Optimization of Nested Invocation on Replicas in Object-based Systems

Katsuya Tanaka and Makoto Takizawa

Tokyo Denki University
Email {katsu, taki}@takilab.k.dendai.ac.jp

An object-based system is composed of multiple objects which are encapsulation of data and methods. Objects are replicated in order to increase reliability and throughput. If a method t is invoked on multiple replicas and each instance of t invokes another update method u, u is performed multiple times on replicas and then the replicas get inconsistent, i.e. redundant invocations. In addition, since an instance of a method on each replica issues a request to its own quorum, more number of the replicas are manipulated than the quorum number, i.e. quorum explosion. We discuss a protocol named QB (quorum-based) one to resolve the redundant invocations and quorum explosion. We show how many replicas manipulated and requests issued are reduced by the QB protocol.

オブジェクトの多重化環境におけるメソッドの入れ子呼出し方式の最適化

田中 勝也　滝沢 誠

東京電機大学

E-mail {katsu, taki}@takilab.k.dendai.ac.jp

高信頼分散オブジェクトシステムでは、複数のレプリカに多重化されたオブジェクトが、メソッドの双方向呼出しにより協調動作を行なう。ここで、各メソッドは、呼び出されたメソッドが他のメソッドを呼び出す、すなわち入れ子型として実現されている。仮に、トランザクションが、複数のレプリカに対して、メソッドtの実行を要求し、メソッドtの各インスタンスが、複数のレプリカに対して更新メソッドuの実行を要求するとする。このとき、uのインスタンスによる更新処理が、同一のレプリカ上で複数回実行される場合がある。これを冗長呼び出しと呼ぶ。加えて、uを発行するレプリカの集合、すなわちuのコーラムが、レプリカ間で異っている場合、必要数以上のレプリカがロックされてしまう。これを、コーラム破裂と呼ぶ。本研究では、こうしたメソッドの入れ子呼び出しにおいて発生する問題について議論し、その解決法を示す。

1 Introduction

Objects are replicated in order to increase the reliability, availability, and performance in distributed object-based applications [12]. In the two-phase locking (2PL) protocol [3,7], one of the replicas for *read* and all the replicas for *write* are locked. In the quorum-based protocol [8], *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. An object is an encapsulation of data and methods for manipulating the data. A pair of methods conflict on an object if the result obtained by performing the methods depends on the computation order. In the papers [13,15], the quorum concept for read and write is extended to objects supporting more abstract level of methods. Suppose a pair of update methods t and u are issued to replicas x_1 and x_2 of an object x. The method t may be performed on one replica x_1 and the other method u on another replica x_2 if t and u are compatible. Here, the state of x_1 is different from x_2. The newest versions of x_1 and x_2 can be obtained by performing methods performed on the other replica, i.e. u is performed on x_1 and t is performed on x_2. As long as t and u are issued, the methods are performed on replicas in their quorums. If some method v conflicting with t is issued to a replica x_1, every instance of t not performed so far is required to be performed on x_1. "$N_t + N_u > a$" if t and u conflict. Even if a replica is updated by t or u, i.e. *write*, $N_t + N_u \leq a$ only if t and u are compatible.

In the object-based system, methods are invoked in a nested manner. Suppose a method t on an object x invokes a method u on another object y. Let x_1 and x_2 be replicas of the object x. Let y_1 and y_2 be replicas of y. A method t is issued to the replicas x_1 and x_2. We assume that every method is deterministic. That is, the same computation for t is done on x_1 and x_2. Then, the method t invokes the other method u on y_1 and y_2. Here, u is performed twice on each replica although u should be performed only once on each of the replicas y_1 and y_2 if the replicas are updated by u. Otherwise, y gets inconsistent. This is a *redundant invocation*. In addition, an instance of the method t on x_1 issues a method u to replicas in its own quorum Q_1, and another instance of t on x_2 issues a method u to replicas in Q_2 where $|Q_1| = |Q_2| = N_u$, but $Q_1 \neq Q_2$. More number of replicas are manipulated than the quorum number N_u, i.e. $|Q_1 \cup Q_2| \geq N_u$. If the method u furthermore invokes another method, the number of replicas to be manipulated is more increased and eventually all the replicas are manipulated. 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 overview the quorum-based protocol. In section 3, we discuss how to resolve the redundant invocation of methods on replicas. In section 4, we discuss how to resolve the quorum explosion. In section 5, we evaluate the quorum-based protocol in terms of number of replicas to be manipulated and number of requests to be issued.

