3M−02 Quorum-based Locking Protocol for Replicated Objects *

Katsuya Tanaka and Makoto Takizawa †
Tokyo Denki University ‡
e-mail {katsu, taki}@takilab.k.dendai.ac.jp

1 Introduction

Distributed applications are realized in object-based frameworks like CORBA. In order to increase the reliability, availability, and performance, the objects are replicated. In the two-phase locking (2PL) protocol [1], one of the replicas for *read* and all the replicas for *write* are locked. In the quorum-based protocol [2], *quorum* numbers N_r and N_w of the replicas are locked for *read* and *write*, respectively. The subset of the replicas is a *quorum*. Here, a constraint "$N_r + N_w > a$" for the number a of the replicas has to be satisfied. A pair of methods t and u of an object o conflict if the result obtained by performing t and u depends on the computation order. Before performing t, a *quorum* number N_t of the replicas of an object o are locked in a mode of t. Suppose a pair of methods t and u are issued to the replicas. t may be performed on one replica o_t and u on the other o_u if t and u are compatible. Here, the state of o_t is different from o_u. o_t and o_u can be the same if u and t are performed on o_t and o_u, respectively. The authors [3] discuss a version vector to identify which methods are performed on each replica. In this paper, we discuss a simpler method to exchange methods among replicas.

In the object-based system, methods are invoked in a nested manner. Suppose a method a on an object x invokes a method b on another object y. x is replicated in replicas x_1 and x_2 and y is replicated in y_1 and y_2. A method a is issued to x_1 and x_2. Then, the method a invokes b on y_1 and y_2. Here, b is performed twice on each replica. This is a *redundant invocation*. In addition, an instance of a on x_1 issues b to its own quorum, say Q_1, and a on x_2 issues to Q_2 where $|Q_1| = |Q_2| = N_b$. Since $|Q_1 \cup Q_2|$ is larger than the quorum number N_b, more number of replicas are locked than the quorum number N_b. This is a *quorum explosion*. We discuss how to resolve the redundant invocations and quorum explosions in nested invocations of methods on replicas.

In section 2, we present a system model. In section 3, we extend the quorum concepts to the object-based system. In section 4, we discuss how to invoke methods on replicas in a nested manner.

2 System Model

A system is composed of replicas of objects. The replicas are distributed in multiple computers. Each object supports a collection of methods only by which the object is manipulated. A transaction issues a method request *op* to an object o. Then, *op* is performed on the object o and the response of *op* is sent back to the transaction. Let $op(s)$ denote a state obtained by performing a method *op* on a state s of an object o. A method op_t is *compatible* with op_u iff $op_t \circ op_u\ (s) = op_u \circ op_t\ (s)$ for every state s of o, i.e. the result obtained by performing op_t and op_u on o is independent of the computation order of op_t and op_u. Otherwise, op_t *conflicts* with op_u. The conflicting relation is symmetric but not transitive. The method *op* performed on o may furthermore issue a request to another object. Thus, methods are invoked on objects in a nested manner.

A *cluster* R_o of an object o is a set of replicas $\{o_1, \ldots, o_a\}$ $(a \geq 1)$. Suppose there are two objects x and y in the system. There are three replicas, x_1, x_2, and x_3 for the object x and two replicas y_1 and y_2 for y. The object x supports a method t which invokes a method u on the object y. A transaction T issues a request t to replicas of x. In the famous two-phase locking (2PL) protocol, a transaction T issues a *write* request to all the replicas, x_1, x_2, and x_3 but a *read* request to one replica, say x_1. In the quorum-based protocol, T issues *write* and *read* requests to quorum numbers N_w and N_r of replicas of x, respectively. Here, $N_w + N_r > a$ where a is the total number of the replicas of x, i.e. $a = 3$. For example, a *write* request is issued to a subset $\{x_1, x_2\}$ denoted a *write* quorum Q_w and a *read* request is issued to a subset $\{x_2, x_3\}$ denoted a read quorum Q_r. $N_w\ (= |Q_w|) = N_r\ (= |Q_r|) = 2$. The *read* and *write* methods are surely performed on the replica x_2 in $Q_r \cap Q_w$ while only *write* and *read* are performed on x_1 and x_3, respectively.

