5ZC-06

Group Protocol for Replicated Objects in Quorum-Based Scheme [*]

Keijirou Arai, Hiroaki Higaki, and Makoto Takizawa [†]
Tokyo Denki University [‡]
Email : {arai, hig, taki}@takilab.k.dendai.ac.jp

1 Introduction

Objects in distributed systems are replicated to make the systems fault-tolerant in order to improve the reliability and availability of the system. The replicas of the objects are distributed on computers interconnected by networks. A server maintains objects in a computer. Transactions in a client manipulate replicas in servers by issuing requests to the servers. In this paper, we consider a group of replicas of a simple object like a file, which supports only basic *read* and *write* operations. A transaction which initiate in a client computer sends a *read* request to only one replica and sends a *write* request to all the replicas. In the group communications, message are causally delivered. That is, a message m_1 *causally precedes* another message m_2 if the sending event of m_1 *happens before* m_2. In addition to the causally ordered delivery, some messages are required to be totally ordered. That is, a message m_1 totally precedes m_2 if m_1 causally precedes m_2. If m_1 and m_2 are not causally ordered, every pair of common destination computers of m_1 and m_2 deliver m_1 and m_2 in the same order. The totally ordered delivery is required to be supported in a group of replicas. The replicas have to be mutually consistent. In the read-one-write-all scheme, larger computation and communication overheads are implied for write dominating applications since *write* requests are sent to all the replicas. In the quorum-based scheme, a *read* request may be sent to more than one replica and a *write* request may not be sent to all the replicas. In this paper, we discuss *insignificant* messages which are received but can be omitted and are not required to be causally and totally ordered in the quorum-based scheme. We discuss a quorum-based group (QG) protocol which supports the ordered delivery of significant messages. In section 2, we define a quorum-based precedency of messages. In section 3, we discuss insignificant messages not to be ordered. In sections 4, the QG protocol is discussed.

2 Quorum-based precedency

2.1 Quorums

Computers p_1, ..., p_n are interconnected in an asynchronous network. Messages may be lost and the delay time is not bounded in the network, i.e. the network is asynchronous. Objects are stored in servers and transactions in clients manipulate the objects in the servers. Clients and servers can be realized in a computer and replicas of an object are stored in servers. Each computer has at most one replica of each object. Let o_a^t denote a replica of an object o_a in a computer p_t. Let $R(o_a)$ be a *cluster* of o_a, which is a set of the replicas $o_a^{r_1}$, ..., $o_a^{r_q}$ of o_a $(q \leq n)$. A transaction T_i sends *read* requests to N_{ar} $(\leq q)$ replicas in a read quorum Q_{ar} of an object o_a and *write* requests to N_{aw} $(\leq q)$ replicas in a write quorum Q_{aw} of an object o_a. N_{ar} and N_{aw} are quorum numbers. $Q_{ar} \subseteq R(o_a)$ and $Q_{aw} \subseteq R(o_a)$. Here, $Q_{ar} \cup Q_{aw}$ $= R(o_a)$ and $N_{ar} + N_{aw} > q$. If a quorum is constructed each time a request is issued, $N_{aw} + N_{aw}$ $> q$. Each replica o_a^t has a version number v_a^t. T_i obtains a version number v_a^t from a replica o_a^t which is the maximum in the write quorum Q_{aw}. v_a^t is incremented by one. Then, the version numbers of the replicas in Q_{aw} are replaced with v_a^t.

2.2 Quorum-based precedency

Each transaction T_i initiated in a computer p_u is given an identifier $tid(T_i) = \langle V(T_i), id(p_u)\rangle$ where $V(T_i)$ is a logical clock of p_u when T_i is initiated. And $id(p_t)$ denote an identifier of p_t. The logical clock of p_u is realized by a vector $V = \langle V_1, \ldots, V_n\rangle$ where each element V_t is initially 0. 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 $V_1 \geq V_2$ or $V_1 \leq V_2$, V_1 and V_2 are *comparable*. Otherwise, V_1 and V_2 are *uncomparable* $(V_1 \parallel V_2)$. On receipt of m from the computer p_u, p_t manipulates the vector V as follows:

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

That is, T_i is initiated in p_u after p_u receives a message from another transaction T_j in p_t iff $V(T_i) > V(T_j)$.

For a pair of transaction identifiers $tid(T_i)$ $(= \langle V(T_i), id(p_u)\rangle)$ and $tid(T_j)$ $(= \langle V(T_j), id(p_t)\rangle)$,

- $tid(T_i) < tid(T_j)$ iff:
  1. $V(T_i) < V(T_j)$, or
  2. $id(p_u) < id(p_t)$ if $V(T_i) \parallel V(T_j)$.

A transaction T_i *precedes* another transaction T_j $(T_i \rightarrow T_j)$ iff $tid(T_i) < tid(T_j)$. Here, it is straightforward either $T_i \rightarrow T_j$ or $T_j \rightarrow T_i$ for every pair of transactions T_i and T_j.

Each request message m has a sequence number $m.sq$. A computer p_t increments the sequence number sq by one each time p_t sends a message. Hence, $m_1.sq$ $< m_2.sq$ iff p_t sends m_1 before m_2. Request messages are preceded as follows. Each request message m has a identifier of source computer $m.src$. Each request message m has a vector clock, i.e. $V(T_i) =$ $\langle V_1, \ldots, V_n\rangle$.
Each request message m has a $m.op$ type of operation op, i.e. r or w.

