マルチメディア通信と分散処理 102－31
（２００１．３．２２）

Group Protocol for Quorum-Based Replication

Keijirou Arai, Katsuya Tanaka, and Makoto Takizawa

Tokyo Denki University
E-mail {arai, katsu, taki}@takilab.k.dendai.ac.jp

Distributed applications are realized by cooperation of multiple processes which manipulate data objects like databases. Objects in the systems are replicated to make the systems fault-tolerant. We discuss a system where read and write request messages are issued to replicas in a quorum-based scheme. In this paper, a quorum-based (QB) ordered (QBO) relation among request messages is defined to make the replicas consistent. We discuss a group protocol which supports a group of replicas with the QBO delivery of request messages.

コーラム方式に基づいたグループプロトコル

新井 慶次郎　田中 勝也　滝沢 誠

東京電機大学理工学部情報システム工学科
E-mail {arai, katsu, taki}@takilab.k.dendai.ac.jp

現在の分散型システムは、データベースオブジェクトを操作する複数のプロセスの協調動作によって実現されている。さらに、システム内の各オブジェクトはシステムの信頼性と可用性を向上するために多重化される。本研究では、read と write 要求がコーラム方式に基づき、レプリカの集合に発行されるシステムを考える。本論文では、レプリカ間の一貫性を保証する順序 (コーラム順序:QBO) と、メッセージを配送する方式の提案を行なう。ここでは、必要な要求メッセージのみを配送することにより、システム全体のスループットを向上させるプロトコルの設計と評価について論じている。

1 Introduction

Data objects are replicated in order to increase performance and reliability of a system. In this paper, we consider a simple object like a file, which supports only basic *read* and *write* operations. The replicas of the objects are distributed in data servers. Users in clients initiate transactions in application servers. Transactions manipulate replicas by issuing requests to data servers. The data and application servers are distributed in computers.

A transaction sends a *read* request to one replica and sends a *write* request to all the replicas in order to make the replicas mutually consistent. Another way is the quorum-based scheme [3], where each of *read* and *write* requests are sent to a subset of replicas named *quorum*. Each computer exchanges messages with other computers, where messages are requests issued by transactions and responses from replicas in the computers. It is significant to discuss in which order to deliver the request messages to replicas in each computer.

In the group communications [4, 5, 10–12], a message m_1 *causally precedes* another message m_2 if the sending event of m_1 *happens before* the sending event of m_2 [9]. If m_1 causally precedes m_2, m_1 is required to be delivered before m_2 in every common destination of m_1 and m_2. In addition, write requests issued by different transactions are required to be delivered to replicas in a same order. Thus, the *totally* ordered delivery of messages is also required to be supported in a group of replicas.

Some message m transmitted in the network may be unexpectedly delayed and lost in the network. The replica has to wait for m even if the replica receives messages following m. Raynal *et al.* [1] discuss a group protocol for replicas where write requests delayed can be omitted based on the write-write semantics. The authors [6] present a TBCO (transaction-based causally ordered) protocol where only messages exchanged among conflicting transactions are ordered where objects are not replicated.

Replicas receive read and write requests from transactions in the quorum-based scheme. Suppose a replica receives a write request w_1 and then a read request r. There might be some write request w_2 between w_1 and r, which is not destined to the replica. If there exists w_2, it is meaningless to perform r and w_1 since an obsolete data written by w_1 is read by r. We discuss *significant* messages for each replica which are to be causally/totally ordered in the quorum-based scheme. We present a quorum-based group (QG) protocol which supports a group of replicas with the ordered delivery of significant messages.

In section 2, we present a system model. In section 3, we define a quorum-based precedent relation of messages and we discuss what messages to be ordered. In sections 4 and 5, we present design and evaluation of the QG protocol.

2 System Model

Computers $p_1, \ldots, p_n$ are interconnected in an asynchronous network. That is, messages may be lost and the delay time is not bounded in the network. Replicas of data objects are stored in data servers and transactions in application servers manipulate objects in data servers [Figure 1]. Let o_t denote a replica of an object o in a computer p_t. Let $R(o)$ be a *cluster* of the object o.

