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

分散型応用は、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きている。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、コーラム方式に基づき、read と write 要求がオブジェクトの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、レプリカ間の一慣性を保証する要求メッセージ間の順序 (コーラム順序:QBO) と、この順序にメッセージを配送する方式の提案をする。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

Distributed systems are realized in a 3-tier client server model. Users in clients initiate transactions in application servers. Transactions manipulate objects by issuing requests to data servers. Data and application servers are distributed in computers. Computers which have servers exchange request and response messages on behalf of the servers. Some computer may have both application and data servers. Thus, a collection of computers are exchanging request and response messages. Objects in data savers are replicated in order to increase performance and reliability. In this paper, we consider a system which includes replicas of simple objects like files, which supports basic *read* and *write* operations.

A transaction sends a *read* request to one replica and sends *write* to all the replicas in order to make the replicas mutually consistent in a two-phase locking protocol [3]. The two-phase locking based on read-one-write-all principle is efficient only for read dominating applications. Another way is the quorum-based scheme [3], where each of *read* and *write* requests is sent to a subset of replicas named *quorum*. The more frequently a request is issued, the smaller a quorum is.

In the group communications [4, 9, 10], a message m_1 *causally precedes* another message m_2 if the sending event of m_1 *happens before* m_2 [8]. 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 *write* messages is also required to be supported in a group of replicas. 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 [5] present a transaction-based causally ordered protocol where only messages exchanged among conflicting transactions are ordered where objects are not replicated.

Some message m transmitted in the network may be unexpectedly delayed and lost in the network. Even if messages causally/totally preceded by such a message m are received, the messages cannot be delivered until m is received. Suppose a write request w_1 and then a read request r are issued to a replica. If there exists some write request w_2 between w_1 and r, which is not destined to the replica, it is meaningless to perform r and w_1 since an obsolete data written by w_1 is read by r. Thus, it is critical to discuss what messages to be delivered to replicas in what order. Requests to be delivered are referred to as *significant*. If only significant requests are delivered in each replica, less number of requests are required to be delivered and requests stay in a queue for shorter time. We discuss a group protocol named QG (quorum-based) one where only significant message are delivered. We evaluate the QG protocol in local area network and wide area network like the Internet with respect to how many request messages can be omitted and how long each message waits in a queue.

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 section 4, we present the QG protocol. In section 5, we discuss the evaluation of the QG protocol.

2 System Model

Computers $p_1, \ldots, p_n$ are interconnected in an asynchronous network where messages may be lost and the delay time is not bounded in the network. Applications are realized in a 3-tier client server model. Replicas of data objects are stored in data servers and transactions in application servers issue *read* and *write* requests to data servers to manipulate objects [Figure 1]. Let o_t denote a replica of an object o in a computer p_t. Let $R(o)$ be a *cluster*, i.e. a set of replicas of the object o.

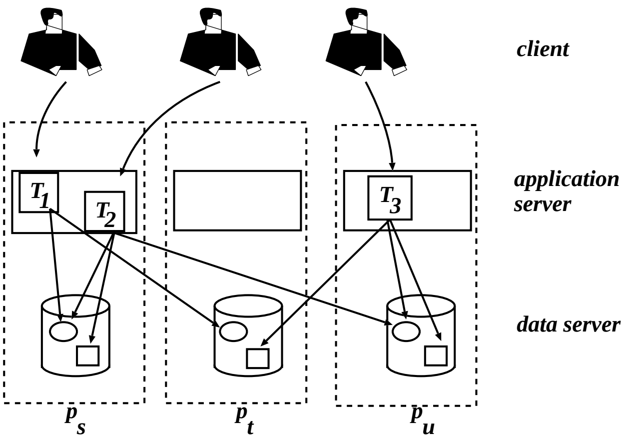


Figure 1: System model.

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*. On receipt of a request op from a transaction T_i, op is performed on the replica o_t in the data server of 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. Let op_i^t denote an instance of 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). After manipulating replica, T_i issues either a $commit(c)$ or $abort(a)$ request message to the replicas. On receipt of c or a request, every lock held by T_i is released.

