マルチメディア通信と分散処理 108－7
（２００２．６．６）

Replication Models of Object-based Systems

Kenichi Hori and Makoto Takizawa
Department of Computers and Systems Engineering
Tokyo Denki University
E-mail {hori, taki}@takilab.k.dendai.ac.jp

We discuss how to invoke methods on replicas of objects in a nested manner. If a method t is invoked on multiple replicas and each instance of t on the replicas invokes a method u on another object y, the method u is performed multiple times on the object of y although u is required to be performed just once. Then, the object gets inconsistent. This is redundant invocation. In addition, if each instance of the method t issues a request u to its quorum, more number of the replicas are manipulated than the quorum number of the method u. This is quorum explosion. We discuss an invocation protocol named Q protocol to resolve the redundant invocation and quorum explosion, and evaluate the protocol.

オブジェクトベースシステムにおける多重化モデル

堀　健一　滝沢 誠
東京電機大学大学院理工学研究科情報システム工学専攻

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、メソッド 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する。加えて、メソッド u の送信先のレプリカの集合（コーラム）が、送信元の t のインスタンス間で異っている場合、結果的に u のコーラム数以上のレプリカが操作されてしまう。これを、コーラム爆発とする。本研究では、冗長呼び出しとコーラム爆発を解決する入れ子呼び出し方法を示し、これの評価を行う。

1 Introduction

Objects are replicated in order to increase the reliability and availability in object-based applications. There are many discussions on how to replicate state-full servers like database servers [4, 6–9, 12, 13], like the two-phase locking protocol [6] and quorum-based protocol [8, 9]. An object is an encapsulation of data and abstract methods. A pair of methods conflict on an object if the result obtained by performing the methods depends on the computation order. In the paper [12], the quorum concept for read and write is extended to abstract methods. Suppose a pair of methods t and u are issued to replicas x_1 and x_2 of an object x. Even if a replica is updated by t or u, $N_t + N_u \leq a$ only if t and u are compatible.

In object-based systems, 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. Suppose a method t is issued to the replicas x_1 and x_2. Every method is assumed to be deterministic. Then, the method t invokes another method u on replicas y_1 and y_2. Here, the method u is performed twice on each replica although only one instance of u should be performed. If multiple instances of the method u are performed on some replicas, the replicas are gets inconsistent. This is a *redundant invocation.* An instance of the method t on the replica x_1 issues a method u to replicas in its own quorum Q_1, and another instance of t on x_2 issues 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 for a method u than N_u, i.e. $|Q_1 \cup Q_2| \geq N_u$. If the method u furthermore invokes another method, replicas to be manipulated are more increased. Even all the replicas may be manipulated although the quorum number is smaller than the little number of the replicas. This is a *quorum explosion*. In order to increase the reliability and availability, a method issued has to be performed on multiple replicas. On the other hand, the replicas may get inconsistent by the redundant invocations and the overhead is increased by the quorum explosion. We discuss how to resolve the redundant invocation and quorum explosion to occur in nested invocations of methods on multiple replicas.

In section 2, we discuss what kinds of problems to occur in nested invocation of methods on replicas. In sections 3 and 4, we discuss how to resolve the redundant invocation and the quorum explosion. In section 5, we evaluate the protocol.

2 Nested Invocation on Replicas

2.1 Methods

Methods are procedures for manipulating objects. We classify methods into *dependent* and *independent* types according to whether or not the results obtained by performing methods depend on object state. Computation of a dependent method t depends on state of an object x. Independent methods are performed independently of object state. Methods are also classified into *update* and *non-update* types of methods according to whether or not object state is changed by performing methods.

Let $t_1 \circ t_2$ show that a method t_2 is performed on an object after a method t_1. Let us consider a method *display* on *counter*. Each *display* shows same value even if any number of *display* is performed, i.e. *display* $\circ$ *display* = *display*. Here, suppose *increment* is performed after the first *display* before the second *display*. The second *display* shows different value than the first *display*. A method t is referred to as *conflict* with a method u iff the result obtained by performing t and u depends on the computation order of t and u. Otherwise, the method t is *compatible* with u.