A transaction T_i is initiated in an application server and sends *read* and *write* requests to ma-

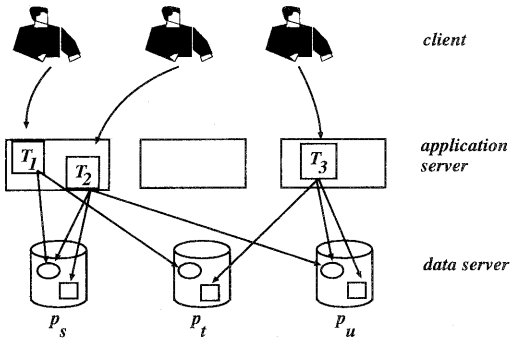

Figure 1: System model.

nipulate a replica o_t in a data server of computer p_t. On receipt of a request op from T_i, op is performed on the replica o_t in the data server of the computer p_t if any operation conflicting with op is being neither performed nor waited. Otherwise, op is waited in the queue. This is realized by the locking protocol. If the replica o_t could be locked, op is performed. Let op_i^t denote an operation op issued by T_i to manipulate a replica o_t in p_t, where op is either r(read) or w(write). The transaction T_i issues either a $commit(c)$ or $abort(a)$ request message to the replicas to which T_i issues *read* and *write* requests. On receipt of c or a request, the lock on o_t held by T_i is released.

A computer is assumed to support a data server and application servers. A computer may send requests issued by a transaction while receiving requests to the server from other computers. Thus, each computer exchanges read and write requests with other computers. In this paper, we discuss in what order request messages received are delivered to replicas in each computer.

A pair of operations op_1 and op_2 on an object are referred to as *conflict* iff op_1 or op_2 is *write*. Otherwise, op_1 and op_2 are *compatible*.

A transaction T_i sends *read* requests to N_r replicas in a read quorum Q_r and *write* to N_w replicas in a write quorum Q_w of an object o. N_r and N_w are *quorum numbers*. $Q_r \subseteq R(o)$, $Q_w \subseteq R(o)$, and $Q_r \cup Q_w = R(o)$ and $N_r + N_w > q$. If a quorum is dynamically constructed each time a request is issued, $N_w + N_w > q$. Each replica o_t has a version number v_t. T_i obtains a version number v_t from a replica o_t which is the maximum Q_w. v_t is incremented by one. Then, the version numbers of the replicas in Q_w are replaced with v_t. T_i reads the replica whose version number is maximum in Q_r. Since $N_r + N_w > q$, every read quorum surely includes at least one newest replica.

3 Precedent Relation of Requests

3.1 Quorum-based precedency

A request message m from a transaction T_i is enqueued into a receipt queue RQ_t in a computer p_t. Here, let $m.op$ show an operation op, i.e. r or w, $m.o$ be an object o to be manipulated by op, $m.dst$ be a set of destination computers, and $m.src$ be the source computer. A top request m in RQ_t is dequeued and then an operation $m.op$ is performed on a replica o_t in p_t where $m.o = o$. RQ_t shows a sequence of *read* and *write* requests received but not yet performed in p_t.

Each computer p_u maintains a vector clock $V = \langle v_1, \ldots, v_n \rangle$ [10] where n is the number of computers. For every 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 neither $V_1 \geq V_2$ nor $V_1 \leq V_2$, V_1 and V_2 are *uncomparable* ($V_1 \parallel V_2$). V is initially $\langle 0, \ldots, 0 \rangle$. Each time a transaction is initiated in p_u, $v_u := v_u + 1$ in p_u. When T_i is initiated, $V(T_i) := V$. A message m sent by T_i carries the vector $m.V = \langle v_1, \ldots, v_n \rangle$ ($= V(T_i)$). On receipt of m from p_u, V is manipulated in p_t as $v_s := \max(v_s, m.v_s)$ for $s = 1, \ldots, n$ ($s \neq t$).