[Quorum-based ordering (QBO) rule] A request m_1 *quorum-based precedes* (*Q-precedes*) another request m_2 $(m_1 \prec m_2)$ if

1. $m_1.op$ and $m_2.op$ conflict if $m_1.V < m_2.V$,

[*] 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, 桧垣 博章, 滝沢 誠
[‡] 東京電機大学

2. $id(m_1.src) < id(m_2.src)$ and $m_1.op$ and $m_2.op$ conflict if $m.V \parallel m_2.V$, or
3. $m_1.sq < m_2.sq$ if $m_1.V = m_2.V$, i.e. m_1 and m_2 are sent by a same transaction. □

Messages received by a computer p_t are stored in the receipt queue RQ_t. If $m_1 \prec m_2$, m_1 precedes m_2 in RQ_t. If neither $m_1 \prec m_2$ nor $m_2 \prec m_1$, m_1 and m_2 are *Q-uncomparable* ($m_1 \parallel m_2$). Even if a request message m_1 causally precedes another request m_2, $m_1 \prec m_2$ does not hold if $m_1.op$ and $m_2.op$ are compatible according to the QBO rule 1.

3 Insignificant messages

We define insignificant requests. A request m_1 *locally precedes* another request 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. That is, $m_1 \rightarrow_t m_2$ if $m_1.op$ and $m_2.op$ conflict and m_1 precedes m_2 in RQ_t.

[**Definition**] A request message m_1 *globally precedes* another request m_2 ($m_1 \rightarrow m_2$) iff $m_1 \rightarrow_t m_2$, $m_1 \rightarrow_t m_3$ and $m_3 \rightarrow_u m_2$, or $m_1 \rightarrow m_3 \rightarrow m_2$ for some computers p_t and p_u and for some request m_3. □

[**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 request w such that $w_i^t \rightarrow w \rightarrow r_j^v$. Here, r_j^t is also *current*. □ Unless w_i^t and r_j^t are current, w_i^t and r_j^t are referred to as *obsolete*.

[**Definition**] A write request w_i^u *absorbs* w_j^t if w_i^t directly precedes w_j^t, w_i^u directly precedes w_j^u, or w_i^u absorbs w_k^v and w_k^v absorbs w_j^t for some w_k^v. A read request r_i^t *absorbs* another one r_j^t iff $r_i^t \rightarrow_t r_j^t$ and there is no write w_k^t such that $r_i^t \rightarrow w_k^t \rightarrow r_j^t$. □

[**Definition**] A request m is *insignificant* in a receipt queue RQ_t iff (1) m is obsolete or absorbed, or (2) m is a read request and m is current for an insignificant write. □

4 Group Protocol

4.1 Delivery of requests

In order to detect insignificant requests, each replica has to find whether or not each of undestined requests and uncertain ones is *write*. Each message m carries vectors of *write counters* $m.W = \langle m.W_1, \ldots, m.W_n \rangle$. Each computer p_t manipulates variables $W = \langle W_1, \ldots, W_n \rangle$, where each W_u is initially zero. Suppose p_t sends a message m. If m is a *write* request, $W_u := W_u + 1$ for every destination p_u of m. Otherwise, W is not changed. $m.W := W$. On receipt of a write request m from p_s, p_t manipulates the write counter W as follows: On receipt of a write request m from p_s, p_t manipulates the write counter W as follows:

- $W_u := \max(W_u, m.W_u)$ for $u = 1, \ldots, n$;

In the receipt queue RQ_t, messages received are ordered in "$\prec$" according to the QBO rule.

[**Theorem 1**] Let m_1 and m_2 be messages in the receipt queue RQ_t where m_1 directly locally precedes m_2. There is such an undestined write request m_3 that $m_1 \prec m_3 \prec m_2$ in p_t if the following condition holds:

- $m_1.W < m_2.W$ if $m_1.V < m_2.V$.
- $m_1.W < m_2.W$ and $m_1.sq < m_2.sq$ if $m_1.V = m_2.V$.

[**Proof**] First, let us consider a case $m_1.V < m_2.V$. Suppose $m_1.W < m_2.W$ and there is no write request m_3. Since $m_1.V < m_2.V$, m_1 causally precedes m_2. Since $m_1.W < m_2.W$, there must be a *write* request after m_1 before m_2. It contradicts the assumption.

Next, consider a case $m_1.V = m_2.V$. It is clear since m_1 and m_2 are sent by a same computer. □

Here, if m_2 is *read*, m_2 is insignificant because p_t does not receive a write request which m_2 is to read. If m_1 is *write*, m_1 is insignificant. Thus, each computer can decide if each ready request is insignificant.

5 Concluding Remarks

This paper has discussed a group protocol for a group of replicas where the replicas are manipulated in the quorum-based scheme. A transaction sends read and write requests to the quorum number of the replicas. We have defined insignificant messages which need not be ordered for a replica. We have presented the QG (quorum-based group) protocol where each replica decides whether or not requests received are insignificant which supports the QBO delivery of messages. The QG protocol delivers request messages without writing for insignificant message.

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] 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.