J. Software Engineering & Applications, 2009, 2: 276-282
doi:10.4236/jsea.2009.24035 Published Online November 2009 (http://www.SciRP.org/journal/jsea)

A CORBA Replication Voting Mechanism for Maintaining the Replica Consistent

Guohua WU, Xiaojun LI, Qiuhua ZHENG, Zhen ZHANG

College of Computer Science, Hangzhou Electronic University, Hangzhou, China.
Email: lixjun007@gmail.com

Received April 23rd, 2009; revised June 29th, 2009; accepted July 8th, 2009

ABSTRACT

Nowadays, more and more applications are being developed through distributed object computing middleware, such as CORBA, their requirements for fault-tolerance, especially real time and critical system, become more and more critical. Despite almost ten years have passed since the earliest FT-CORBA standard was promulgated by Object Management Group (OMG), CORBA is still facing many challenges when it is used for distributed applications developing, as the standard is complex and lack of understanding. This paper focus on the consistency of the replicated object and the network partition problem, it incorporates a CORBA Replication Voting Mechanism (CRVM) to meet the challenge which makes a good performance.

Keywords*: *distributed computing, CRVM, fault-tolerance, replica consistent, networking partition

1. Introduction

The progress of the distributed application system and object-oriented programming technology has led to distributed object middleware, where objects are distributed across processors. Typical middleware applications contain sending client objects' requests and receiving reply from server objects, which implementing through message sent across the network. The Common Object Request Broker Architecture (CORBA) [1] is a standard for middleware and it is established by the Object Management Group.

CORBA has become one of the most popular middleware developing technologies, which is supported on almost every combination of hardware and operating system in existence. CORBA uses OMG Interface Definition Language (IDL) to define interface for objects, which is CORBA's fundamental abstraction mechanism for separating object interfaces from their implementations. A client only needs to know the IDL interface without the language-specific implementation of the server object. Under the CORBA's standard, clients and servers can communicate with the TCP/IP-base Internet Inter-ORB Protocol (IIOP), careless of the heterogeneity in their respective platform and operating system. Clients are allowed to invoke operation without caring about the server objects' physical location. These are attributed to the ORB, which make clients and servers transparent to each other's differences in platform, programming language and location.

More and more distributed system desire highly-performance, including dependability, efficiency, reliability etc, thus adding fault-tolerant standard to CORBA becomes more and more pressing. OMG adopted Fault-Tolerant CORBA standard in the late 1990s (1999.2 Version1.0, 2001.9 Version2.0). Although almost ten years have passed, due to the diverse set of fault-tolerance requirements and the large varieties of distributed applications requiring fault-tolerance, the current version of FT-CORBA standard compromises on the number of interfaces, policies, and features it provides. As a result, FT-CORBA vendors are free to provide proprietary extensions [2].

Replication is the most basic way we adopt to achieve fault tolerance, however it still faces many challenges. How to maintain replica consistency is a typical problem, as Fault tolerance will obviously fail if replicas are in different status during a servicing time. Another problem is network partition. There is no support for the consistent remerging of the replicas of CORBA objects following a network partition [3]. This paper focuses on these two problems as mentioned above and it introduces a voting mechanism CRVM to meet the challenges.

The remainder of this paper is organized as follows. Section 2 describes FT-CORBA standard and existing fault-tolerance strategies. Section 3 provides the related work about fault-tolerance in CORBA. Section 4 describes the dynamic voting algorithm in detail. Section 5, we put forward to vote mechanism (CRVM). Section 6 describes an implementation. Section 7 concludes the

paper.

2. The review of FT-CORBA Standard and Strategies

2.1 The FT-CORBA Standard

To achieve the purpose of fault-tolerant, FT-CORBA standard provides three mechanisms: Replication, Fault Detection and Fault Recover.