A computer supports data 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 transaction T_i sends *read* 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*. $Q_r \cup Q_w = R(o)$, $N_r + N_w > q$, and $N_w + N_w > q$. Each replica o_t has a version number v_t. T_i obtains a maximum version number v_t in Q_w. v_t is incremented by one. Then, the version number of every replica in Q_w is replaced with the maximum value v_t. T_i reads the replica whose version number is maximum in Q_r.

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 type op, i.e. r or w. Let $m.o$ be an object o to be manipulated by op, $m.dst$ be a set of destination computers, and $m.src$ show 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 of an object o ($= m.o$) in p_t.

Each computer p_u maintains a vector clock $V = \langle v_1, \ldots, v_n \rangle$ [9]. For every pair of vector clocks $A = \langle a_1, \ldots, a_n \rangle$ and $B = \langle b_1, \ldots, b_n \rangle$, $A \geq B$ if $a_t \geq b_t$ for $t = 1, \ldots, n$. If neither $A \geq B$ nor $A \leq B$, A and B are *uncomparable* ($A \parallel B$). A vector V is initially $\langle$ 0, …, 0 $\rangle$ in every computer. Each time a transaction is initiated in a computer 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 a computer p_t as $v_s := \max(v_s, m.v_s)$ for $s = 1, \ldots, n$ ($s \neq t$).

A transaction T_i initiated in p_u 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 $V(T_i) \parallel V(T_j)$, $tid(T_i) < tid(T_j)$ 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 precedes* ($Q - precedes$) 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 $tid(m_1) = tid(m_2)$. □

$m_1 \parallel m_2$ if neither ($m_1 \prec m_2$) nor ($m_1 \succ m_2$). A pair of messages m_1 and m_2 received by a computer p_t are ordered ($m_1 \rightarrow_t m_2$) in RQ_t:

- If $m_1 \prec m_2$, m_1 precedes m_2 ($m_1 \rightarrow_t m_2$) .
- Otherwise, $m_1 \rightarrow_t m_2$ if $m_1 \parallel m_2$ and m_1 is received before m_2.

"$m_1 \rightarrow_t m_2$" shows "m_1 *locally precedes* m_2 in p_t". m_1 *globally precedes* 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.

3.2 Significant messages

Due to unexpected delay and congestions in the network, some destination computer may not receive a message m. Messages causally/totally preceding m cannot be delivered without receiving m.

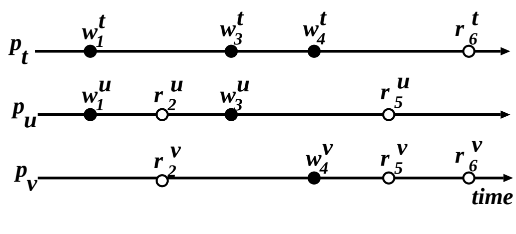


Figure 2: Receipt sequences.

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 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.

If a read request r is performed on a replica o_t, data of o_t written by some write request w is derived by r. Here, it is significant to discuss by what write request data read by a read request is written. 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 $w_i^t \rightarrow_t w^t \rightarrow_t r_j^t$.

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 $w_i^t \rightarrow w \rightarrow r_j^v$. Here, r_j^t is also *current*. A request which is not current is *obsolete*. In addition, if a write w_2 is performed on a replica o_t after w_1 is performed, o_t is overwritten by w_2 and the data written by w_1 disappear.

- 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 $r_i^t \rightarrow w \rightarrow r_j^t$.

[**Definition**] A request m is *significant* in 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.

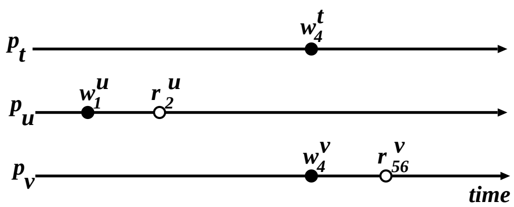


Figure 3: Significant sequences of Figure 2.

4 Group Protocol

4.1 Transmission and receipt

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 quorum Q_{op} is constructed by randomly selecting N_{op} replicas in the cluster $R(o)$ each time a request op is issued. A request message m sent by a transaction T_i in p_t includes the following attributes:

$m.SSQ =$ 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 $\langle c_1, \ldots, c_n \rangle$.