A transaction T_i is given a unique identifier $tid(T_i)$. $tid(T_i)$ is a pair of the vector clock $V(T_i)$ and a computer number $no(T_i)$ of p_u. For a pair of transactions T_i and T_j, $id(T_i) < id(T_j)$ if $V(T_i) < V(T_j)$. If $V(T_i)$ and $V(T_j)$ are uncomparable, $tid(T_i) < tid(T_j)$ if $no(T_i) < no(T_j)$. Hence, for every pair of transactions T_i and T_j, either $tid(T_i) < tid(T_j)$ or $tid(T_i) > tid(T_j)$.

Each request message m has a sequence number $m.sq$. sq is incremented by one in a computer p_t each time p_t sends a message. For each message m sent by a transaction T, $m.tid$ shows $tid(T)$.

[Quorum-based ordering (QBO) rule] A request m_1 *quorum-based* ($Q-$) *precedes* m_2 ($m_1 \prec m_2$) if $m_1.op$ conflicts with $m_2.op$ and

1. $tid(m_1) < tid(m_2)$, or
2. $m_1.sq < m_2.sq$ and $tid(m_1) = tid(m_2)$. □

If $m_1 \prec m_2$, m_1 precedes m_2 in RQ_t. Otherwise, m_1 precedes m_2 in RQ_t if $m_1 \parallel m_2$ and m_1 is received before m_2. Only a pair of conflicting requests m_1 and m_2 are required to be ordered in the same order "$\prec$" in every pair of common destination computers of m_1 and m_2.

[Theorem 1] Let m_1 and m_2 be conflicting requests issued by different transactions.

- $m_1 \prec m_2$ if m_1 causally precedes m_2.
- Otherwise, $m_1 \prec m_2$ if a source computer of m_1 has a larger identifier than m_2. □

4 Significant Messages

Due to unexpected delay and congestions in the network, some destination computer may not receive a message m. The replicas have to wait for m and cannot deliver messages causally/totally preceding m. The response time and throughput can be improved if messages not necessarily to be delivered are removed from the receipt queue and are not waited.

A request m_1 *locally precedes* m_2 in a computer p_t ($m_1 \rightarrow_t m_2$) iff m_1 *Q-precedes* m_2 ($m_1 \prec m_2$) or $m_1 \rightarrow_t m_3 \rightarrow_t m_2$ for some request m_3. m_1 *globally precedes* another request m_2 ($m_1 \rightarrow m_2$) iff $m_1 \rightarrow_t m_2$ or $m_1 \rightarrow_t m_3 \rightarrow m_2$ in some computer p_t.

Figure 2 shows receipt queues of *three* computers p_t, p_u, and p_v, each of which has a replica of an object o. $N_r = N_w = 2$. For example, p_t receives write requests w_1^t, w_3^t, and w_4^t, and then a

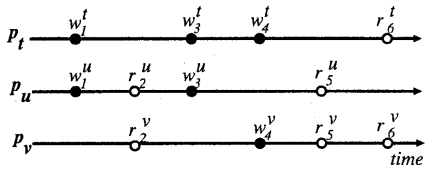

Figure 2: Receipt sequences.

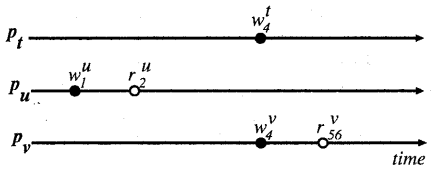

Figure 3: Significant sequences of Figure 2.

read request r_6^t, i.e. $w_1^t \rightarrow_t w_3^t \rightarrow_t w_4^t \rightarrow_t r_6^t$. $w_1^t \rightarrow r_2^u$ since $w_1^u \rightarrow_u r_2^u$. Neither $r_5^v \rightarrow_v r_6^v$ nor $r_6^v \rightarrow_v r_5^v$ since r_5^v and r_6^v are compatible.

A read r_j^t *reads* data *written by* a write w_i^t in p_t ($w_i^t \Rightarrow_t r_j^t$) iff $w_i^t \rightarrow_t r_j^t$ and there is no write w^t such that $w_i^t \rightarrow_t w^t \rightarrow_t r_j^t$.