1) Replication: There is a Replication Manager (RM) responsible for replicating objects and distributing the replications across the system. Each replica has an individual Interoperable Object Reference (IOR) to identify them，RM itself also has many copies. The replication styles can be divided into the following two types [4]:

- Passive replication: only one of the server replicas is designated as the primary one, which responses for the client's requests. In the warm passive replication styles, the remaining passive replicas, called backups, are updated periodically with the primary replica so that one of them can be selected to replace when the primary one fails. In the cold passive replication styles, the remaining backup replicas are "cold", they would neither process the client's requests nor make update with the primary one. To allow for recovery, the state of the primary replica is periodically checked and stored in a log. Once the primary replica failed, a backup replica is selected and initialized from the log to replace as the new primary one.
- Active replication: all of the server replicas maintain the same state, each one responses to client's requests. There is no influence when any one of them failed, keeping the running of replicas without any problem.

2) Fault Detection: Fault Detection implement by FD which is in charge of detecting the possible failure in the system and generates related reports. FD is arranged into a hierarchical structure: object level, process level and host level due to different detected objects. FD would generate a fault report to the Fault Notifier (FN) when a failure is detected, then the FN transmit it to RM and the objects which have been registered to FN and are interested in it. There are two kinds of detecting styles, pull-based and push-based.

- pull-based: FD periodically send a message Is_alive () to the detected objects, which should echo "I_am_alive ()" in the limited time. The objects would be considered erring in case of the FD did not receive the response in a certain time, and then FD would send fault report to RM for handling. It is passive for detected objects to echo, they won't return "I_am_alive ()" unless receive FD's "Is_alive ()".
- push-based: In this scheme, it is also known as a "heartbeat monitor". The detected object would send the message "I_am_alive ()" forwardly at set intervals. It is same to the pull-based styles, the object would be considered erring if FD did not receive the response in a certain time. It is active for detected objects to send the message "I_am_alive ()" periodically.

3) Recovery: With FT-CORBA fault-tolerant mechanism, the objects' state would be logged automatically if the replica inherit and implement the Checkpoint table and Updateable interfaces, once receive the notice the object would be returned to the latest state.

2.2 The Strategies of Existing FT-CORBA System

Generally speaking, research on FT-CORBA and its applications can be divided into the three strategies [5], the integration strategy, the interception strategy, and the service strategy, which are described in detail as below:

- Integration strategy: In this strategy it has to modify the ORB core to provide the necessary fault tolerance support, thus, it is certain to violate the FT-CORBA standard.

Because the modification is added to the ORB, application's interfaces to the ORB remains unchanged, it implies that the replication of server objects can be made transparent to the client objects. Electra [6] and Orbix+Isis [7] are examples of the integration strategy.

- Interception strategy: In this strategy, an interceptor is added into the architecture to capture the system's call, and then to modify the request parameter to change the behavior of the application without the application or the ORB being aware of the interceptor's existence and operation, finally repackage the call and deliver it through multicast. However, the interceptor has to be ported to every operating system which is intended to run the CORBA application. The examples of this strategy are Enternal System [8], and AQuA framework [9], etc.
- Service strategy: This strategy enhances CORBA through adding a new service which just likes one of CORBA Common Object Service. In order to achieve fault tolerance, a set of interfaces are defined to provide the policies and mechanisms. In other words, fault tolerance can be provided as apart of the standard suite of CORBA ORBs. Since the CORBA service is a collection of CORBA object fully above the ORB, the ORB is certainly unnecessary to be modified. However, the application code may require modification for supporting the fault-tolerance service.

The comparison of different strategies outlined above is illustrated in Table 1 [3].

2.3 Replication

No matter adopting which kind of strategy to achieve the purpose of fault-tolerant, replication is the most basic way, which has been widely applied in distributed systems. The purpose of replication is to provide multiple, redundant, identical copies, or replicas, of an object so that the object can continue to provide useful services,

Table 1. Comparison of three strategies

Strategy	Advantages	Disadvantages	System
Intergration	Transparent to client applications (Without modify any co- de of client ap- plications)	Bad portable due to modify ORB and IDL	Orbix+Isis Electra
Interception	Without modify ORB and transparent to the applications	Interceptor needs to be ported to every operating system which intend to use it	Eternal
Service	Has a good portability to client applications	Sever applications are lack of transparent.(Program developer have to rewrite interface of the related objects)	DOORS AQuA

even though some of its replicas fail, or as the processors hosting some of its replicas fail [10].

This technology duplicates system resources and distributes them into hosts which locate in different places. Certainly, the status and behavior of the replicas must been maintaining the same, once a certain replica failed, the back-up replicas could take over and continue to provide services. This process is transparent to the client which is not aware of the server object weather has failed or not since the back-up replicas still keep on proving service to meet the client's demand.

