a^t)$ in RQ_t. □

In Figure 2, $w_1^u \Rightarrow_t r_2^u$. However, $w_0^v \Rightarrow_t r_2^v$ does not hold.

In the quorum-based scheme, only if a *read* request could read the newest version of the replica, the *read* request is meaningful. Otherwise, it is meaningless to perform the read request.

[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. Otherwise, $r_j^t(o_a^t)$ reads the older 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 is current for r_2^u and r_2^u is current. However, r_2^v is obsolete.

According to the *write-write* semantics, if a *write* request w_2 is performed just after a write w_1 on a replica, w_1 is overwritten by w_2 according to the write-write semantics. Hence, w_1 needs not be performed if w_2 is surely performed.

[Definition] A write request $w_i^t(o_a^t)$ *directly precedes* another write request $w_j^u(o_a^u)$ iff $w_i^t(o_a^t) \rightarrow w_j^u(o_a^u)$ and there is no read $r_k^v(o_a^v)$ such that $w_i^t(o_a^t) \rightarrow r_k^v(o_a^v) \rightarrow w_j^u(o_a^u)$. □

In Figure 2, w_1^u directly precedes w_0^u. However,w_3^t does not directly precede w_1^t since $w_1^t \rightarrow r_2^u \rightarrow w_3^t$.

Figure 3 shows there are three computers p_t, p_u, and p_v receiving read and write requests. Here, a notation op_i^s shows a request *op* which is issued by a transaction T_i and is to be performed by a computer p_s. An object o is replicated in p_t, p_u, and p_v. That is, p_t, p_u, and p_v have replicas o^t, o^u, and o^v of an object o, respectively. The read and write quorum numbers for an object o are given as $N_r(o) = N_w(o) = 2$. The 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. The computers p_u and p_v receive read and write requests as shown in Figure 3. 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^t does not directly precede w_3^t because r_2^u is performed after w_1^u before w_3^u in p_u. 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.

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

[WW - rule] If $w_i^t(o_a^t) \rightarrow_t w_j^t(o_a^t)$ 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, $w_i^t(o_a^t)$ is insignificant in RQ_t. □

In Figure 3, w_1^t and w_3^t are insignificant in the computer p_t. The value written by w_1^t and w_3^t are overwritten by w_4^t.

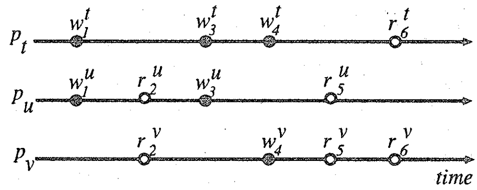

Figure 3: Insignificant requests.

[RR - rule] If $r_i^t(o_a^t) \rightarrow_t r_j^t(o_a^t)$ and $r_i^t(o_a^t) \rightarrow r_j^t(o_a^t)$ does not hold, $r_j^t(o_a^t)$ is insignificant in RQ_t. □

If $r_j^t(o_a^t)$ is insignificant, the response of $r_i^t(o_a^t)$ is sent to the source computer of $r_j^t(o_a^t)$ in addition to the source computer of $r_i^t(o_a^t)$. In Figure 3, 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.

Obsolete requests in RQ_t are also insignificant.

[Obsolete request rule]

1. If $w_i^t(o_a^t)$ is obsolete in RQ_t, $w_i^t(o_a^t)$ is insignificant.
2. If $r_i^t(o_a^t)$ is obsolete in RQ_t, $r_i^t(o_a^t)$ is insignificant. □

In Figure 3, since w_3^u is obsolete, w_3^u is insignificant. 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 4. Here, r_{56}^v shows r_5^v where the response of r_5^v is sent to not only T_5 but also T_6.

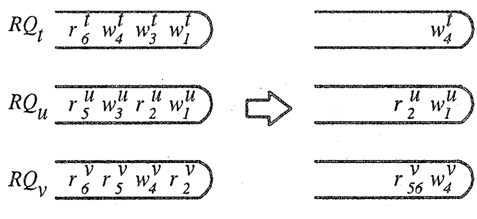

Figure 4: Omission of significant requests.

[Theorem] For every read request $r_i(o)$ on an object o, there is at least one replica o^t of o where a read request $r_i^t(o^t)$ is current. □

Let L_t be a local log of a computer p_t, i.e. a sequence of requests which p_t performs without removing insignificant requests. Let L be a collection $\{L_1, \ldots, L_n\}$ of the local logs. Let L_t' be a local log of p_t obtained from L_t by removing insignificant requests and L' be $\{L_1, \ldots, L_n\}$. For a pair of read request $r_j^t(o_a^t)$ and write request $w_i^t(o_a^t)$, $w_i^t(o_a^t) \rightarrow_L r_j^t(o_a^t)$ if $w_i^t(o_a^t)$ is current for $r_j^t(o_a^t)$ in L. L' is named a *reduced* log of L.