2 Quorum-based Replication

2.1 Traditional quorum concept

A transaction invokes a method t by issuing a request t to an object x. Then, the method t is performed on the object x and then the response is sent back to the transaction. In this paper, we assume each method is synchronously invoked. Here, the method t may invoke other methods, i.e. nested invocation. A method t is *compatible* with a method u iff the result obtained by performing the methods t and u on the object x is independent of the computation order. Otherwise, t *conflicts* with u. We assume the conflicting relation is symmetric but not transitive.

Suppose there are three replicas x_1, x_2, and x_3 of an object x which supports a method t. In the quorum-based protocol [8], a transaction issues *write* and *read* requests to subsets of the replicas named *quorums* including some numbers N_w and N_r of replicas of x, respectively. Here, $N_w + N_r > 3$. For example, a *write* request is issued to replicas in a quorum $Q_w = \{x_1, x_2\}$ and a *read* request is issued to a quorum $Q_r = \{x_2, x_3\}$. 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. Each replica x_i has a version number b_i. b_i is incremented by one each time *write* is performed on x_i. If a request t is issued, a replica x_i whose version number b_i is maximum in a quorum Q_t is found. If the request is *write* with a value v, v is written into the replica x_i. The version number b_i is incremented by one. Then, b_i is sent to the replicas in Q_w. If the request is *read*, the value v_i of x_i is derived.

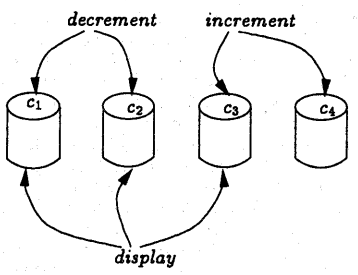

Figure 1: Object-based quorums.

2.2 Object-based quorum

An object is an encapsulation of data and procedures for manipulating the data. The object is allowed to be manipulated only through procedures named *methods*. Traditional quorum-based locking protocols for primitive methods *read* and *write* on simple files are extended to methods of objects [13, 15].

Let us consider a *counter* object c which supports three types of methods *increment* (*inc*), *decrement* (*dec*), and *display* (*dsp*). Suppose there are four replicas c_1, c_2, c_3, and c_4 of the *counter* object c, i.e. the cluster R_c is $\{c_1, c_2, c_3, c_4\}$. According to the traditional quorum-based theory, *inc* and *dec* are considered to be *write* methods because they change the state of the object c. Hence, $N_{inc} + N_{dec} > 4$, $N_{dsp} + N_{inc} > 4$, and $N_{dsp} + N_{dec} > 4$. For example, $N_{inc} = N_{dec} = 3$ and $N_{dsp} = 2$. Since *dsp* conflicts with *inc* and *dec*, $N_{dsp} + N_{inc} > 4$ and $N_{dsp} + N_{dec} > 4$ in our protocol [Figure 1]. On the other hand, *inc* and *dec* are considered to be compatible because the state obtained by performing *inc* and *dec* is independent of the computation order.

Quorums of an object x have to satisfy the following constraint.

[Object-based Quorum (OBQ) constraint] If a pair of methods t and u conflict, $N_t + N_u > a$ where a is the total number of the replicas. □

It is noted that $N_t + N_u \leq a$ if t and u are compatible even if t or u is update. The OBQ constraint implies the following properties:

[Property] Every pair of conflicting methods t and u of an object x are performed on at least k $(= N_t + N_u - a)$ replicas in the same order. □