Some advantages of replication:

- Enhanced system available: Replicative resource distribute in different hosts, the remaining replica could keep on providing service when one certain host failed. For example, suppose there are n hosts and the error probability of each host is p, obviously, the available of the resource could be expressed: 1-pn，it implies that the more replicas it would be the stronger available.
- Fault-tolerant: it is described above.

Although replication has many advantages, the challenges resulted are also critical .One important issue is how to ensure the consistency of the replicas. Besides, there is no good solution to resolve network partition. The FT-CORBA standard does not provide mechanism to handle faults due to network partitioning [2].

In this paper we introduce CRVM to resolve the problem mentioned above: replica consistency and network partition.

3. Related Works

Previous study on fault tolerance has made great help to FT-CORBA standard which is shown as follows.

Electra [6] was developed at the University of Zurich, as one of the earliest implementations in accordance with fault tolerance CORBA standard, which adopted integration strategy to achieve maintaining replica consistency. It contains a modified ORB core and the Hours toolkit [11] which is exploited to provide reliable totally ordered group communication mechanisms. In an Electura host, a CORBA client can request a replicated server object for operating with out caring about where is the server object or the number of them, even if the server object exists.

Orbix+Isis [11] was developed by IONA technologies which is similar to Electra, this product also modifies the internals of the ORB in order to adjust to Isis toolkit [12] which provides the reliable ordered multicast protocols. Besides, Orbix+Isis is the first commercial product which complies with the standard.

Distributed Object-Oriented Reliable System (DOORS) [2] which was developed at Lucent technologies, provided a service strategy to manage the object groups and replica objects. This product was absorbed in passive replication which employs libraries for the transparent checkpointing of applications and state recovery.

Eternal [8] which was developed at University of California, Santa Barbara, adopted interception strategy to support both active and passive replication styles. The mechanism implemented in different parts of the Eternal system together with its logging recovery mechanisms to ensure strong replica consistency without modifying either the application or the ORB.

4. Dynamic Voting Algorithm

Dynamic voting algorithm [13,14] proposed by two scholars Sushil Jajodia and Dvid Mutchler is used to control replication. The algorithm is designed for the fault-tolerant of replicated database, its major purpose is to maintain highly-consistency of the database and enhance system available. Actually, it has a good performance in fault-tolerant as well as in maintain consistency of data.

In our algorithm, we assume that all of the hosts and networks may go wrong, except for the Byzantium fault. In other words, the host would stop running without processing any order once the host failure occurred. Since the property of replicated database itself can tolerate host fault or process fault, another contribution of the algorithm is put forward FT strategy to resolve the network partition problem.

It adopts majority decision to determine which part of network contains more database, called majority partition, can continue running. In comparison, the remaining database should stop immediately to avoid two parts of database running simultaneously without communicating to update for consistency. The remaining database keep on running may lead to inconsistent, which is illustrated in Figure 1.

In simple terms, fault-tolerant of this algorithm is to duplicate data, distribute them to different hosts, and maintain data consistency through voting method. It means before accessing to a replica data, a voting step must be processed to insure the replica data is contained

in the majority partition.

Some parameters are necessary to meet the demand of the algorithm, which is needed when processing judgment.

- Version Number (VN): an integer, expresses the update times of the replica data;
- Site Cardinality (SC): an integer, expresses the number of replica data, i.e. number of hosts that replica is stored.
- Distinguished Site (DS): a replica data identifier, which is assigned as primary one.

Client send request to a single replica data without being aware of how does the replica data communicate with each other, it's transparent to client. Since every replica data can process the request from client, it also enhances the system's performance. The running diagram of dynamic voting algorithm is illustrated in Figure 2.