[Theorem] A log L and a reduced log L' of L satisfy the following properties :

1. $w_i^t \rightarrow_L r_j^t$ iff $w_i^t \rightarrow_{L'} r_j^t$ and
2. L and L' includes the same read 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. Suppose that there are four computers p_t, p_u, p_v, and p_w, each of which has a replica of an object o as shown in Figure 5. Suppose that $N_r(o) = 3$ and $N_w(o) = 3$. Here, $k = 2$. If p_w is faulty, the quorum numbers are changed to $N_r(o) = 2$ and $N_w(o) = 2$. 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 $N_w(o) = 3$ 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 $N_r(o) = 2$ since T_4 knows p_w is faulty. Here, there is no need T_4 sends w_3 to three replicas o^t, o^u, and o^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. Hence, w_3^u is insignificant.

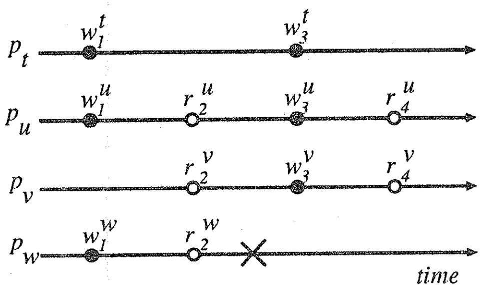

Figure 5: Faulty replica.

Let $Q(op)$ be a quorum set Q_{op} of an operation op and $N(op)$ be a quorum number N_{op} of op. Let $OQ(op)$ be a set of operational replicas in $Q(op)$.

[Redundant write rule] For a write request w_i, $| OQ(w_i) | > N_w(o)$, $| OQ(w_i) | - N_w(o)$ replicas which receive w_i do not perform w_i. □

5 Quorum-based Group Protocol

We present a QG (quorum-based group) protocol for a group of replicas on computers $p_1, \ldots, p_n$. A request message m sent by a transaction T_i at a computer p_t is composed of the following attributes :

$m.sq$ = sequence number of m.

$m.ssq$ = vector of subsequence numbers, $\langle ssq_1, \ldots, ssq_n \rangle$.

$m.ack$ = receipt confirmation $\langle ack_1, \ldots, ack_n \rangle$.

$m.id$ = transaction identifier of T_i, i.e. $tid(T_i) = \langle V_1, \ldots, V_n \rangle$.

$m.op$ = type of operation op, i.e. r or w.

$m.o$ = identifier of object to be manipulated by op.

$m.src$ = identifier of source computer p_t.

$m.dst$ = set of destination computers.

$m.view = \langle f_1, \ldots, f_n \rangle$ where $f_u = 1$ if T_i considers p_u is operational, $f_u = 0$ otherwise for $u = 1, \ldots, n$.

$m.dt$ = data.

Each replica o_a^t has a version vector $V_a^t = \langle V_1, \ldots, V_n \rangle$. The replica o_a^t takes a request m from a receipt queue RQ_t. Then, o_a^t performs op_i^t (= $m.op$) if the following condition is satisfied :

- $m.id > V_a^t$.

Here, o_a^t creates a thread for op_i^t. If op_i^t completes, the version vector V_a^t is changed as follows :

- $V_a^t := m.id$;

Each message m carries a sequence number $m.sq$. Each time a computer p_t sends a message, sq is incremented by one. A message m is sent to some, not all computers. Hence, a message gap cannot be detected by using the sequence number. In order to detect a message gap, a computer p_t manipulates variables $ssq = \langle ssq_1, \ldots, ssq_n \rangle$, $rsq = \langle rsq_1, \ldots, rsq_n \rangle$, and $rq = \langle rq_1, \ldots, rq_n \rangle$. A message carries data of ssq and rsq. Each time p_t sends a message m to a computer p_u, a subsequence number ssq_u is incremented by one and $m.ssq_v := ssq_v$ for $v = 1, \ldots, n$. On the other hand, rsq_u shows a subsequence number ssq_t of message which p_t expects to receive next from p_u ($u = 1, \ldots, n$). Suppose that p_t receives a message m from p_u. If $m.ssq_t = m.rsq_u$, p_t considers that p_t has received every message which p_u had sent before m and $rsq_u := rsq_u + 1$. rq_u is updated as $rq_u := \max(rq_u, m.sq)$.