[Definition] A write request w_i^t is *current* for a *read* request r_j^t in a receipt queue RQ_t iff $w_i^t \Rightarrow_t r_j^t$ and there is no write w such that $w_i^t \rightarrow w \rightarrow r_j^t$. Here, r_j^t is also *current*. □

A request which is not current is *obsolete*. In Figure 2, $w_4^v \Rightarrow_v r_5^v$ and $w_3^u \Rightarrow_u r_5^u$. w_4^v and r_5^v are current but w_3^u and r_5^u are obsolete.

[Definition]

- A write request w_j^t *absorbs* another write request w_i^t if $w_i^t \rightarrow_t w_j^t$ and there is no read r such that $w_j^t \rightarrow_t r \rightarrow_t w_i^t$.
- A current read request r_i^t *absorbs* another read request r_j^t iff $r_i^t \rightarrow_t r_j^t$ and there is no write w such that $r_i^t \rightarrow w \rightarrow r_j^t$. □

In Figure 2, w_3^t absorbs w_1^t and w_4^t absorbs w_3^t and w_1^t. r_5^v absorbs r_6^v. If neither $r_i^t \rightarrow r_j^t$ nor $r_j^t \rightarrow r_i^t$ in p_t, r_i^t and r_j^t read the same data because there is no write request between r_i^t and r_j^t in RQ_t. Hence, data derived by r_i^t can be sent to not only the source computer p_s of r_i^t but also p_v of r_j^t in the response of r_j^t.

[Definition] A request m is *significant* in a receipt queue RQ_t iff m is neither obsolete nor absorbed. □

In Figure 2, r_6^v is current but is absorbed by r_6^v. r_5^v and r_6^v are merged into one read request r_{56}^v which returns the response to the transactions T_5 and T_6. Thus, w_1^t, w_3^t, and r_6^t are insignificant in p_t. r_5^u is insignificant in p_u and r_2^v is also insignificant in p_v. Figure 3 shows a sequence of significant requests for each computer obtained in Figure 2 by removing *insignificant* requests. This sequence is referred to as *significant* sequence. Furthermore, p_t is allowed to deliver messages without waiting for insignificant messages.

5 Group Protocol

5.1 Message format

We present a QG (quorum-based group) protocol for a group of replicas $o_1, \ldots, o_n$ of an object o in computers $p_1, \ldots, p_n$ ($n \geq 1$), respectively. A request message m sent by a transaction T_i in p_t is composed of the following attributes:

$m.op$ = type of operation op, i.e. r or w.
$m.o$ = object o to be manipulated by $m.op$.
$m.src$ = source computer p_t.
$m.tid$ = transaction identifier of T_i.
$m.dst$ = set of destination computers supporting replicas of o in the quorum Q_{op}.
$m.sq$ = sequence number.
$m.SSQ$ = subsequence numbers $\langle ssq_1, \ldots, ssq_n \rangle$.
$m.ACK$ = receipt confirmation $\langle ack_1, \ldots, ack_n \rangle$.
$m.V$ = vector clock, i.e. $V(T_i) = \langle v_1, \ldots, v_n \rangle$.
$m.C$ = write counters $\langle c_1, \ldots, c_n \rangle$.
$m.data$ = data.

Replicas to be in a quorum Q_{op} are randomly selected each time a request op is issued.

5.2 Transmission and receipt

Each message m carries a sequence number $m.sq$. Each time p_t sends a message, sq is incremented by one. Since a message is sent to some computers in the group, a message gap cannot be detected by using the sequence number. In order to detect a message gap, variables $SSQ = \langle ssq_1, \ldots, ssq_n \rangle$, $RSQ = \langle rsq_1, \ldots, rsq_n \rangle$, and $RQ = \langle rq_1, \ldots, rq_n \rangle$ are manipulated in p_t. Each time p_t sends a message m to a computer p_u, a *subsequence number* ssq_u is incremented by one. The message m carries $m.ssq$ where $m.ssq_v := ssq_v$ for $v = 1, \ldots, n$.