2.2 Replications

As discussed by Wiesmann [13], there are different ways to replicate processes and database servers. Processes are stateless while database servers are statefull. There are three ways to replicate processes, *active*, *passive*, and *hybrid* ones. In the active replication [11], every replica receives same messages in a same sequence, same computation is performed on every replica, and same sequence of output is sent back. Here, the process p is required to be deterministic. The process is operational as long as at least one replica is operational. In the passive replication [5], there is one primary replica, say p_1, and the other replicas are secondary. Messages are sent to only the primary replica p_1 and the computation is performed on only the primary replica p_1. No computation is performed on any secondary replica. A checkpoint of the primary replica p_1 is eventually taken and then is sent to all the secondary replicas. The hybrid replication [2] is same as the passive one except that messages are sent to not only the primary replica but also the secondary replicas.

On the other hand, database servers are statefull. Ways to replicate database servers are classified with respect to which replica a request is issued to *eager* and *lazy*, and when other replicas are updated, *primary* and *everywhere*. Requests are immediately performed on replicas as soon as requests are issued in the eager type. On the other hand, requests are not immediately performed in the lazy one. In the primary replication, requests are performanced only on a primary replica. In the everywhere replication, requests are performed on all the replicas. For example, the two-phase locking protocol [3] is an eager, everywhere type because all the replicas are updated by a *write* request. The quorum-based replication [7,8] can be classified as somewhere, lazy one.

2.3 Primary-secondary replication

Objects are encapsulations of data and methods for manipulating the data. Objects are manipulated only through invoking methods supported by the objects. Here, suppose a transaction T invokes a method t on an object x. The method t is realized by invocations of other methods, say a method u on an object y. Thus, methods are invoked on objects in a *nested* manner.

Suppose there are replicas $x_1, \ldots, x_a$ (a>1) of an object x and replicas $y_1, \ldots, y_b$ (b>1) of another object y. One way to invoke a method t on the replicas of x is a *primary-secondary* one. First, the transaction T issues a request t to only a primary replica x_1. Then, the method t is performed and *then* a request u in t is issued to a primary replica y_1 [Figure 1]. After the method commits, the state of the primary replica is eventually transmitted to the secondary ones. For example, a checkpoint is taken on the primary replica and then the checkpoint data is transferred to secondary ones. Since only one instance of the method t invokes u, neither redundant invocations nor quorum explosions occur. For example, suppose y_1 is faulty when t_1 invokes u_1 on a replica y_1. One secondary replica, say y_2 is taken as a primary replica and t_1 invokes u_2 on primary replica y_2. Thus, the primary-secondary way implies less availability. Since every request is issued to a primary replica, the primary replica is overloaded.

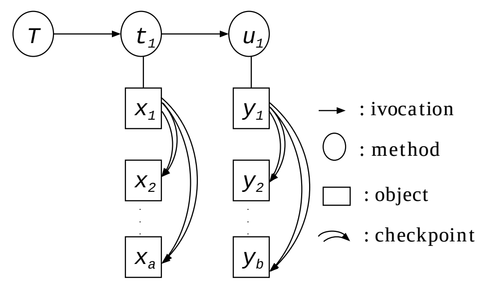

Figure 1: Primary-secondary replication.

2.4 Quorum-based replication

Suppose a transaction T issues a method t to replicas in the quorum $Q_t = \{x_1, x_2\}$ where N_t = 2. Furthermore, the method t issues a request u to replicas of the object y in the quorum of the method u, say $N_u = 2$. Let t_i be an instance of the method t performed on a replica x_i ($i = 1, 2$). Each instance t_i issues a request u to replicas in a quorum Q_{ui}. Suppose $Q_{u1} = Q_{u2} = \{y_1, y_2\}$. Here, let u_{i1} and u_{i2} show instances of the method u performed on replicas y_1 and y_2, respectively, which are issued by a method instance t_i (i = 1, 2) [Figure 2]. Suppose the method u is "$y = 2*y$". However, the replica y_1 is multiplied by four since a pair of instances u_{11} and u_{21} are performed on y_1. Thus, y_1 gets inconsistent. y_2 also gets inconsistent. This is a *redundant invocation*, i.e. a method on a replica is invoked multiple times by multiple instances of a method. Since every method is deterministic, the same computation of the method t is performed on the replicas x_1 and x_2. Here, t_1 and t_2 are referred to as *same clone* instances of the method t. u_{11}, u_{12}, u_{21}, and u_{22} are also same clone instances of the method u.