Algorithm processes and procedures are described as follows:

1) Set synchronous control: We assume that replica S receives request operation, it would send Concurrency_Request to remaining replicas at first. This part achieves synchronous control by using time stamp, thus, ensure there is only one replica is updated at a certain time. Bernstein [10] had ever advanced an algorithm to achieve mutual exclusion in distributed system which is described in detail as below:

Assume that process Pi wants to enter a critical section, a new time stamp (TS) would be generated and then (Pi, TS) would be dispatched to all processes. How does Pi reply to (Pi, TS), assent or delay depends on the three factors as showed below:

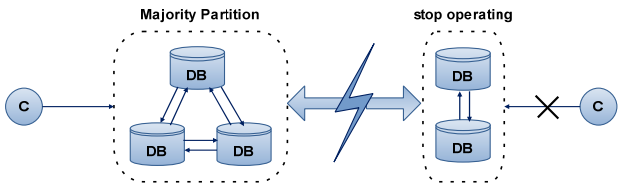

Figure 1. The network partition fault

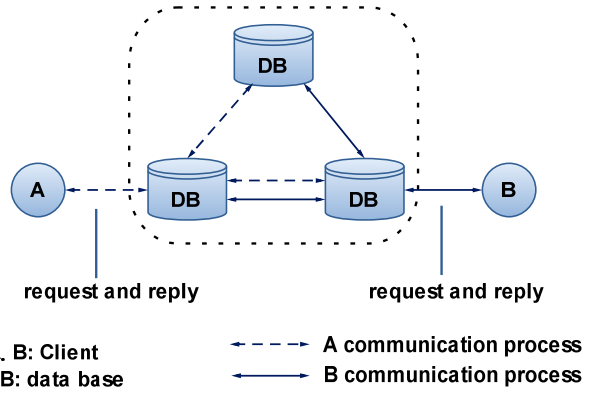

Figure 2. The communication among replicated DBs

1).If Pj is just alive in the critical section, it will chose to delay the response;

2) If Pj is not interested in the critical section, it will reply immediately;

3) If Pj also wants to enter the critical section, it will compare its TS with Pi's, obviously if TSj > TSi, send reply immediately, otherwise delay the reply.

The algorithm mentioned above has several features: Obtain mutual exclusion function, never get dead lock or starvation for whether process could enter the critical section obeys to the value of TS which ensures process served by FCFS.

2) Send Vote_Request: S execute Lock_Request after finishing Concurrency_Request, if lock successfully, S then send Vote_Request to remain replica data, supposed as Si, otherwise, S must execute termination protocol, and then do Lock_Request again, if still failed, the operation would be ended.

With executing termination protocol, S would send Decision_Request (contain DS and request identifier) to all remaining replica data. The replicas which receive request then reply whether update operation has finished, if finished, S then executes Realese_Lock.

3) Return status: Si will lock itself at first when receives Vote_Request, if locked successfully, returns VN SC DS to S, otherwise, executes termination protocol, later locks again, if still failed, returns null to S.

4) Determine majority partition: S collects all status replied by Si, and then executes Is_Distinguished to determine whether the network S is contained in the majority partition which is shown as follows:

m = number of replicas
M = max{VNi: 1≤i≤m} (an integer)
I = {i: VNi =M, 1≤i≤m } (a replica set)
N = {SCi: i∈I} (an integer)
If $|I| > \frac{N}{2}$ or ($|I| = \frac{N}{2}$ and DS exists) then
Do_Update
else
end

If S doesn't belong to majority partition, the update request has to abort. Firstly, S executes Release_Lock and sends message "Abort" to Si, and then Si would execute Release_Lock once receives "Abort".

If S belongs to majority partition, determines whether the status of S is the latest first. If negative, S should ask Si for it through message"Catch_Up".

5) Response the request and deliver out latest status: S updates firstly and then executes Do_Updata: VNi+=1, SCi=m. After that, S delivers message "Commit" (contain Request identifier and status) to Si. Finally, S replies the result to client.

6) Si will update status once receives "Commit" and then executes Release_Lock request.

5. CRVM for FT-CORBA

Despite the dynamic voting algorithm is not designed especially for CPRBA object fault-tolerant, however, there are many similarities between replicated database and replicated object. Our research indicates that dynamic voting algorithm is proper to maintain consistency of replicated object, resolve network partition, crash failure etc. Thus in this paper we propose a new CORBA Replication voting mechanism (CRVM) for FT-CORBA. Figure 3 illustrates the CRVM working graph.