The variables rq_u and rsq_u show a sequence number (sq) and a subsequence number (ssq_t) of a message which p_t expects to receive from p_u ($u = 1, \ldots, n$), respectively. Suppose p_t receives a message m from p_s. If $m.ssq_t = m.rsq_s$, p_t has received every message which p_s had sent to p_t before m, i.e. no message gap. Then, $rsq_s := rsq_s + 1$. $rq_s := \max(rq_s, m.sq)$. If $m.ssq_t > rsq_s$, p_t finds p_t has not received some gap message m' from p_s where $m.rsq_s \leq m'.ssq_t < m.ssq_t$. The selective retransmission is adopted. Hence, m is enqueued into the receipt queue RQ_t.

When p_s sends a message m to p_t, $m.ack_v := rq_v$ ($v = 1, \ldots, n$). p_t knows p_s has accepted every message m' from p_u where $m'.sq < m.ack_u$. p_t manipulates the matrix ACK ; $ACK_{su} := m.ack_{su}$ for u=1, ..., n.

[Definition] A message m from p_s is *locally ready* in a receipt queue RQ_t iff $m.ssq_t = 1$ or every message m_1 from p_s in RQ_t such that $m_1.ssq_t < m.ssq_t$ is locally ready. □

A message m received from a computer p_s is locally ready in p_t if $m.ssq_t = rsq_s$. If m is locally ready in RQ_t, p_t receives every message which p_s has sent to p_t before sending m. A message m_1

directly precedes m_2 for a computer p_s in RQ_t ($m_1 \rightarrow_{ts} m_2$) iff $m_1.ssq_t = m_2.ssq_t - 1$.

[Definition] Let m be a message from p_s.

- *partially ready* in RQ_t iff
  1. m is locally ready or
  2. m is a *read* and there is a partially ready message m_1 in RQ_t such that
     - $m_1 \rightarrow_{ts} m$, and
     - $m.op$ is compatible with $m_2.op$, i.e. $m_2.op = r$ for every message m_2 where $m_1.ssq_t < m_2.ssq_t < m.ssq_t$, i.e. p_s sends m_2 to p_t after m_1 before m but p_t does not receive m_2.
- m is *ready* in RQ_t iff
  1. m is locally ready and there is some locally ready message $m_1 (\prec m)$ from every p_u ($\neq p_t$) in RQ_t, or
  2. m is partially ready, and for every p_u ($\neq p_s$), if there is no locally ready message $m_1 (\succ m)$ from p_u in RQ_t, there is a partially ready message $m_2 (\succ m)$ from p_u. □

A message m can be decided to be partially ready according to the following rule:

- A message m from a computer p_s is partially ready in RQ_t if
  1. $m.ssq_t = rsq_s$, i.e. m is locally ready, or
  2. $m.op = r$ and $m_1.c_t = m_2.c_t$ for a pair of requests m_1 and m_2 such that $m_1 \rightarrow_{ts} m \rightarrow_{ts} m_2$.

Suppose p_t receives m_1 from p_s and has received no message from p_s after receiving m_1. Suppose p_t receives m_2 from another computer p_u. If $m_1.sq < m_2.ack_s$, p_t knows p_s has sent some message m_3 such that $m_1.sq < m_3.sq \leq m_2.ack_s$. However, p_t cannot know whether or not m_3 is destined to p_t.

[Definition] A message m from a computer p_s is *uncertain* in RQ_t iff p_t does not receive m, p_t knows that some computer p_u ($\neq p_s$) has received m, i.e. p_t receives such a message m_1 that $m.sq < m_1.ack_s$ from p_u, and p_t does not know whether or not m is destined to p_t. □

5.3 Delivery of requests

Suppose a computer p_t receives a message m. Let m_u denote a message sent by p_u where $m_u \prec m$ and there is no message m'_u from every computer p_u such that $m_u \prec m'_u \prec m$. Let max(m_1, ..., m_n) be a *maximum message* m_v such that $m_s \prec m_v$ for every m_s ($s = 1, \ldots, n$). Here, m_v *directly Q-precedes* m in p_t.