[Definition] A pair of instances t_1 and t_2 of a method t are *same clones* if t_1 and t_2 are invoked by a same instance or by same clones. □

Each replica has to satisfy the following constraint.

[Invocation constraint] At most one clone instance of a method invoked in a transaction is performed on each replica if the method is a dependent or update type. □

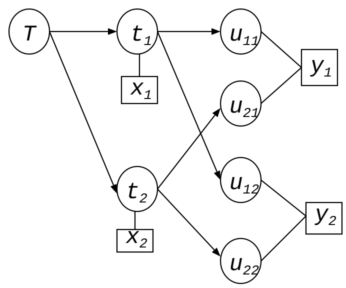

Figure 2: Redundant invocation.

In the transaction T discussed in the preceding subsection, suppose quorums are $Q_{u1} = \{y_1, y_2\}$ and $Q_{u2} = \{y_2, y_3\}$. An instance of the method u is performed on each replica in $Q_{u1} \cup Q_{u2} = \{y_1, y_2, y_3\}$. $|Q_{u1} \cup Q_{u2}|$ $(= 3) \geq N_u$ $(= 2)$. This means that more number of replicas of the object y are manipulated 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. The deeper level in a transaction, the more number of replicas are manipulated. This is *quorum explosion*.

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

3 Redundant Invocation

In order to resolve the redundant invocation, we have to make clear whether or not every pair of instances issued to each replica are same crones in a transaction. An identifier $id(t_i)$ for each instance t_i invoked on a replica of an object x is composed of a method type t and identifier of the object x, i.e. $id(t_i) = t{:}x$. Each transaction T has a unique identifier $tid(T)$, e.g. thread identifier of T. Each method t invoked in a transaction 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. $iseq(T, t)$ is incremented by one each time T invokes a method. Suppose an instance t_i on a replica x_i invokes an instance u_k on a replica y_k. $id(t_i) = t{:}x$. The transaction identifier $tid(u_k)$ is $tid(t_i){:}id(t_i){:}iseq(t_i, u_k)$ $= tid(t_i){:}t{:}x{:}iseq(t_i, u_k)$. $id(u_k) = u{:}k$. Thus, $tid(u_k)$ shows an invocation sequence of methods from T to the instance u_k.

[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 same crone instances of the t invoked in a transaction. □

We assume that $tid(T) = 6$ in Figure 2. Suppose T invokes a method t after invoking three methods, i.e. $iseq(T, t_1) = iseq(T, t_2) = 4$. Since $tid(t_1) = tid(t_2) = tid(T){:}iseq(T, t_1) = tid(T){:}iseq(T, t_2) = 6{:}4$ and $id(t_1) = id(t_2) = t{:}x$, t_1 and t_2 are same crone instances. t invokes another method u after invoking one method. Here $iseq(t,u)$=2, $tid(u_{11}) = tid(u_{12}) = tid(t_1){:}id(t_1){:}2$ $= 6{:}4{:}t{:}x{:}2$. $tid(u_{21}) = tid(u_{22}) = tid(t_2){:}id(t_2){:}2$ $= 6{:}4{:}t{:}x{:}2$. Since $tid(u_{11}) = tid(u_{21})$, u_{11} and u_{21} are same crone instances on a replica y_1.

A method t invoked on a replica x_h is performed as follows:

1. If no method is issued to a replica x_h, an instance t_h is performed and a response *res* of t is sent back. $\langle t, res, tid(t_h)\rangle$ is stored in the log L_h.
2. If $\langle t, res, tid(t'_h)\rangle$ such that $tid(t_h) = tid(t'_h)$ is found in L_h, the response *res* of t'_h is sent back as the response of t_h without performing t_h. Otherwise, t is performed on the replica x_h as presented at step 1.