3 Object Quorums

3.1 Quorum constraint

Let $N_t\ (= |Q_t|)$ be the *quorum* number of op_t. The quorums have to satisfy the following constraint.

[OBQ constraint]

- $N_t + N_u > a$ iff op_t conflicts with op_u. □

In the quorum-based protocol, $N_t + N_u > a$ if a pair of methods op_t and op_u are update ones. On the other hand, the OBQ constraint means that $N_t + N_u > a$ only if op_t conflicts with op_u.

3.2 Exchanging procedure

Each replica o_h has a log L_h where a sequence of update methods performed on o_h are stored. Initially, L_h is empty. Suppose that a method *op* is issued to o_h. Here, let L_h be a sequence of update methods $\langle op_{h1}, \ldots, op_{hm}]$. If *op* is compatible with every method op_{hi}, *op* is enqueued into L_h, i.e. $L_h = \langle op_{h1}, \ldots, op_{hm}, op]$ and *op* is performed on o_h. Suppose that *op* conflicts with a method op_{ht} and *op* is compatible with every method op_{hf} $(f > t)$ in L_h. There might be some replica o_k whose log L_k includes some method op_{kj} which is compatible with a method in L_h but conflicts with *op*, and is not performed on o_h. Such method op_{kj} is required to be performed on o_h before *op* is performed. Here, another replica o_k has a log $L_k = \langle op_{k1}, \ldots, op_{kl}]$ and *op* is issued to o_k. *op* conflicts with op_{ku} and *op* is compatible with every method op_{kf} $(f > u)$. According to the OBQ property, every pair of

---

*レプリカ間の一貫性を保証するロック手法の提案
†田中 勝也　滝沢 誠
‡東京電機大学

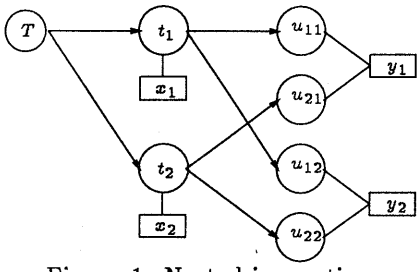

Figure 1: Nested invocation

methods op_{hi} in L_h and op_{kj} in L_k are compatible. Here, a method op_{kj} in L_k is referred to as *missing method* for a method op on o_h iff op_{kj} is not performed on o_h and op_{kj} conflicts with op. Here, every missing method op_{ki} for op in L_k is required to be performed on o_h before op is performed. Then, op is performed on o_h. All the methods conflicting with op are marked in L_h and op is enqueued into L_h. If an update method op is marked in every log, op is performed on every replica and some conflicting method is performed after op. Hence, op is removed from every log.

A transaction T issues a request op to the replicas in a quorum Q_{op}.

1. On receipt of op, a log L_h of a replica o_h is searched. If every method in L_h is compatible with op, op is enqueued into L_h and op is performed on o_h.
2. If there is a some method in L_h which conflicts with op, a replica o_h sends a log L_h to T.
3. T collects the logs from the replicas, i.e. $L = \cup\{L_h \mid o_h \in Q_h\}$. T sends a log $L_h' = \{op' \mid op' \in L - L_h$ and op' conflicts with $op\}$ to the replica o_h. A method in L_h' is performed on o_h. Then, op is performed on o_h. Every method conflicting with op in L_h is marked.

4 Nested Invocation

Suppose that there are replicas $x_1, \ldots, x_a$ of the object x and replicas $y_1, \ldots, y_b$ of the object y. A transaction T issues a method t to replicas in the quorum Q_t, say $N_t = 2$. Suppose t is issued to replicas x_1 and x_2. Furthermore, t issues a request to replicas in the quorum of y to invoke a method u. Here, suppose $N_u = 2$. Let t_1 and t_2 be instances of the method t performed on replicas x_1 and x_2, respectively. Each of t_1 and t_2 issues a request of the method u to replicas in a quorum of t. Here, let Q_{u1} and Q_{u2} be quorums for t_1 and t_2, respectively. Suppose $Q_{u1} = Q_{u2} = \{y_1, y_2\}$. t_1 and t_2 issue u to y_1 and y_2. Here, let u_{i1} and u_{i2} be instances of the method u performed on replicas y_1 and y_2, which are issued by the instance t_i ($i = 1, 2$), respectively. If u changes a state of y, a state of y_1 is inconsistent because two instances u_{11} and u_{21} of u from t_1 and t_2 are performed on y_1 [Figure 1]. This is a *redundant invocation.*