Hence, $N_{inc} + N_{dec} \leq 4$, e.g. $N_{inc} = N_{dec} = 2$. Suppose $Q_{inc} = \{c_1, c_2\}$ and $Q_{dec} = \{c_3, c_4\}$. Since either *inc* or *dec* is performed on replicas in the quorums, the states of c_1 and c_3 are different. However, if *dec* is performed on c_1 and c_2 and *inc* on c_3 and c_4 here, the states of c_1, c_2, c_3, and c_4 can be the same. This is an *exchanging procedure* where every method t performed on one replica is sent to other replicas where the method t is not performed and only methods compatible with the method t are performed. As long as only compatible methods *inc* and *dec* are issued to the replicas, the exchanging procedure is not required to be executed. Suppose *dsp* is issued to three replicas c_1, c_2, and c_3 where $Q_{dsp} = \{c_1, c_2, c_3\}$. *dsp* conflicts with *inc* and *dec*. *dsp* cannot be performed on replicas c_1, c_2, and c_3 because only *inc* is performed on c_1 and c_2 and only *dec* on c_3. Before performing *dsp*, *dec* has to be performed on c_1 and c_2 and *inc* on c_3. *inc* and *dec* can be performed in any order because they are compatible. Here, c_1, c_2, and c_3 get the same because both *inc* and *dec* are performed on every replica. Then, *dsp* can be performed. The detail of the protocol is discussed in the paper [15].

3 Redundant Invocation

3.1 Invocation on replicas

In the object-based system, methods are invoked in a nested manner. Suppose there are multiple replicas $x_1, \ldots, x_a$ of an object x and multiple replicas $y_1, \ldots, y_b$ of another object y and a method t of x invokes a method u of y. One way to invoke a method t on replicas of the object x is a *primary-secondary* replication. There is one *primary* replica, say x_1 and the others x_2, $\ldots$, x_a are *secondary* ones. Let y_1 be a primary replica of the object y. Every request is issued to a primary replica [Figure 2]. The method t is performed on a primary replica x_1, and then a request u is issued to a primary replica y_1. After the method on the primary replica commits, the state of the primary replica x_1 is transmitted to the secondary ones. The method is performed on only the primary replica and is not performed on any secondary replica. If a primary replica is faulty, this invocation way does not support higher availability because a new primary replica is required to be selected from the secondary ones. We take an approach where a method is issued to multiple

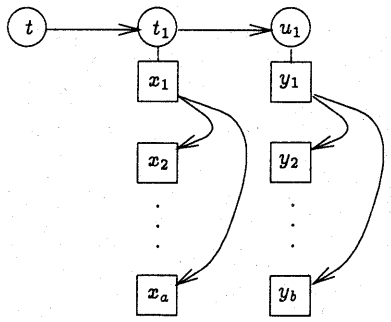

Figure 2: Primary-secondary replication.

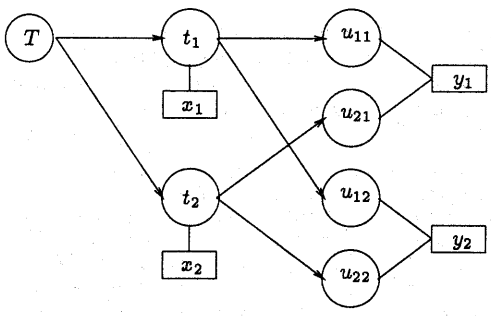

Figure 3: Redundant invocation.

replicas where the methods are performed.

Let $t_1, \ldots, t_m$ be instances of a method t on an object x in a transaction T. Suppose that the method t invokes a method u on an object y. Let $S(t_i)$ be a set of replicas of the object y to which an instance t_i issues a method u. If $|S(t_1) \cup \cdots \cup S(t_m)| > |Q_u|$, the transaction T manipulates more number of replicas of the object y than the quorum number $|Q_u|$, i.e. the quorum of the method u is exploded. $|S(t_1) \cup \cdots \cup S(t_m)| = |Q_u|$ is required to be held for the transaction T.

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 u to replicas of the object y in the quorum of the 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 instances t_1 and t_2 issues a request u to replicas in a quorum of u. Here, let Q_{u1} and Q_{u2} be quorums of u decided for the instances t_1 and t_2, respectively, where $|Q_{u1}| = |Q_{u2}| = |Q_u|$. Suppose $Q_{u1} = Q_{u2} = \{y_1, y_2\}$. Each of the instances t_1 and t_2 issues a request u to the same replicas y_1 and y_2. Here, let u_{i1} and u_{i2} show instances of the method u performed on replicas y_1 and y_2, which are issued by an instance t_i ($i = 1, 2$), respectively [Figure 3]. If u is an update type, a state of the replica y_1 is inconsistent because two instances u_{11} and u_{21} from the methods t_1 and t_2 are performed on the replica y_1, respectively. For example, suppose the value of y is multiplied by two through the method u. Every replica of the object y must be multiplied by two. However, the replica of y is finally multiplied by four since u is performed twice on the replica. This is a *redundant invocation*, i.e. a method on a replica is invoked by multiple instances of a method.