In Figure 2, suppose u_{11} is issued to the replica y_1. $\langle u$, response of u_{11}, $tid(u_{11})\rangle$ is stored in the log L_1. Then, u_{21} is issued. Since $tid(u_{11}) = tid(u_{21})$, i.e. u_{11} and u_{21} are same crones, u_{21} is not performed but the response of u_{11} as the response of u_{21} is sent to t_2. Here, at most one crone instance is surely performed on each replica. In addition, each method can be performed on some replica even if a replica is faulty.

4 Quorum Explosion

4.1 Basic protocol

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. 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. If $Q_{uh} = Q_{uk} = Q_u$, only the same replicas are manipulated for every instance of u. If a method is frequently invoked, the replicas in the quorum are overloaded. The quorum of the method u has to be randomly decided each time u is invoked. If some replica is faulty, the quorum including the faulty replica has to be updated. We have to discuss a mechanism to randomly create a quorum Q_{ui} for each invoker instance t_i to invoke a method u in presence of replica fault of y, $Q_{ui} = Q_{uj}$ only if instances t_i and t_j are same crones in a transaction. $Q_{ui} \neq Q_{uj}$ can hold if t_i and t_j are different crones.

We introduce a following function *select* to decide a quorum:

1. A function $select(i, n, a)$ gives a set of n numbers out of $1, \ldots, a$ for a same initial value i where $n \leq a$. For example, $select(i, n, a) = \{h \mid h = (i + \lceil \frac{a}{n} \rceil (j-1))$ *modulo* 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. $I = select(numb(tid(t_h)), N_u, b)$ is obtained, where N_u is quorum number of u and b is a total number of replicas of y, i.e. $\{y_1, \ldots, y_b\}$. Let $tid(t_h)$ be $s_1{:}s_2{:}\cdots{:}s_g$.

Here, $numb(tid(t_h))$ is $(s_1 + \cdots + s_g)$ *modulo* a. $I \subseteq \{1, \ldots, b\}$ and $|I| = N_u$. Then, $Q_{uh} = \{y_i \mid i \in I\}$.

Every pair of same crone instances have the same transaction identifier *tid*. An instance t_h on every replica x_h issues a method u to the same quorum Q_{uh} as the other same crones. In addition, a quorum Q'_{uk} obtained for another crone instance t'_h is different for Q_{uh}. Hence, no quorum explosion occurs [Figure 3].

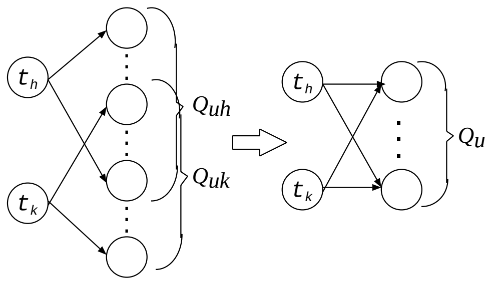

Figure 3: Resolution of quorum explosion.

Some replica may be faulty. Suppose a method t invokes a method u on replicas of an object y. Let Y be a set $\{y_1, \ldots, y_b\}$ of replicas of y. Here, suppose some replica y_h is faulty. Here, the quorum number N_u can be decremented by one as far as at most k replicas of the object y are faulty, i.e. $N_u + N_v - b = k$ for every method v conflicting with the method u. In one case, an invoker instance does not know that the replica y_h is faulty. Here, a quorum Q_{ui} including N_u replicas are selected by *select*. The instance t_i issues a method u to replicas of y in Q_{ui}. Since there is no response from y_h, the instance t_i finds that y_h is faulty. In another case, the instance t_i knows that y_h is faulty. If $y_h \in Q_{ui}$, y_h is removed from Q_{ui}. Here, $|Q_{ui}| = N_u - 1$. Unless $y_h \in Q_{ui}$, one replica y_l is removed from Q_{ui}. For example, a replica y_l where l is the minimum in Q_{ui} is selected and removed from Q_{ui}. The method u is required to be issued to $(N_u - 1)$ replicas of the object y.