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 p_u, not only sq but also a *subsequence number* ssq_u are incremented by one. The message m carries sq and ssq_v ($v = 1, \ldots, n$).

The variables rq_u and rsq_s show a sequence number (sq) and a subsequence number (ssq_s) of a message which p_t expects to receive from p_u ($s = 1, \ldots, n$), 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.

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 $m'.sq < m.ack_u$. On receipt of m, $ACK_{su} := m.ack_{su}$ for u=1, …, n. 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. m_1 *directly precedes* m_2 for 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 which a computer p_t receives from p_s.

- m is *partially ready* in RQ_t iff
  1. m is locally ready or
  2. $m.op$ = *read* and there is a partially ready message m_1 in RQ_t such that
     - $m_1 \rightarrow_{ts} m$, and
     - $m_2.op = r$ for every message m_2 where $m_1.ssq_t < m_2.ssq_t < m.ssq_t$.
- 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.□

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 p_s is *uncertain* in RQ_t iff p_t does not receive m, p_t knows that some 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 of $p_t \in m.dst$. □

4.2 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 $\max(m_1, \ldots, m_n)$ 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, …, 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, 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.

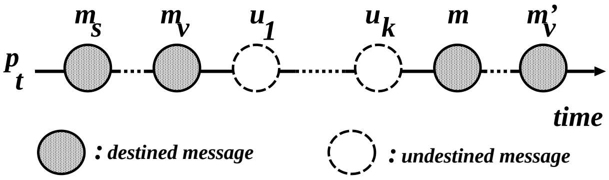


Figure 4: Receipt sequence of messages.

Table 1: Insignificant messages.

m_v	m	u_1, …, 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		if depends on request following m of m_v is significant.
write	write		m_v is insignificant(obsolete).

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**]Let m_1 and m_2 be messages received in a 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 ⟨0,0,0,0⟩. $N_r = 2$ and $N_w = 3$. p_1 sends a write request w_1 to p_2, p_3, and p_4. $w_1.C$ = ⟨0,1,1,1⟩. On receipt of w_1, C = ⟨0,1,1,1⟩ in p_2, p_3, and p_4. Then, p_2 sends w_2 to p_1, p_2, and p_3. Here, $w_2.C$ = ⟨0,1,1,1⟩ + ⟨1,1,1,0⟩ = ⟨1,2,2,1⟩. Then, p_3 sends r_3 to p_2 and p_4 where $r_3.C$ = ⟨1,2,2,1⟩. $r_3.V > w_1.V$ and $r_3.C$ (= ⟨1,2,2,1⟩) $> w_1.C$ (= ⟨0,1,1,1⟩). 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. □

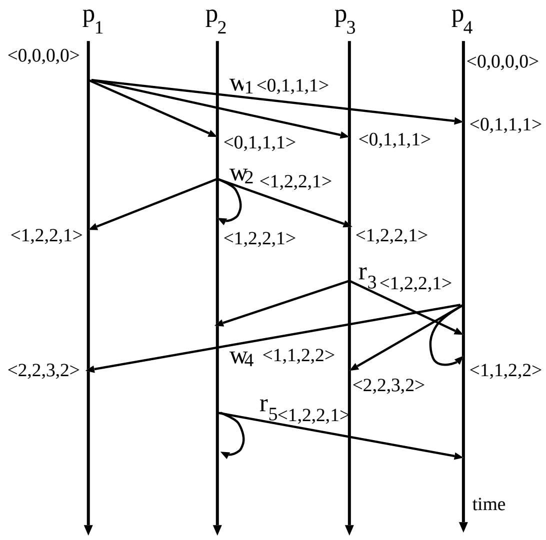


Figure 5: Obsolete messages.

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 $m_1 \rightarrow_{ts} m \rightarrow_{ts} m_2$.

5 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 ($t = 1, \ldots, n$).
2. Transactions are initiated in each computer p_t. Each transaction issues one request, read or write request. A computer p_t sends one request issued every τ time units. τ is a random variable.
3. It takes π time units to perform one request in each computer.
4. N_r and N_w are quorum numbers for read and write, respectively. $N_r + N_w \geq n + 1$ and $n + 1 \leq 2N_w < n + 2$.
5. Each computer p_t randomly decides which replica to be included in a quorum for each request *op* given the quorum number N_{op}.
6. It takes δ time units to transmit a message from one computer to another. δ is summation of minimal delay time $min\delta$ and random variable ϵ.