A transaction T invokes a method t on an object x and then t invokes a method u on an object y as shown in Figure 3. t_1 and t_2 show instances of a method t. Here, the instances t_1 and t_2 are referred to as *replica instances* of a same method t if t_1 and t_2 are invoked by a same instance or by different replica instances of a same method. In Figure 3, u_{11}, u_{12}, u_{21}, and u_{22} are replica instances of the method u. The following constraint has to be satisfied in invocations of a method t on multiple replicas of an object x.

[Invocation constraint] At most one replica instance of a method invoked by a transaction is performed on each replica. □

[Theorem] If every method is invoked on a replica so that the invocation constraint is satisfied, the replica is consistent. □

Suppose a replica is not changed by a method t. A replica of an object x is not changed even if t is performed t is performed multiple times on the replica. Hence, the invocation constraint is modified as follows:

[Weak invocation constraint] At most one replica instance of a method which changes a replica is performed on each replica. □

Let t be a method on an object x invoked by a transaction T.

$N_t(T)$ = number of replicas of x where t is invoked by T.
$I_t(T, x_i)$ = number of replica instances of t which are performed on a replica x_i.
$I_t(T)$ = total number of replica instances of t $= \sum_i I_t(T, x_i)$.

The following constraint have to be satisfied.

$$N_t(T) \geq N_t$$
$$I_t(T) \geq N_t(T)$$

If $N_t(T) = N_t$ and $I_t(T) = N_t$, the method t is *minimally* invoked by T. If t is minimally invoked on replicas by T, no redundant constraint occurs. Furthermore, no quorum explosion occurs.

3.2 Resolution

We discuss how to satisfy the invocation constraint. Each instance t_i of a method t invoked on a replica of an object x has an identifier $id(t_i)$. The identifier $id(t_i)$ is composed of method type t and identifier of the object x. Each transaction T has a unique identifier $tid(T)$, e.g. thread identifier of T. If T invokes a method t, t is assigned a transaction identifier $tid(t)$ as a concatenation of $tid(T)$ and invocation sequence number $iseq(T, t)$ of t in T. The *invocation sequence* number $iseq(T, t)$ is incremented by one each time T invokes a method:

- If t is a first method invoked by T, $iseq(T, t) = 1$.
- If T invokes t after t' and invokes no method before t after t', $iseq(T, t) = iseq(T, t') + 1$.

Here, suppose a method t_2 on an object x_2 is invoked by a method instance t_1 on x_1. $tid(t_2)$ is given as a concatenation of $tid(t_1)$, $id(t_1)$, and $iseq(t_1, t_2)$, i.e. $tid(t_2) := tid(t_1):id(t_1):iseq(t_1,$

t_2). Thus, the identifier $tid(t_i)$ shows an invocation sequence of method instances from T to t_i.

[Theorem] Let t_1 and t_2 be instances of a method t. $tid(t_1) = tid(t_2)$ iff t_1 and t_2 are replica instances of the method t invoked in a transaction. □

In Figure 3, $tid(T)$ is assumed to be 6. Suppose the transaction T invokes a method t on replicas x_1 and x_2 of an object x after invoking three methods. $iseq(T, t_1) = iseq(T, t_2) = 4$. Here, $tid(t_1) = tid(t_2) = tid(T)$:$iseq(T, t_1) = tid(T)$:$iseq(T, t_2) = 6$:4 and $id(t_1) = id(t_2) = t$:x. The method t invokes another method u on replicas of an object y after invoking one method. $tid(u_{11}) = tid(u_{12}) = tid(t_1)$:$id(t_1)$:2 = 6:4:$t$:$x$:2. $tid(u_{21}) = tid(u_{22}) = tid(t_2)$:$id(t_2)$:2 = 6:4:$t$:$x$:2. Since $tid(u_{11}) = tid(u_{21})$, u_{11} and u_{21} are replica instances of the method u on a replica y_1.

