A New Voting Approach to Fault-Tolerant CORBA

José-Carlos Martínez-Vélez   Jehan-François Pâris

Department of Computer Science
University of Houston, Houston, TX 77204-3475

jcv@shellus.com, paris@acm.org

Abstract

We propose a new voting protocol for a CORBA-compliant replication service as a suitable solution to the strong fault-tolerant requirements of the latest generation of applications. Our new replication service works as a highly available collection of CORBA objects providing transparent support of the replication services to the client applications. We differ from other FT CORBA proposals in that we bypass the read-one/write-all model and implement instead a dynamic voting protocol. To reduce space overhead, we replace some replicas with witnesses *that only contain the metadata. We also reduce the message traffic overhead by using* cohort sets, *thus eliminating the need for metadata updates at every client access.*

1 Introduction

The most promising approach to solve the distribution, interface and integration problems for leading-edge applications is the OMG's Common Object Request Broker Architecture (CORBA). It provides mechanisms for the development of component-based distributed applications that satisfy requirements for interoperability, efficiency and flexibility. In spite of its success as a component framework, the issue of fault-tolerant CORBA is in its infancy. The limited specification that has been adopted by the OMG does not address yet all the strong fault tolerance requirements of mission-critical applications.

Current research on fault-tolerant CORBA has focussed on using object-group communication mechanisms [3, 8, 10] to replicate the critical objects of each application. These engines maintain the consistency of all members of a group of replicated objects by providing reliable delivery of messages among all the replicas. The mechanism is straightforward but it cannot handle network partitions without using complex configurations that recognize gateways as active members of the processing groups. This is a limitation that could affect the replicated object's availability by preventing full access to the replicas or by degrading its performance.

We present a new dynamic voting protocol that provides access to the replicated object during network partitions in a natural way. This new protocol is based on object replication control and assures the consistency of the set of replicas through quorum computations. In the absence of Byzantine failures, these computations guarantee the correctness of the current states of the replicas. This robustness is the main advantage of replication protocols over group communication protocols.

The replication control protocol we present is the first one to combine the respective advantages of dynamic voting, witnesses and voting without version numbers. As our Markov analysis shows, combining these two techniques results in a more efficient replication control protocol that provides higher data availabilities than static voting protocols and protocols that do not include witnesses.

Since this is the first study investigating the possibility of using a voting protocol to provide fault-tolerant CORBA services, our proposal describes how our dynamic voting protocol could be inserted within the current FT CORBA specification. We also present some experimental data about a simple prototype version of our proposed replication control protocol. They show the superiority of the voting approach over the object-group communication mechanisms.

The following section presents an overview of the related work on FT CORBA, as well as on replication control protocols. Section 3 introduces the new voting protocol implemented by our framework and its availability analysis. An overview of the possible architecture for FT CORBA with voting is also shown with a description of a prototype in section 4. Finally, the concluding remarks are presented in section 5.

2 Related work

Current research for FT CORBA [11] has strictly focussed on object-group communication. As a result, voting protocols have been totally ignored. Here, we cover the related research giving the foundation for our FT CORBA implementation using a new object replication voting protocol which eliminates the use of version numbers/timestamps [14] and reduces the overhead by maintaining *witnesses* [12] and requiring only a majority of replica replies.

2.1 Replication models

The configuration of the different types of fault tolerance approaches [11] is based on the quality of service (QoS) required for the CORBA objects. For our purposes, we define QoS in terms of three variables: degree of availability, degree of reliability and performance degradation. Availability is measured as the ratio of the available time over the total time, and reliability as the probability of no failures over a time interval. For example, some CORBA objects are supposed to actively exist for long periods of time (active configurations). Others do not have that requirement (passive configurations) and are only active, together with the application objects, for short periods of time. Different types of fault-tolerance (active or passive), through replication, make the distinction between these three objectives. Other factors must also be considered in some fashion, for example, the data loss due to replica failures; scalability, meaning how many replicas can be used and how many concurrent clients the service can handle. Resource usage is another measure for a fault-tolerant algorithm. It simply measures how many resources are needed for the algorithm to work properly. We focus on CORBA objects with strong fault tolerance requirements. To satisfy the requirements we propose an active replication protocol. Our emphasis is on object replication control algorithms supporting high availability, high reliability and low performance impact on the application objects. We believe that the current proposals do not achieve these goals in their entirety.

Current object-group communication implementations follow the *read-one/write-all* protocol [7]. This protocol specifies that all replicas must be updated at every access. It assumes, therefore, that all replicas will always remain identical, providing efficient read access. This very intuitive protocol suffers from a major disadvantage: since every write must update all replicas, it makes replicated objects less available than non-replicated objects. Our proposal overcomes this disadvantage by requiring only a majority of replicas to be consistent. As we describe below, not only do we propose a protocol optimizing this requirement, we also optimize our multicast by using an updated version of the quorum-oriented communication mechanism [5].