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 $P_r + P_w = 1$. □

In the QG protocol, only the significant request messages are performed on each replica. If there is at most one request in a receipt queue, all requests which arrive at the computer are performed. Thus, the more number of messages are included in the receipt queue, the more number of messages are not performed since more number of messages can be considered to be insignificant. First, we consider a group of five replicas ($n = 5$) where $N_r = N_w = 3$. We measure the ratio of significant messages (SR) to the total number of messages issued and the average waiting time (W) of each message in a receipt queue. Here, we assure $P_r = 0.8$ and $P_r = 0.2$.

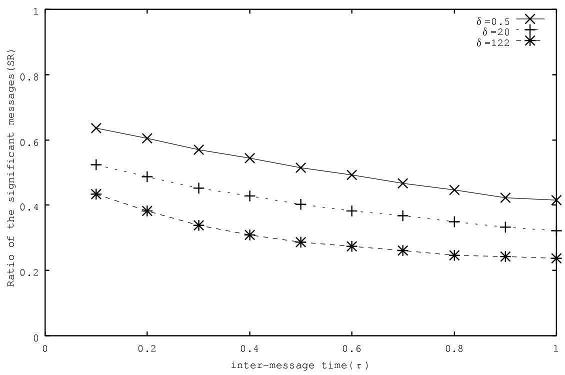


Figure 6: Ratio of significant requests (SR).

Figures 6 and 7 show the ratio of significant messages (SR) and the average waiting time (W) for inter-transaction time τ for $n = 5$. Here, δ shows the delay time. $\delta = 0.5$[msec] means a local area network. $\delta = 20$ shows a nation-wide network, i.e. Japan, and $\delta = 120$ indicates world-wide network. For example, it take about 0.5[msec] to deliver a message from one computer to another in a local area network. It takes about 120[msec] to transmit a message from Japan to the US. In a wide area network, more number of messages are in a transmission. Hence, the larger τ is, the more number of messages arrive at each replica.

In Figure 6 the ratio of significant messages (SR) in the receipt queue is 0.6 for $\tau = 0.2$[msec], This means about 50% of the messages arriving at a computer are considered to be significant in a local area network ($\delta = 0.5$). If each computer sends a message every 0.2[msec] ($\tau = 0.2$), $SR = 0.5$ for $\delta = 0.5$ and $\tau = 0.8$. Only 50% of request messages transmitted in the network are insignificant, i.e. can be omitted in the receipt queue for $\tau = 0.6$ and $\delta = 0.5$, i.e. local area network. In the wide area network ($\delta = 122$), about 70% of request messages can be omitted in the receipt queue for $\tau = 0.6$. Thus, the more number of messages are included in the receipt queue, the more number of messages are not performed.

Figure, 7 shows the average waiting time (W) of the QG protocol for τ. n/τ shows number of messages per msec which a process receives. Here,

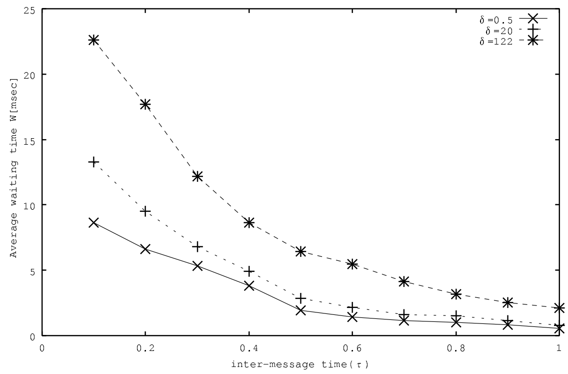


Figure 7: Average waiting time (W).

$n = 5$, the shorter τ is, the more number of messages a process receives.