Suppose a method t is invoked on a replica x_h. The method t is performed on x_h as follows:

1. If t is issued to a replica x_h, an instance t_h of t is created and a response *res* of t is sent back. A tuple $\langle t, res, tid(t_h)\rangle$ is stored in the log L_h of x_h.
2. If $\langle t, res, tid(t'_h)\rangle$ such that $tid(t_h) = tid(t'_h)$ is found in the log L_h, i.e. t_h and t'_h are replica instances of a same method t, the response *res* of t'_h stored in the log L_h is sent back without performing t'_h.

Thus, even if multiple instances of a method are issued to a replica, they can be detected to be the replica instances by using the transaction identifier *tid*. By the resolution of the redundant invocation presented here, at most one replica instance of a method t is surely performed on each replica even if instances on multiple replicas invoke the method t. That is, $I_t(T, x_i) \leq 1$ for every method t on a replica x_i. In addition, the method t can be performed on a replica even if a replica x_i is faulty. Furthermore, a method u invoked by t is performed on some number of replicas of y. In Figure 3, u_{11} is performed on the replica y_1. $\langle u$, response of u, $tid(u_{11})\rangle$ is stored in the log L_1. Then, u_{21} is issued. Since $tid(u_{11}) = tid(u_{21})$, u_{21} is not performed and the response of u_{11} stored in L_1 is sent to t_2.

4 Quorum Explosion

4.1 Basic protocol

In Figure 3, the quorums Q_{u1} and Q_{u2} are assumed to be the same, $Q_{u1} = Q_{u2}$. Here, we assume Q_{u1} and Q_{u2} are different, $Q_{u1} \neq Q_{u2}$, say $Q_{u1} = \{y_1, y_2\}$ and $Q_{u2} = \{y_2, y_3\}$. Here, the method u is performed on each replica in a subset $Q = Q_{u1} \cup Q_{u2} = \{y_1, y_2, y_3\}$. The method u is performed twice on the replicas in $Q_{u1} \cap Q_{u2} = \{y_2\}$ as presented in the preceding section. If another transaction manipulates the object y through 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 number of replicas are manipulated by the transaction T than the quorum number N_u. Then, the instances of the method u on the replicas in $Q_{u1} \cup Q_{u2}$ issue further requests to other replicas and more number of replicas are manipulated. That is, $N_u(T) = N_u$. This problem is referred to as *quorum explosion*.

[Definition] A quorum of an object x for a method t is *exploded* if replica instances of t invoked in a transaction T are performed on more number of replicas of x than the quorum number N_t of t. □

Suppose a method t on an object x invokes a method u on an object y. Let Q_{uh} be a quorum of the method u invoked by an instance t_h of the method t on a replica x_h, i.e. subset of replicas of the object y. In order to resolve the quorum explosion, a pair of quorums Q_{uh} and Q_{uk} have to be the same for every pair of replicas x_h and x_k where an instance of the method t is performed. If a quorum is *a priori* decided for a method t, i.e. $Q_{uh} = Q_{uk} = Q_u$, only the same replicas are manipulated for every instance of the method u. If some method is frequently invoked, the replicas in the quorum are frequently manipulated. The replicas are overloaded. If some replica is faulty, a quorum including the faulty replica is required to be changed.

We take a following way to resolve the quorum explosion:

1. A same function *select* for generating a same set of numbers is supported for each replica. That is, $select(i, n, a)$ gives a set of n numbers out of 1, ..., a for a same initial value i where $n \leq a$. For example, $select(i, n, a) = \{h \mid h = (i + \lceil \frac{a}{n} \rceil (j - 1)) \ mod \ a$ for $j = 1, \ldots, n\} \subseteq \{1, \ldots, a\}$.
2. Suppose an instance t_h on a replica x_h invokes a method u. For each replica x_h, $I = select(numb(tid(t_h)), N_u, b)$ is obtained, where $numb(tid(t_h))$ is a number obtained from $tid(t_h)$, N_u is $|Q_u|$, and b is a total number of replicas, i.e. $\{y_1, \ldots, y_b\}$. Suppose $tid(t_h)$ is s_1:s_2:$\cdots$:s_g. For example, $numb(tid(t_h))$ is $(s_1 + \cdots + s_g)$ mod a. That is, $I \subseteq \{1, \ldots, b\}$ and $|I| = N_h$. Then, a quorum Q_h is constructed as $\{y_i \mid i \in I\}$.