The same as original algorithm, we have to set some parameters to satisfy judgment demand, which is expressed as follows:

- Object State (OS): an integer used to signify update time of the object.
- Object Cardinality (OC): an integer used to indicate how many replica objects exist which maintain the same status.
- Distinguished Object (DO): an IOR related to an identified object which is used when CRVM wants to judge whether current object belongs to the majority partition.

The CRVM is expressed as follows:

```
    [Illustrate]
    cur_ser_obj: the current server object which link to
client and raise voting;
    alive_obj: the objects reply voting request from
cur_ser_obj;
    T: the objects' time stamp formed by IOR and system
time;
    R: client request;
    S: status of object.
    OM: object management;

    OS, OC and DO initiated by OM
status ∈ {null, active, voting, commit, abort}, initially null
  lock ∈ {true, false}, initially false
  LockedTimeStamp ∈ T, initially null
  TimeStamp ∈ T, initially null
  RequestList, a table store request and result
```

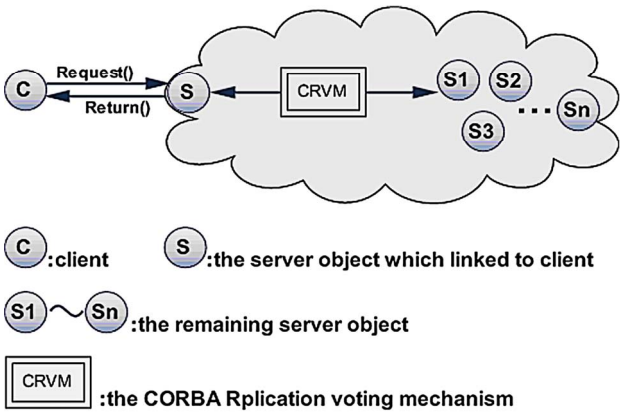

Figure 3. CRVM between replicas

```
[status transfer]
if status == active
send Concurrency_Request to all alive_obj
  status = voting
if status == voting
if (!Lock_Request)
    send Decision_Request to all alive_obj
    if (Lock_Request)
    send Vote_Request to all alive_obj
       if (majority partition)
          store request and result in Request Record
          status = commit
       else
          status = abort
if status = commit
  send Commit and object states to all alive_obj
status = idle
if status = abort
  send Abort to all alive_obj
    status = idle
```

```
[request processing]
receive ("Request",req,ts ), req ∈ R, ts ∈ T
if req not in RequestRecord
    status = active
else
    return request-record(ts)
receive("Concurrency_Request",ts), ts ∈ T
  if not conform to Berdtein's algorithm
    waiting
  else
    return
receive ("Decesion_Request",ts), ts ∈ T
  if ts found in LockRecord
    return false
  else
    return true
receive ("Vote_Request",ts), ts ∈ T
  if (!Lock_Request)
    send Decision_Request to all objects
  if (Lock_Request)
    LockedTimeStamp = ts
    store ts in LockRecord
    return OS, OC, DO
else
      return null
receive ("Commit",obj),obj ∈ S
lock = false
setup object state
receive ("Abort")
    lock = false
```

6. Implementation

FT-CORBA uses redundancy mechanism to achieve fault-tolerance, replication itself can tolerate process fault and host fault, our voting mechanism focus on maintaining consistency of replicated object and network partition fault.

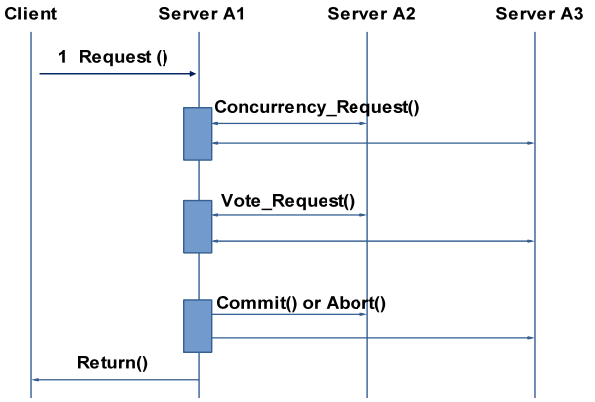

Figure 4. CRVM voting processing

The process of our voting mechanism is illustrated in Figure 4.