If a faulty replica y_h is recovered, y_h informs all the operational replicas. Each invoker instance t_i on a replica x_i obtains a same quorum Q_{ui} for a method u to be invoked by *select* function. In one case, the instance t_i perceives that y_h is still faulty. If y_h is included in Q_{hi}, y_h is removed from Q_{ui}. The instance t_i issues a method u to replicas in Q_{ui}. In another case, t_i knows that y_h is recovered. The instance t_i issues u to the quorum Q_{ui} obtained. Thus, each invoker instance can obtain a quorum of an invoker method based on only its own view showing which replica is operational.

4.2 Modified protocol

Each instance t_h on a replica x_h issues a method request u to N_u replicas of the object y. Hence, 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 $\{y_1, \ldots, y_b\}$ $(b = N_u)$ of the method u obtained by the function *select* for each instance t_h. If each instance t_h issues a request u to only a subset $Q_{uh} \subseteq Q_u$, the number of requests issued to the replicas of the object y can be reduced. Here, $Q_{u1} \cup \ldots \cup Q_{ua} = Q_u$.

In order to tolerate the fault of a replica, each replica y_k in Q_u is required to receive a method request u from more than one instance of the method t. Let r (≥ 1) be a *redundancy factor*, i.e. the number of the requests to be issued to each replica y_k in Q_u. For each instance t_h on a replica x_h in $Q_t = \{x_1, \ldots x_a\}$ where $a = N_t$, Q_{uh} is constructed for the method u as follows ($h = 1, \ldots, a$):

If $a \geq b \cdot r$, $Q_{uh} = \{y_k \mid k = \lceil \frac{hb}{a} \rceil\}$ *if* $h \leq r \cdot b$

$Q_{uh} = \phi$ *otherwise.*

If $a < b \cdot r$, $Q_{uh} = \{ y_k \mid (1 + \lfloor \frac{(h-1)b}{a} \rfloor) \leq k < \lceil 1 + (\lfloor \frac{(h+r-1)b}{a} \rfloor - 1) \text{ modulo } b \rceil \}$.

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

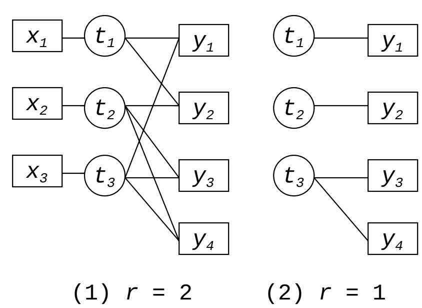

Figure 4: Invocations.

5 Evaluation

We evaluate the invocation protocol named quorum-based invocation (Q) protocol discussed in this paper to resolve the redundant invocation and quorum explosion to occur in nested method invocations on multiple replicas. The Q protocol is evaluated in terms of number of replicas manipulated, number of requests issued, and response time compared with the primary-secondary invocation (P) protocol shown in Figure 1.

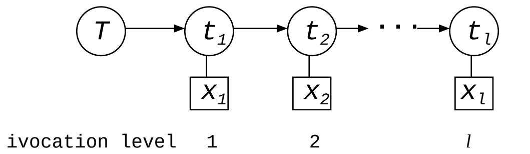

Figure 5: Invocation model.

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 t_2 on x_2, $\cdots$ as shown in 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 $i(\leq l)$ shows a level of invocation. l shows the invocation level of the transaction T. Let r_i be a *redundancy factor* on an object x_i. In the primary-secondary (P) protocol, only a method on a primary replica is invoked as presented in Figure 1. Suppose a method t_i invokes another method t_{i+1} on a primary replica x_{i+1}[Figure 6]. If a primary replica x_{i+1} is faulty, one secondary replica x'_{i+1} is taken as a new primary replica of x and a method t_{i+1} on the replica x_{i+1} is invoked again. In addition, the replica x'_{i+1} might be faulty during invocation of t_{i+1}. Let f_i be probability that a replica of an object x_i is faulty. Thus, the higher the fault probability f_i is, the longer it takes to perform the transaction T. We assume $f_1 = f_2 = \ldots = f_l = f$.