Every replica instance invoked by a same instance has the same transaction identifier as presented in the preceding subsection. Hence, $select(numb(tid(t_h)), N_u, b) = select(numb(tid(t_k)), N_u, b)$ for every pair of replica instances t_h and t_k of the method t invoked by a same instance. An instance t_h of the method t on every replica x_h issues a request of the method u to the same quorum Q_{uh} $(= Q_u)$. Hence, no quorum explosion occurs [Figure 4].

The transaction identifier *tid* includes thread identifiers. Hence, the quorum of the method u can be differently constructed for each invoker of the method t depending on *tid* of the invoker.

4.2 Modified protocol

In this approach, each instance t_h on a replica x_h issues a request u to N_u replicas of the object y. Since each of N_t instances of the method t issues N_u requests, totally $N_t \cdot N_u$ requests are transmitted. We try to reduce the number of requests transmitted in the network. Let Q_u be a quorum of the method u obtained by the function *select* for each instance of the method t. Every instance t_h has the same quorum Q_u for a method u which to be invoked by t. If each instance t_h issues a request u to only a subset $Q_{uh} \subseteq Q_u$, we can re-

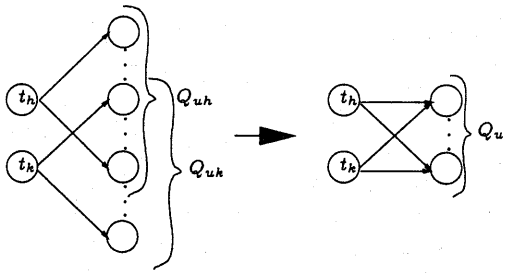

Figure 4: Resolution of quorum explosion.

duce the number of requests issued to the replicas of the object y. Here, let Q_t be a set $\{x_1, \ldots x_a\}$ and Q_u be $\{y_1, \ldots y_b\}$ where $a = N_t$ and $b = N_u$. Here, $Q_{u1} \cup \ldots \cup Q_{ua} = Q_u$.

In order to increase the reliability, each replica y_k in the quorum Q_u is required to receive a request u from more than one instance of the method t. Let r (≥ 1) be a *redundancy factor*, i.e. the number of requests of u to be issued to each replica y_k in Q_u. For each instance t_h on a replica x_h in the quorum Q_t, Q_{uh} is constructed for the method u as follows ($h = 1, \ldots, a$):

1. If $a \geq b \cdot r$,

$$Q_{uh} = \begin{cases} \{y_k \mid k = \lceil \frac{hb}{a} \rceil\} \text{ if } h \leq r \cdot b. \\ \phi \quad \text{otherwise.} \end{cases}$$

2. If $a < b \cdot r$,

$$Q_{uh} = \{\, y_k \mid (1 + \lfloor \tfrac{(h-1)b}{a} \rfloor) \leq k < [\, 1 + (\, \lfloor \tfrac{(h+r-1)b}{a} \rfloor - 1) \bmod b \,]\}.$$

For example, suppose instances t_1, t_2, and t_3 of a method t on replicas x_1, x_2, and x_3 issue a request u to replicas y_1, y_2, y_3, and y_4, i.e. the quorum $Q_t = \{x_1, x_2, x_3\}$ and $Q_u = \{y_1, y_2, y_3, y_4\}$. Suppose the redundancy factor r is 2. Hence, $Q_{uh} = \{y_k \mid (1 + (\lfloor \frac{(h-1)4}{3} \rfloor) \leq k \leq (1 + (\lfloor \frac{(h-1)4}{3} \rfloor + \lfloor \frac{8}{3} \rfloor - 1) \bmod 4)\}$. Hence, $Q_{u1} = \{y_1, y_2\}$, $Q_{u2} = \{y_2, y_3, y_4\}$, and $Q_{u3} = \{y_3, y_4, y_1\}$. Two requests from the instances of the method t are issued to each replica y_k. For example, suppose an instance t_1 on x_1 is faulty. t_2 sends u to replicas in Q_{u2} and t_3 sends u to replicas in Q_{u3}. Since $Q_{u2} \cup Q_{u3} = \{y_1, y_2, y_3, y_4\}$, u is sent to every replica in the quorum Q_u. If $r = 1$, $Q_{u1} = \{y_1\}$, $Q_{u2} = \{y_2\}$, and $Q_{u3} = \{y_3, y_4\}$. Thus, totally $r \cdot N_u$ requests of the method u are issued to the replicas in Q_u. Even if $(r-1)$ instances out of t_1, ..., t_a are faulty, the quorum number N_u replicas of y are manipulated.