1) Firstly Client sends a request to server, OM then chooses a object（maybe a replicated object）to serve it, for instance we can make it through pinging the fastest object;

2) The object who receives the request would start voting mechanism, if the request has been severed then returns null, if not it sends Concurrency_Request to A2 A3 for synchronous control;

3) Send Voting_Request. A1 executes Lock_Request itself after finishing Concurrency_Request, case lock operation successfully, A1 then sends Vote_Request to A2 A3, otherwise, A1 must execute termination protocol, and then do Lock_Request again, if still failed, the operation would end this time;

With executing termination protocol, A1 would send Decision_Request (contain DS and request identifier)to A1 A2 who receive request then reply whether update operation has finished, if finished, A1 then execute Realese_Lock;

A2 A3 would lock itself through Lock_Request when receive Vote_Request, case lock successfully, return its VN SC DS to S, otherwise, execute termination protocol, later lock again, if still failed, return null to A1;

4) Determine majority partition: A1 collects all status replied by A2 A3, and then executes Is_Distinguished to determine whether the network A1 contained in the majority partition. According to the result, it decides to send Commit () or Abort ().

5) A1 returns the result.

The following is an example of CRVM process. We simulated the ATM depositing and withdrawing money operation to show how to ensure the consistency of replicated object and resolve network partition fault, IDL file is shown as follows:

```
interface atmOperate
{
  float inquiry ();                  // balance inquiries
  void deposit(in float num);        // depositing money
  void withdraw(int float num);      // withdrawing money
}
```

We define three operations: inquiry () deposite (in float num) and withdraw (in float num). The server-side object is responsible for deposits withdraws and inquiries business, in order to achieve fault-tolerance, server-side starts three replicated objects A1 A2 and A3, their status can be expressed by balance which we set it as S in following figures. For simplicity, we set an account with its balance $300.00 and had operated 5 times, the distinguished object is A1. So the initial status of A1 A2 and A3 is illustrated in Figure 5.

When a network partition fault occurred between A1 A2 and A3, after being requested by the operation deposit (500.00), A2 gets synchronous with A1 through CRVM while A3 remains constant which is illustrated in Figure 6.

When the fault is removed and a withdraw(300.00) request is sent to A1, since CRVM would find all normal objects to participate voting, A3 can re-join voting group and get synchronous with A1 which is illustrated in Figure7.

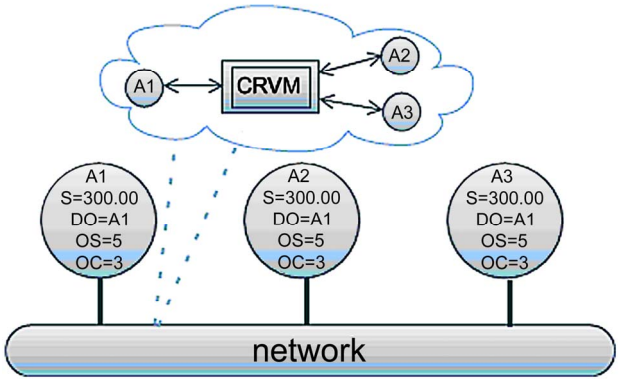

Figure 5. Initial status

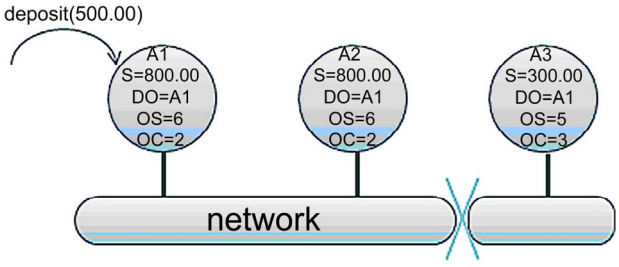

Figure 6. The status after operation deposit (500.00)

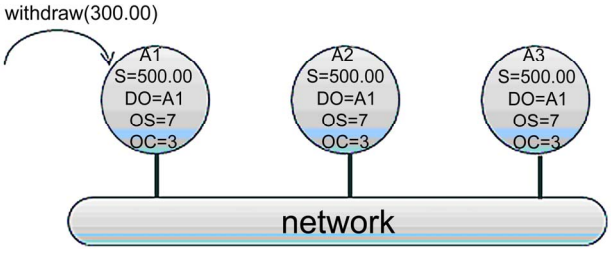

Figure 7. The status after operation withdraw (300.00)