If m is ready in RQ_t, p_t has surely received a partially ready message m'_u from every computer p_u such that $m_u \prec m \prec m'_u$. The messages m_1, ..., m_n are also partially ready. p_t can deliver m after m_1, ..., m_n. Let m'_u be a partially ready message which p_u sends to p_t such that $m_u \prec m \prec m'_u$ and there is no message m''_u from p_u such that $m \prec m''_u \prec m'_u$. If m'_u is locally ready, every message m''_u which p_u sends to p_t after sending m_u before m'_u is not destined to p_t. If m'_u is partially ready but not locally ready, m_u is uncertain. Suppose there are undestined or uncertain messages $u_1, \ldots, u_k$ such that $m_v \prec u_1 \prec \ldots \prec u_k \prec m$ as shown in Figure 4. p_t receives a message m_v (= max(m_1, ..., m_n)) and then receives m but does not receive u_1, ..., u_k. If m is locally ready, the messages u_1, ..., u_k are undestined. If m is partially ready, some message u_i is uncertain. Table 1 summaries how m and m_v are insignificant.

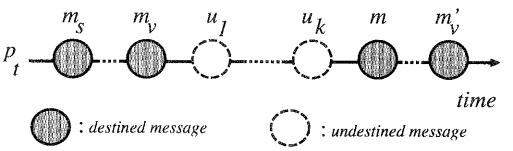

Figure 4: Receipt sequence of messages.

Table 1: Insignificant messages.

m_v	m	$u_1, \ldots, u_k$	Insignificancy
read	read	every u_l is *read*.	m is insignificant (absorbed by m_v). m is merged to m_v.
		some u_l is *write*.	m is insignificant (obsolete).
write	read	every u_l is *read*.	m and m_v are significant.
		some u_l is *write*.	m and m_v are insignificant(obsolete).
read	write		m is not decided.
write	write		m_v is insignificant(obsolete).

If m'_u is not *partially ready*, p_t does not receive a message m''_u from a computer p_u such that $m_u \prec m''_u \prec m \prec m'_u$ and m''_u is destined to p_t. If $m.op$ conflicts with $m''_u.op$, p_t has to wait for a message m''_u such that $m \prec m''_u$ from p_u. If m and m''_u are *read*, p_t can deliver m without waiting for m_u. From the definition, not only m'_u but also every request m''_u is *read* if m'_u is partially ready.

In order to detect insignificant requests in RQ_t, p_t manipulates a vector of *write counters* $C = \langle c_1, \ldots, c_n \rangle$, where each element c_u is initially zero. Suppose p_t sends a message m. If m is a *write* request, $c_u := c_u + 1$ for every destination p_u of m. $m.C := C$. Each message m carries write counters $m.C = \langle m.c_1, \ldots, m.c_n \rangle$. On receipt of a write request m from a computer p_s, $c_u := \max(c_u, m.c_u)(u = 1, \ldots, n)$.

[Theorem 2] Let m_1 and m_2 be messages received by a computer p_t in a receipt queue RQ_t where m_1 precedes m_2. There exists such a write request m_3 that $m_1 \prec m_3 \prec m_2$ if $m_1.C < m_2.C$ and $m_1.V < m_2.V$. □

[Example 1] In Figure 5, each of four computers p_1, p_2, p_3, and p_4 has a replica of an object x and a write counter C is $\langle 0,0,0,0 \rangle$. $N_r = 2$ and $N_w = 3$. p_1 sends a write request w_1 to p_2, p_3, and p_4. and $w_1.C = \langle 0,1,1,1 \rangle$. On receipt of w_1, C is changed to $\langle 0,1,1,1 \rangle$ in p_2, p_3, and p_4. Then, p_2 sends w_2 to p_1, p_2, and p_3. Here, $w_2.C = \langle 0,1,1,1 \rangle + \langle 1,1,1,0 \rangle = \langle 1,2,2,1 \rangle$. Then, p_3 sends r_3 to p_2 and p_4 where $r_3.C = \langle 1,2,2,1 \rangle$ since C is not changed on sending any read request. $r_3.V > w_1.V$ and $r_3.C$ (=

$\langle 1,2,2,1 \rangle) > w_1.C$ $(= \langle 0,1,1,1 \rangle)$. From the theorem, p_4 finds that some undestined write exists between w_1 and r_3. Here, w_1 and r_3 are insignificant in p_4.