5 Evaluation

We evaluate the QB protocol to resolve the redundant invocation and quorum explosion to occur in nested invocations of methods on replicas in terms of number of replicas manipulated and number of requests issued. The following three protocols R, Q, and N are considered:

1. Protocol R: without redundant invocation and quorum explosion.
2. Protocol Q: without redundant invocation.
3. Protocol N:

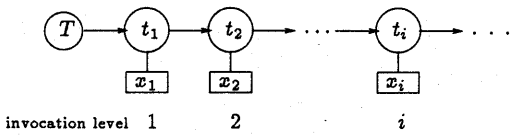

Figure 5: Invocation model.

In the protocol Q, the redundant invocation is prevented but the quorum explosion is not resolved. In the protocol R, neither redundant invocation nor quorum explosion occur. The protocol R shows the QB protocol discussed in this paper. In the protocol N, redundant invocation and quorum explosion may occur.

In the evaluation, we take a simple invocation model where a transaction T first invokes a method t_1 on an object x_1, then t_1 invokes another method t_2 on x_2, $\cdots$ [Figure 5]. Here, let a_i be the number of replicas of an object x_i ($i = 1, 2, \ldots$). Let N_i be the quorum number of a method t_i ($N_i \leq a_i$), where i shows a level of invocation. In the protocol, a transaction T first issues N_1 requests of t_1 to the replicas of x_1. Then, each instance of t_1 on a replica issues N_2 requests of t_2 to the replicas of x_2. In the protocol N, a method t_2 invoked by each instance of t_1 is performed. Here, totally $N_1 \cdot N_2$ requests are performed. In the protocol Q, at most one instance of t_2 is performed on each replica of x_2 by the resolution procedure of the redundant invocation. Since the quorum explosion is not resolved, the expected number QE_2 of replicas where t_2 is performed is $a_2[1 - (1 - \frac{N_2}{a_2})^{N_1}]$. Then, each instance of t_2 issues requests of t_3 to N_3 replicas of x_3. Here, $a_3[1 - (1 - \frac{N_3}{a_3})^{N_1 N_2}]$ replicas are manipulated in the protocol N and $QE_3 = a_3[1 - (1 - \frac{N_3}{a_3})^{QE_2}]$ replicas in the protocol Q. In the protocol R, t_2 is performed on only N_2 replicas of the object x_2.

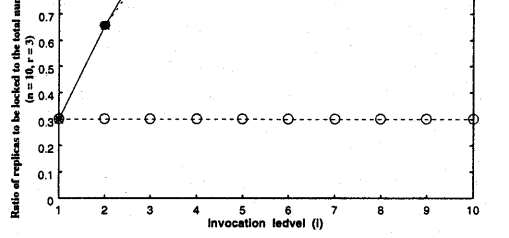

Figure 6: Ratio of replicas manipulated ($a = 10$ and $N = 3$).

In the evaluation, we assume that $a_1 = a_2 = \ldots = a = 10$ and $N_1 = N_2 = \ldots = N$ ($\leq a$). Figure 6 shows the ratios of replicas where a method is performed to the quorum number a of the replicas at each invocation level i for quorum number $N = 3$. The dotted line with white circles shows the ratio for the protocol R. The straight line indicates the protocol N and the other dotted line with black circles shows the protocol Q. If methods are invoked at a deeper level than two for $N = 3$, all the replicas are manipulated if the redundant invocation and quorum explosion are prevented.

Figure 7 shows the number of request messages transmitted by instances of a method t_i for $N = 3$. The vertical axis shows $\log(m)$ for m = number of requests issued. In the protocol N, N^i request messages are issued to replicas of the object x_i because N^{i-1} instances are performed on replicas of the object x_{i-1}. In the protocol Q, there are $a[1 - (1 - \frac{N}{a})^{QE_{i-2}}]$ replicas of x_{i-1} where instances of a method t_{i-1} are performed. Hence, $a[1 - (1 - \frac{N}{a})^{QE_{i-2}}]N$ request messages are transmitted. In the protocol R, we assume the redundancy factor $r = N$ in this evaluation. N^2 request messages are transmitted. Table 1 summarizes the number of requests issued for each protocol.