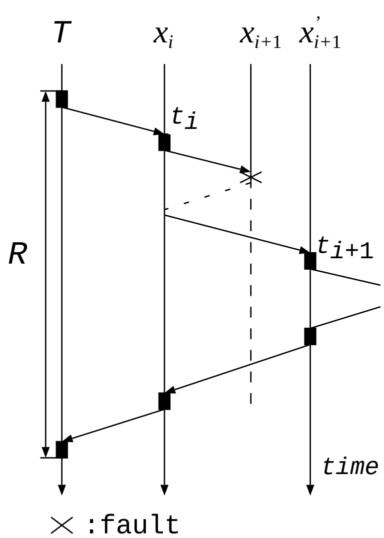

Figure 6: Primary-secondary (P) protocol.

In the Q protocol, each method t_i is performed on only N_i replicas of an object x_i as long as at least r_i replicas are operational. We assume that $a_1 = a_2 = \ldots = a = 10$, $N_1 = N_2 = \ldots = N_l = N$ ($\leq a$), and $r_1 = r_2 = \ldots = r_l = r$.

Figure 7 shows the number of replicas where methods are performed in a transaction whose invocation level is l. "Protocol Q" shows the Q

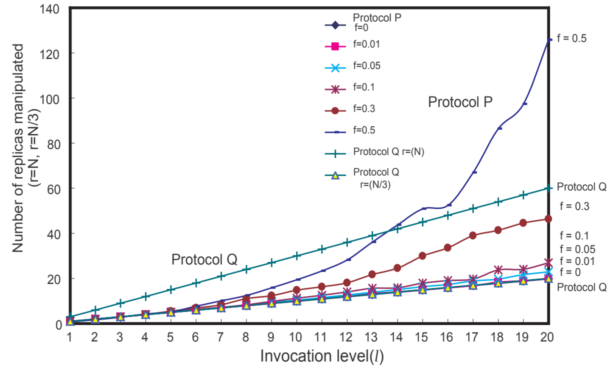

Figure 7: Number of replicas manipulated.

protocol with the number of replicas $a = 10$ and quorum number $N = 3$. Only the quorum number N of replicas, i.e. three replicas, out of ten replicas are manipulated at each invocation level. Protocol P shows number of replicas manipulated for fault probability f(=0, 0.01, 0.05, 0.1, 0.3, 0.5). If a replica is faulty, another replica is manipulated. As shows in Figure 7, the more replicas are faulty, the more replicas are manipulated.

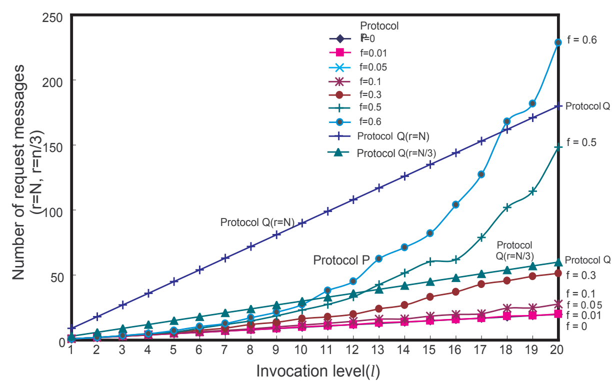

Figure 8: Number of request messages issued.

Figure 8 shows the number of request messages transmitted. In the Q protocol, rN messages are transmitted at each invocation level. Hence, rN^2l request messages are transmitted for a transaction with invocation level l. For $r = N$, N^2 request messages are transmitted at each invocation. For $r = N/3$, $N^2/3$ messages are transmitted. In Figure 8, the numbers of request messages issued are shown for $r = N$ and $r = N/3$. In the P protocol, totally i request messages are transmitted if no fault occurs, i.e. $f = 0$.