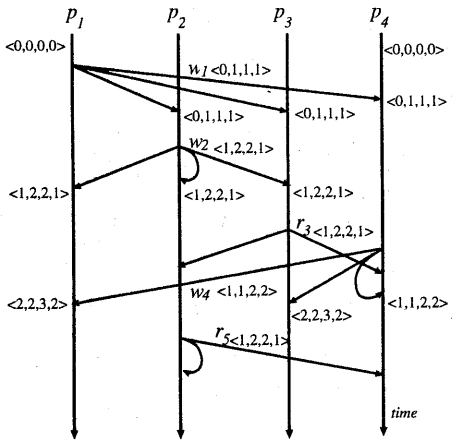

Figure 5: Obsolete messages.

p_2 receives w_2 but not w_4. On the other hand, p_4 receives w_4 but not w_2. Here, $w_2.C = \langle 1,2,2,1 \rangle$, $w_4.C = \langle 1,1,2,2 \rangle$, and $r_5.C = \langle 1,2,2,1 \rangle$. In p_4, $w_4.C \parallel r_5.C$ and w_4 and r_5 are not causally related, $w_4.V \parallel r_5.V$. From the theorem, p_2 cannot decide if an undestined write exists between w_2 and r_5. On the other hand, w_2 causally precedes r_5, i.e. $w_2.V < r_5.V$ and $w_2.C = r_5.C$ $(= \langle 1,2,2,1 \rangle)$. Hence, p_2 considers that no write exists between w_4 and r_5. However, p_4 does not decide if r_5 is obsolete since $r_5.C \parallel w_4.C$. That is, w_2 and r_5 are not ready and have to stay in RQ_4 until p_4 receives a message from p_1 and p_3.□

6 Evaluation

The QG protocol is evaluated by measuring the number of requests performed in each computer and waiting time of each message in a receipt queue through the simulation. We make the following assumptions on the simulation:

[**Assumptions**]

1. Each computer p_t has one replica o_t of an object o $(t = 1, \ldots, n)$. Here, n is a number of computers.
2. Transactions are initiated in each computer p_t. p_t sends one request issued by a transaction every τ time units. τ is a random variable.
3. It takes π time units to perform one request in each computer.
4. p_t randomly decides which replica to be included in a quorum for each request.
5. It takes δ time units to transmit a message from a computer to another. δ is summation of minimal delay time min δ and random variable ϵ.
6. N_r and N_w are quorum numbers for read and write, respectively. $N_r + N_w \geq n + 1$ and $n + 1 \leq 2N_w < n + 2$.
7. It is randomly decided which type *read* or *write* each request is. P_r and P_w are probabilities that a request is read and write, respectively, where $P_r + P_w = 1$. □

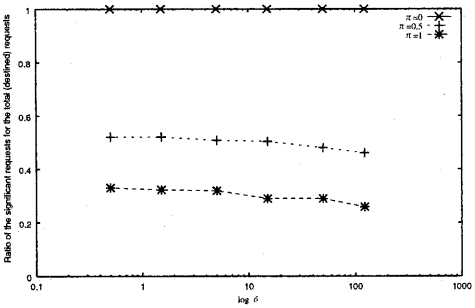

Figure 6: Ratio of significant requests.