Table 1: Number of requests issued.

protocols	number m of requests at level i
Q	$a[1 - (1 - \frac{N}{a})^{QE_{i-2}}]N$
R	N^2
N	N^i

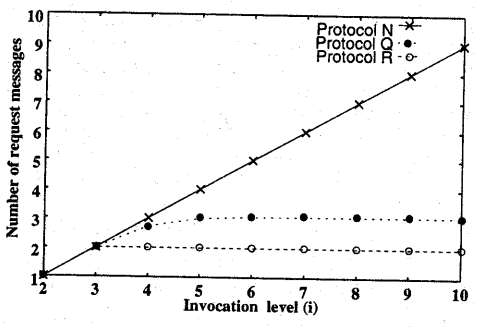

Figure 7: Number of request messages issued ($a = 10$ and $N = 3$).

Let us consider a pair of methods t and u on an object x. According to the traditional protocols [8, 9], a method is considered to be *write* if the object is changed by the method. Hence, if the object x is updated by t or u, $N_t + N_u > a$. However, $N_t + N_u \leq a$ if t and u are compatible in the QB protocol even if t or u is an update method. This means the less number of replicas are manipulated in the protocol R than the traditional ones.

6 Concluding Remarks

This paper discussed how transactions invoke methods on multiple replicas of objects. The object supports a more abstract level of method than read and write. In addition, methods are invoked in a nested manner. If methods are invoked on multiple replicas in a nested manner, multiple redundant instances of a same method may be performed on a replica and more number of replicas than the quorum number may be manipulated. We discussed the QB (quorum-based) protocol how to resolve redundant invocations and quorum explosions to occur in systems where methods are invoked on multiple replicas in a nested manner. By using the QB protocol with the resolution of redundant invocations and quorum explosions, an object-based system including replicas of objects can be efficiently realized.

References

[1] Ahamad, M., Dasgupta, P., LeBlanc R., and Wilkes, C., "Fault FTCS Tolerant Computing in Object Based Distributed Operating Systems," *Proc. 6th IEEE SRDS*, 1987, pp. 115–125.

[2] Barrett, P. A., Hilborne, A. M., Bond, P. G., and Seaton, D. T., "The Delta-4 Extra Performance Architecture," *Proc. 20th Int'l Symp. on* , 1990, pp. 481–488.

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

[4] Bernstein, P. A., and Goodman, N., "The Failure and Recovery Problem for Replicated Databases," *Proc. 2nd ACM POCS*, 1983, pp. 114–122.

[5] Birman, K. P. and Joseph, T. A., "Reliable Communication in the Presence of Failures," *JACM* , Vol. 5, No. 1, pp 1987, pp. 47–76.

[6] Borg, A., Baumbach, J., and Glazer, S., "A Message System Supporting Fault Tolerance," *Proc. 9th ACM Symp. on Operating Systems Principles*, 1983, . 27–39.

[7] Carey, J. M. and Livny, M., "Conflict Detection Tradeoffs for Replicated Data," *ACM TODS*, Vol.16, No.4, 1991, pp. 703–746.

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

[9] Gifford, D. K., "Weighted Voting for Replicated Data," *Proc. 7th ACM Symp. on Operating Systems Principles*, 1979, pp. 150-159.

[10] Korth, H. F., "Locking Primitives in a Database System," *JACM*, Vol. 30, No. 1, 1983, pp. 55-79.

[11] Powell, D., Chereque, M., and Drackley, D., "Fault-Tolerance in Delta-4," *ACM Operating System Review*, Vol. 25, No. 2, 1991, pp. 121–125.

[12] Silvano, M. and Douglas, C. S., "Constructing Reliable Distributed Communication Systems with CORBA," *IEEE Comm. Magazine*, Vol.35, No.2, 1997, pp.56–60.

[13] 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.

[14] Tanaka, K. and Takizawa, M., "Asynchronous Checkpointing Protocol for Distributed Object-Based Checkpoints," *Proc. IEEE ISORC'00*, 2000, pp. 218–225.

[15] Tanaka, K. and Takizawa, M., "Quorum-Based Replication of Objects," *Proc. 3rd DEXA Int'l Workshop on Network-Based Information Systems* (*NBIS-3*), 2000, pp. 33–37.