A message carries receipt confirmation information $ack = \langle ack_1, \ldots, ack_n \rangle$. Each time p_t sends a message m, $m.ack_v := rq_v$ ($v = 1, \ldots, n$). On receipt of m, p_u can know that p_t has received every message from p_s whose sequence number is smaller than $m.ack_s$ ($s = 1, \ldots, n$). p_t manipulates variable $ack = \langle ack_1, \ldots, ack_n \rangle$. On receipt of a message m, $ack_v := \max(ack_v, m.ack_v)$ for $v = 1, \ldots, n$.

[Definition] A message m is *redundant* in a receipt queue RQ_t iff $m.sq < ack_u$ for some destination p_u of m. □

A redundant message m in p_t means that some computer p_u receives m.

[Definition] A request message m is *locally ready* in a receipt queue RQ_t iff every message m' that $m.src = m'.src$ and $m.ssq_t < m'.ssq_t$ is delivered to p_t. □

[Definition] A request message m is *ready* in a receipt queue RQ_t iff m is locally ready in RQ_t and for each computer p_u, there is a locally ready message m' preceded by m in RQ_t. □

If a message m is ready in RQ_t, p_t surely delivers every message preceding m. If a top message m of the receipt queue RQ_t is ready, p_t can take m from RQ_t and deliver m. Here, if the message m is redundant, p_t needs not deliver m in the following cases :

1. If m is a read request, p_t removes m from RQ_t.
2. If m is a write request and m is surely received by more than $N_w(o)$ replicas, p_t removes m from RQ_t.

6 Concluding Remarks

This paper has discussed a group protocol for a replica 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] Ahamad, M., Raynal, M., and Thia-Kime, G., "An Adaptive Protocol for Implementing Causally Consistent Distributed Services," *Proc. of IEEE ICDCS-18*, 1998, pp.86–93.

[2] Bernstein, P. A., Hadzilacos, V., and Goodman, N., "Concurrency Control and Recovery in Database Systems," *Addison-Wesley*, 1987.

[3] Birman, K., Schiper, A., and Stephenson, P., "Lightweight Causal and Atomic Group Multicast," *ACM Trans. Computer Systems*, Vol.9, No.3, 1991, pp.272-314.

[4] Birman, P. K. and Renesse, V. R., "Reliable Distributed Computing with the Isis Toolkit," *IEEE Comp. Society Press*, 1994.

[5] Chandy, K. M. and Lamport, L., "Distributed snapshots: Determining Global States of Distributed Systems," *ACM Trans. on Computer Systems*, Vol.3, No.1, 1985, pp.63–75.

[6] Cheriton, D. R. and Skeen, D., "Understanding the Limitations of Causally and Totally Ordered Communication," *Proc. of ACM SIGOPS'93*, 1993, pp.44–57.

[7] 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.

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

[9] Enokido, T., Higaki, H., and Takizawa, M., "Protocol for Group of Objects," *Proc. of DEXA'98(Lecture Notes in Computer Science, Springer-Verlag,* Vol.1460), 1998, pp.470–479.

[10] 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.

[11] Lamport, L., "Time, Clocks, and the Ordering of Events in a Distributed System," *Comm. ACM*, Vol.21, No.7, 1978, pp.558-565.

[12] Leong, H. V. and Agrawal, D., "Using Message Semantics to Reduce Rollback in Optimistic Message Logging Recovery Schemes," *Proc. of IEEE ICDCS-14*, 1994, pp.227–234.

[13] Mattern, F., "Virtual Time and Global States of Distributed Systems," *Parallel and Distributed Algorithms*, 1989, pp.215-226.

[14] Nakamura, A. and Takizawa, M., "Causally Ordering Broadcast Protocol," *Proc. of IEEE ICDCS-14*, 1994, pp.48–55.

[15] Ravindran, K. and Shah, K., "Causal Broadcasting and Consistency of Distributed Shared Data," *Proc. of IEEE ICDCS-14*, 1994, pp.40-47.

[16] Tachikawa, T. and Takizawa, M., "Significantly Ordered Delivery of Messages in Group Communication," *Computer Communications Journal*, Vol. 20, No.9, 1997, pp. 724–731.

[17] Tachikawa, T., Higaki, H., and Takizawa, M., "Group Communication Protocol for Realtime Applications," *Proc. of IEEE ICDCS-18*, 1998, pp.40–47.

[18] Tanaka, K., Higaki, H., and Takizawa, M. "Object-Based Checkpoints in Distributed Systems," *Journal of Computer Systems Science and Engineering,* Vol. 13, No.3, 1998, pp.125–131.