Next, let us consider response time of transaction with invocation level l in the Q and P protocols. Let δ_i be delay time to send a message from a replica of x_{i-1} to a replica of x_i. Let π_i show time for processing one request on a replica x_i. Here, we assume $\delta_1 = \delta_2 = \ldots = \delta$ and $\pi_1 = \pi_2 = \ldots = \pi$. In the Q protocol, the response time R_Q is ($2\delta + \pi)l$. In the P protocol, the response time R_P is [$2\delta\cdot$(number of request messages) + $\pi\cdot$(number of replicas manipulated)]l for fault probability f, which are obtained from Figures 7 and 8. Here, π = $\alpha \cdot \delta$. Figures 9 and 10 show the ratio R_P/R_Q for α=0.25 and α=4. "α=0.25" shows that the delay time is four times longer than the processing time. "α=4" indicates that the delay time is one fourth of the processing time. These figures show that the Q protocol supports shorter response time than the primary-secondly (P) protocol while implying larger number of messages transmitted. In addition the Q protocol is better in a system where replicas are interconnected in a high-speed local area network.

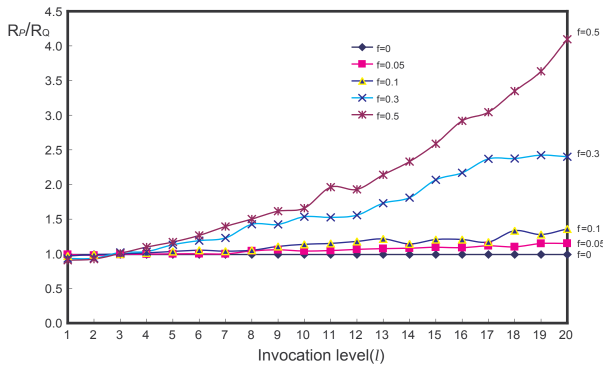

Figure 9: Response time (α = 0.25).

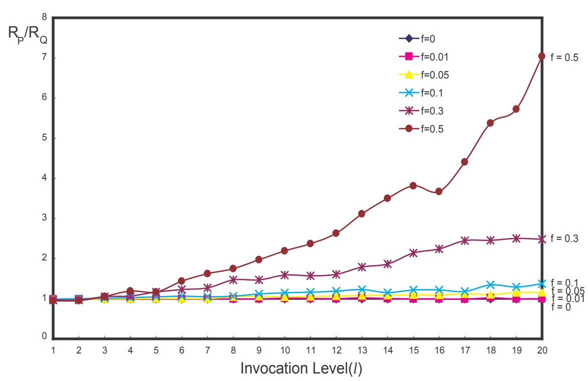

Figure 10: Response time (α = 4).

6 Concluding Remarks

In this paper, we discussed how transactions invoke methods on multiple replicas of objects. Methods are invoked in a nested manner. If methods are invoked on multiple replicas, multiple redundant instances of a same method are performed on a replica and more number of replicas than the quorum number are manipulated. We discussed the Q (quorum-based invocation) protocol where redundant invocations and quorum explosions to occur are resolved. By using the Q 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 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 FTCS*, 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] Budhiraja, N., Marzullo, K., Schneider, B.F., and Toueg, S., "The Primary-Backup Approach," *ACM Press*, 1994, pp.199–221.

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

[7] Chevalier, P. -Y., "A Replicated Object Server for a Distributed Object-Oriented System," *Proc. IEEE SRDS*, 1992, pp.4-11.

[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] Moss, J. E., "Nested Transactions : An Approach to Reliable Distributed Computing," *The MIT Press Series in Information Systems*, 1985.

[11] Schneider, B. F., "Replication Management using the State-Machine Approach," *Distributed Computing Systems, ACM Press*, 1993, pp.169–197.

[12] 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.

[13] Wiesmann, M., et al., "Understanding Replication in Databases and Distributed Systems," *Proc. of IEEE ICDCS-2000*, 2000, pp.264–274.