In order to resolve the redundant invocation, the following strategies are adopted:

1. Each transaction T is identified by a unique transaction identifier *tid*. Each request issued in T carries the *tid* of T.
2. If a method op_t is performed on a replica o_h, op_t and the response with *tid* of the transaction issuing op_t is logged into the log.
3. If op_t is issued to o_h and op_t of the same transaction is found in the log of o_h, the response of op_t stored in the log is sent back without performing op_t.

Next suppose $Q_{u1} \neq Q_{u2}$, say $Q_{u1} = \{y_1, y_2\}$ and

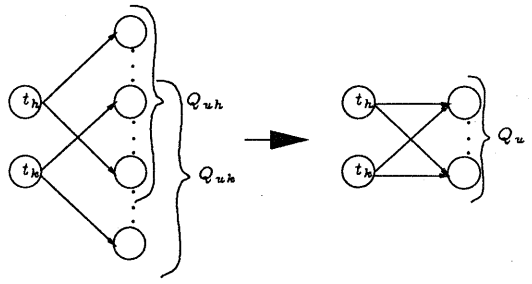

Figure 2: Resolution of quorum explosion.

$Q_{u2} = \{y_2, y_3\}$. The method u is performed on replicas in $Q = Q_{u1} \cup Q_{u2} = \{y_1, y_2, y_3\}$ where u is performed twice on the replicas in $Q_{u1} \cap Q_{u2} = \{y_2\}$. If another transaction manipulates the object y by the method u, u is issued to the replicas in the quorum Q_u, say $\{y_3, y_4\}$. $|Q_{u1} \cup Q_{u2}| \geq |Q_u|$. This means that more replicas are locked than the quorum number N_u of the method u if the method u is invoked by t_1 and t_2. Then, the instances of u on the replicas in $Q_{u1} \cup Q_{u2}$ issue further requests to other replicas and more replicas are locked. This is *quorum explosion.*

Suppose that a method t on an object x invokes a method u on an object y. In order to resolve the quorum explosion, Q_{uh} and Q_{uk} have to be the same for every pair of replicas x_h and x_k where an instance of t is performed. There is a following approach to resolving the quorum explosion:

1. Each replica has a same function *rand* for generating a sequence of random numbers. That is, *rand*(i, n, a) gives a same set of numbers from 1 to a for a same initial value i and the total number n of random numbers to be generated.
2. Each replica x_h gets I = *rand*(*tid*, N_h, a), where *tid* is a transaction identifier of t, N_h is the quorum number of a method u, and a is a total number of replicas. I is a set of replica identifiers. Then, a quorum Q_h is constructed as $Q_h = \{y_i \mid i \in I\}$.

Every instance invoked by a same instance has the same transaction identifier. Thus, every instance of the method d issues a request of the method u to the same quorum. Hence, the quorum explosion is prevented [Figure 2].

5 Concluding Remarks

This paper discussed how multiple transactions invoke methods on replicas of objects. The replicas are not required to perform every update method instance which has been computed on the other replicas if the instance is compatible with the instances performed. We discussed how to resolve redundant invocations and quorum explosions to occur in systems where methods are invoked on multiple replicas in a nested manner.

Reference

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

[2] Garcia-Molina, H. and Barbara, D., "How to Assign Votes in a Distributed System," *JACM*, Vol 32, No.4, 1985, pp. 841-860.

[3] Tanaka, K., Hasegawa, K., and Takizawa, M., "Quorum-Based Replication in Object-Based Systems," *Journal of Information Science and Engineering* (*JISE*), Vol. 16, 2000, pp. 317–331.