Group Protocol for Delivering Requests to Replicas *

1G−06

Keijirou Arai, Katsuya Tanaka, and Makoto Takizawa †
Tokyo Denki University ‡

Abstract

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 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.

1 Introduction

Data objects are replicated in order to increase performance and reliability of a system. In this paper, we consider a system which includes replicas of simple objects like files, which supports 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, where each of *read* and *write* requests are sent to a subset of replicas named *quorum*. It is significant to discuss in which order to deliver the request messages to replicas in each computer. In the group communications, 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. In addition, write requests issued by different transactions are required to be delivered to replicas in a same order. Thus, the *totally* ordered delivery for a pair of *write* request messages is also required to be supported in a group of replicas. Requests to be delivered are reformed to as *significant*. The delivery order of significant message which is meaningful for replicas is also *significant*. Compared with the traditional group protocol, loss number of requests are required to be delivered and requests stay for shorter time. We evaluate the QB protocol in environments of a local area network and wide area network like the Internet.

2 Quorums

2.1 Quorum-based scheme

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. Replicas of data objects are stored in data servers and transactions in application servers manipulate objects in data servers. 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. A transaction T_i is initiated in an application server and sends *read* and *write* requests to manipulate a replica o_t in a data server of computer p_t. 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. The quorums are N_r and N_w are *quorum numbers*. $Q_r \subseteq R(o)$, $Q_w \subseteq R(o)$, $Q_r \cup Q_w = R(o)$ and $N_r + N_w > q$, and $N_w + N_w > q$. Each replica o_t has a version number v_t. T_i obtains a 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.

2.2 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, $m.o$ 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 of an object o(=m.o) in p_t. 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 $V = \langle v_1, \ldots, v_n \rangle$ where n is the number of computers. 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* ($A \parallel B$). The vector V is initially $\langle 0, \ldots, 0 \rangle$ 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 $V(T_i)$ 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 $tid(m_1) = tid(m_2)$. □

*コーラム方式に基づいたグループ通信プロトコル
†新井 慶次郎, 田中 勝也, 滝沢 誠
‡東京電機大学

Messages received by a computer p_t are stored in RQ_t and ordered as follows:

- If $m_1 \prec m_2$, m_1 *locally* 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.

$m_1 \rightarrow_t m_2$ shows "m_1 locally precedes m_2 in p_t", i.e. m_1 precedes m_2 in RQ_t. 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. 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**] For every pair of conflicting requests m_1 and m_2, either $m_1 \rightarrow m_2$ or $m_2 \rightarrow m_1$.

[**Theorem 2**] 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. □

3 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. [**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 $w_i^t \rightarrow w \rightarrow r_j^v$. Here, r_j^t is also *current*. □

A request which is not current is *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 $r_i^t \rightarrow w \rightarrow r_j^t$. □

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

4 Group Protocol

4.1 Detection of insignificant requests

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 3**] 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 $m_1.V < m_2.V$. □

5 Evaluation

The QG protocol is evaluated by 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. Each transaction issues are request, read or write request. 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. 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. p_t randomly decides which replica to be included in a quorum for each request given the quorum number.
6. It takes δ time units to transmit a message from a computer to another.
7. It is randomly decided which type *read* or *write* each request is.

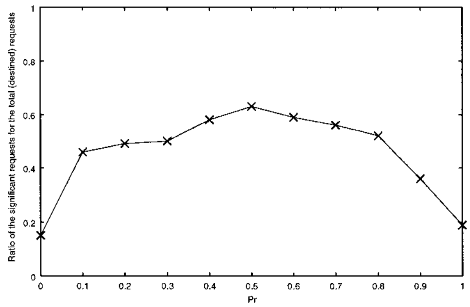

Figure 1: Ratio of read requests(P_r).

Figure 1 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

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. We showed that waiting time of message in a receipt queue can be reduced in the QG protocol.

References

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

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