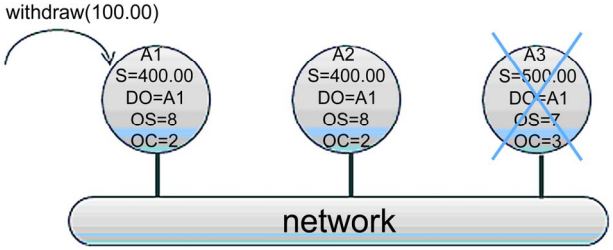

Figure 8. The status after operation withdraw (100.00)

Sending a withdraw (100.00) request to A1 while A3 gets a crash fault, the result is illustrated as Figure 8.

7. Conclusions

Since distributed applications development, fault-tolerance requirement has become more complex. The existed fault-tolerant technology is facing great challenges, for example, how to maintain the consistency of the object replication, how to control concurrent implementation and how to resolve network partition etc.

Despite FT-CORBA standard has been adopted by OMG, there is still a long way to improve it. The real-world applications is more complex and requires more stringent, current standard only provide a framework for fault-tolerance and can only be used for very simple applications. Thus FT-CORBA can not provide a ready solution for complicated applications now.

In this paper we design the CRVM to maintain the consistency of replicated object and resolve network partition problem, we will do further study and improvement on it next step.

8. Acknowledgements

This work is supported by National Fundamental Research Program of China under Grant NO.A1420080190.

REFERENCES

[1] Object Management Group, "Common Object Request Broker Architecture: Core Specification Version 3.0.2 " OMG Technical Committee Document date formal/02 −12−02.

[2] B. Natarajan, A. Gokhale, S. Yajnik, and D. C. Schmidt, "Doors: Towards high-performance fault-tolerant CORBA," Proc. Second Int'l Symp. Distributed Objects and Applications (DOA '00), pp. 39−48, February 2000.

[3] P. Felber and P. Narasimhan, "Experiences, strategies and challenges in building fault-tolerant CORBA Systems," IEEE Transactions on Computers, Vol. 53, No. 5, May 2004.

[4] S. Mullender, ed., Distributed Systems, Addison-Wesley, 1993.

[5] P. Narasimhan, "Transparent fault tolerance for CORBA," Ph.D.dissertation, University of California, Santa Barbara, December 1999.

[6] S. Maffeis, "Adding group communication and fault-tolerance to CORBA," Proc. Object-Oriented Technologies, June 1995.

[7] Kenneth Birman, Robbert, and van Renesse, "Reliable distributed computing with the Isis Toolkit," IEEE Computer Society Press, Los Alamitos, 1994.

[8] P. Narasimhan, L. E. Moser, and P. M. Melliar Smith, "Eternal-a component-based framework for transparent fault-tolerant CORBA," Software -Practice and Experience, pp. 771−788, 2002.

[9] M. Cukier, J. Ren, C. Sabnis, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr, and R. E. Schantz, "AQUA: An adaptive architecture that provides Dependable Distributed Objects," IEEE Symposium on Reliable and Distributed Systems (SRDS), pp. 245−253, October 1998.

[10] P. Narasimhan, L. E. Moser, and P. M. Melliar Smith, "Strong replica consistency for fault-tolerant CORBA applications," Object-Oriented Real-Time Dependable Systems, 2001. Proceedings, Sixth International Workshop, pp. 10−17, January 2001.

[11] R. van Renesse, K. P. Birman, and S. Maffeis, "Horus: A flexible group communication system," Comm. ACM, Vol. 39, No. 4, pp. 76−83, April 1996.

[12] K. P. Birman and R. V. Renesse, "Reliable distributed computing with the Isis toolkit," IEEE Computer Society Press, March 1994.

[13] S. Jajodia and D. Mutchler, "Dynamic voting," ACM SIGMOD International Conference on Management of Data, San Francisco, pp. 227−238, 1987.

[14] S. Jajodia and D. Mutchler, "Dynamic voting algorithms for maintaining the consistency of a replicated database," ACM transactions on Database Systems, Vol. 15, No. 2, pp. 230−280, June 1990.