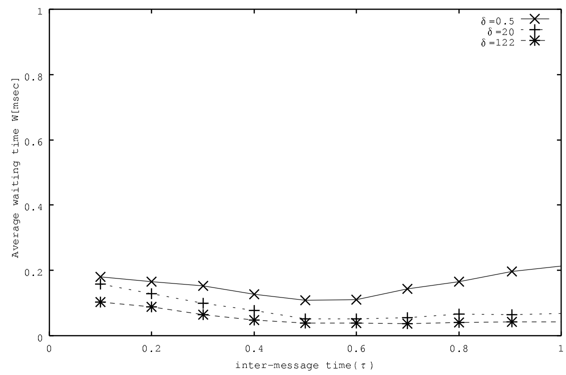
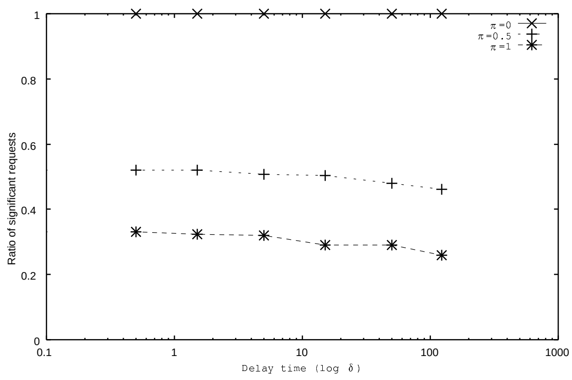


Figure 8: Ratio of average waiting time.

Figure, 8 shows a ratio of the average waiting time of the QG protocol to the traditional group protocol. Figure 9 shows how many requests are performed on 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]. The vertical axis shows what percentage of requests received are significant. Here, about 50% of the messages transmitted 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 10 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, $N_w = N_w = \lceil (n + 1) / 2 \rceil$. Three cases for $\delta = 0.5$, $\delta = 20$, and $\delta = 120$ of average delay time are shown. Figure 10 shows the average waiting time of each message is $O(n)$ for the number n of computers. If messages are kept in the queue according to the traditional protocols, the average waiting time is $O(n^2)$. Thus,



$N_r = 3 \quad N_w = 3$

Figure 9: Ratio of significant requests.

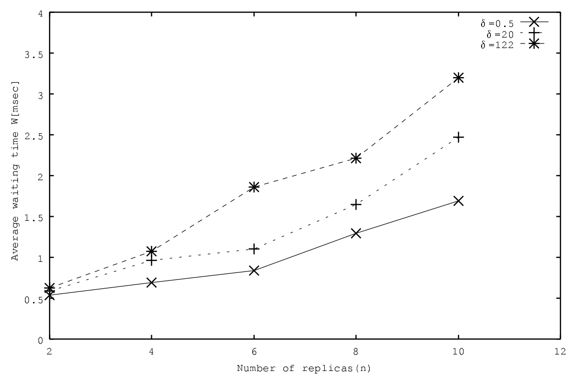


Figure 10: Average waiting time of message.

the average waiting time can be reduced by the QG protocol. Figure 11 shows a ratio of significant messages for P_r. Here, $\pi = 0.5$[msec], $n = 5$, and $N_r = N_w = 3$. In cases $P_r = 0$ 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.

6 Concluding Remarks

This paper discussed a group protocol for a group of computers which have replicas. The replicas are manipulated by read and write requests issued by transactions in the quorum-based scheme. We defined the quorum-based ordered 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 delivery of messages. The QG protocol delivers request messages without waiting for insignificant messages. We showed how many messages to be performed and how long average waiting time of message in a receipt queue can be reduced in the QG protocol compared with the traditional group protocol.

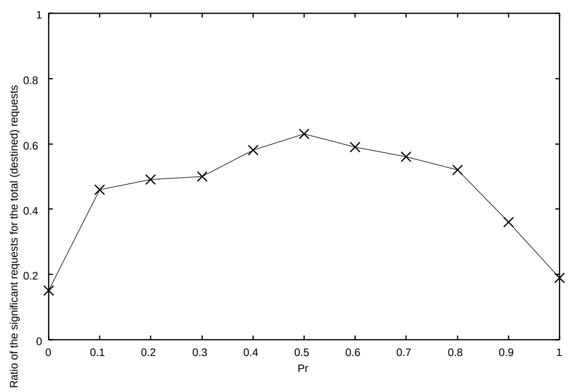


Figure 11: Ratio of read requests(P_r).

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] Arai, K., Tanaka, K., 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] 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.

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

[7] 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.

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

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

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

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

[12] 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.