Figure 6 shows how many requests are performed in each computer by the QG protocol where $n = 5$, $N_r = N_w = 3$, $P_r = 0.8$, $\tau = 10$ for $\pi = 0$, 0.5, 1[msec]. "$P_r = 80$" means that 80% of messages are read requests. "$\tau = 10$" means that each computer sends a message every 10[msec]. The vertical axis shows what percentage of requests received are significant. The delay times $\delta = 5$. $\delta = 20$, and $\delta = 120$ mean that replicas are distributed in a local area network, in a network of Japan, and in the world, respectively. "$\pi = 0.5$" shows that it takes 0.5[msec] to perform a request. Here, about 50% of the messages transmited are significant. That is, half of the messages received are removed from the receipt queue. For $\pi = 1$, about 30% of the messages are significant. $\pi = 0$ shows a processing speed of each request is so fast that it can be neglected. Here, no message stays in a receipt queue. Every request is performed. In the QG protocol, only the significant messages are delivered. This shows that fewer number of requests are performed, i.e. less computation and communication overheads in the QG protocol than the message-based protocol.

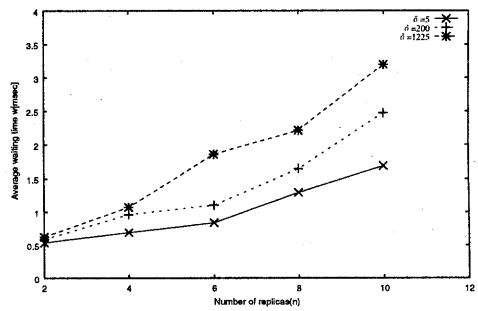

Figure 7: Average waiting time of message.

Figure 7 shows average waiting time w[msec] of message in the receipt queue for number n of

replicas. Here, $P_r = 0.8$, $\tau = 10$[msec], and $\pi = 0.5$[msec]. Here, $N_w = N_w \lceil (n+1) \rceil / 2$. Three cases $\delta = 0.5$, $\delta = 20$, and $\delta = 120$ of average delay time are shown. Figure 7 shows the average waiting time of each message w is $O(n)$ for the number n of computers.

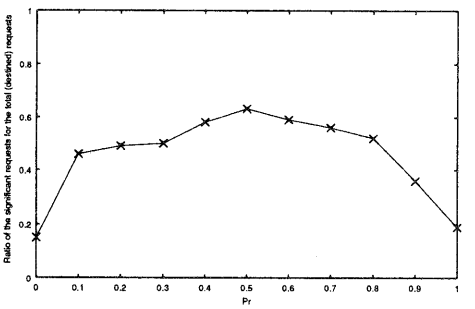

Figure 8: Ratio of read requests(P_r).

Figure 8 shows a ratio of significant messages for P_r. Here, $\pi = 0.5$[msec], $n = 5$, and $N_r = N_w = 3$. "$P_r = 0$" indicates that all the request are *write*. "$P_r = 1$" shows that all the requests are *read*. In case $P_r = 0$ and $P_r = 1$, every request in a receipt queue is *read* and *write*, respectively. In case $P_r = 0$, a last *write* request absorbs every *write* in the queue. In case $P_r = 1$, a top *read* request absorbs every request in the queue. Here, the smallest number of requests are performed. In case "$P_r = 0.5$", the number of insignificant requests removed is the minimum.

7 Concluding Remarks

This paper discussed a group protocol for a group of computer which have replicas where the replicas are manipulated by read and write requests issued by transactions in the quorum-based scheme. We defined the quorum-based ordered (QBO) delivery of messages. We defined significant messages to be ordered for a replica. We presented the QG (quorum-based group) protocol where each replica decides whether or not requests received are significant and which supports the quorum-based ordered (QBO) delivery of messages. The QG protocol delivers request messages without waiting for insignificant messages. We showed that how number of messages to be performed and average waiting time of message in a receipt queue can be reduced in the QG protocol compared with the traditional group protocol.

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] Keijirou, A., Katsuya, T., and Takizawa, M., "Group Protocol for Quorum-Based Replication" *Proc. of IEEE ICPADS'00*, 2000, pp.57–64.

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

[4] 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.

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

[6] 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.

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

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

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

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

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

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

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

[14] 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.