2.2 Voting protocols

Voting protocols guarantee the reliability of the replicated objects by disallowing read and write accesses that cannot collect a quorum of replicas. The best known voting protocols include *majority consensus voting* [15], *weighted voting* [4], *dynamic voting* [1], *dynamic voting, dynamic-linear voting* [6], and *voting with witnesses* [12]. Each one of these protocols relies on quorum computations to provide reliability for the objects. To make such guarantees, the protocols follow two quorum requirements:

$$W_j \cap W_k \neq \varnothing \quad \forall\, j, k \tag{1}$$

$$R_i \cap W_j \neq \varnothing \quad \forall\, i, j \tag{2}$$

where W_j and W_k represent the two arbitrary write quorums and R_i an arbitrary read quorum. These requirements prevent conflicting updates and guarantee that every read will necessarily access at least one of the most recently updated objects.

All voting protocols can be broadly divided into two classes, namely *static protocols* and *dynamic protocols*. Static protocols have *fixed quorums*. Hence they must prevent object updates whenever they cannot gather the votes of a majority of the object replicas. Dynamic protocols adjust their quorums to keep track of changes in replica availability: whenever a write quorum of replicas notices that it cannot contact the remaining replicas, it *excludes* these replicas from quorum computations and computes new read and write quorums satisfying conditions (1) and (2). As a result, dynamic voting protocols can continue to provide write access after the successive failures of a majority of the object replicas. To avoid inconsistent updates, we must prevent excluded replicas from participating in any quorum until they are formally *reincluded* into the current set of voting replicas (the *majority partition* or *majority block*).

Both static and dynamic voting protocols require a minimum of three replicas to improve upon the availability of an non-replicated object. However one of these replicas can be replaced by a *witness* [13]. Witnesses are very small entities that hold no data but maintain enough metadata to identify what it believes to be the most recent version of the object. This information could be a *timestamp* containing the time of the latest update or a *version number*, which is an integer incremented at each object update.

Voting without version numbers [14] is a more recent implementation of the dynamic-linear voting protocol that does not require version numbers. Instead, it keeps a *cohort set* for each replicated object, identifying the replicas that participated in the last write operation. The protocol guarantees that all sites sharing the same cohort set are identical, greatly simplifying the replica metadata and eliminating the need for version numbers.

2.3 FT CORBA current approaches

As we already mentioned, the OMG just published the revised joint submission for FT CORBA. Current research has focused on different models to provide FT CORBA objects for high availability requirements. These models emphasize object-group communication as in group multicast. As we mentioned before, the problem with the current group communication implementations is that they act under the read-one/write-all protocol, where all the replicas of the group are updated at every write access. Many of the application objects supporting strong fault tolerance requirements not only require high availability and reliability, but also performance transparency. To support these requirements for such active CORBA objects, we need a replication service minimizing performance degradation at the same time it satisfies the other fault tolerance requirements.

In the following section we introduce our new object replication voting protocol, including an overview of its implementation architectural pattern.

3 CORBA-compliant object replication components

We consider a CORBA framework maintaining replicated objects at distinct nodes of a computer network. Replicating active objects introduces special challenges as node failures and network partitions are likely to result in inconsistent replica updates. Replication protocols mediate with these challenges to guarantee the consistency of the replicas while providing the highest possible object availability and reliability.

Our goal is to provide a mechanism to build highly available CORBA-compliant applications with replicated critical components. Neither the CORBA standard or conventional/proprietary implementations of CORBA directly address fault tolerance through a voting replication service. The current approaches work with the problem using group communication mechanisms focusing on good delivery of messages to the replicated objects. We believe that this mechanism is not very efficient because it necessarily increases the number of requests between replicas and clients. Our implementation follows a more efficient approach by mediating the accesses to the replicated objects using a quorum-oriented multicast mechanism to access only a majority of the replicas instead of all of them. It also minimizes the network traffic as well as the request/reply ratio between the replicas and the clients. In addition, it maximizes the efficiency of each communication access with the replicas by making requests only when they are really necessary and providing parallel access requests.

3.1 A new object replication control protocol

As we mentioned earlier, our major concern was to select a replication control protocol that would allow updates in the presence of network partitions without endangering the consistency of the replicated objects. This immediately excluded *available copy* protocols because they do not work correctly in the presence of communication failures. We also rejected *object group communication* protocols because none of them allows updates in the presence of network partitions without requiring explicit monitoring of each communication path. Thus our sole possible choice was to select a voting protocol.

Our second objective was to minimize the overhead of our replication control protocol. This was especially important because voting protocols are notoriously bad in that respect. First, they require a minimum of three replicas to improve upon the availability of an non-replicated object. Second, each access to a replicated object involves the collection of a quorum of replicas. Replacing some replicas by witnesses only reduces the storage overhead of voting protocol but does little to reduce their communication overhead.

Our solution to this problem is a novel implementation of witnesses using cohort sets instead of version numbers. Each witness will consist of a bitmap identifying (a) which replicas have participated in the last write operation and (b) which witnesses are included in the current majority block. A witness for a replicated object with n replicas and m witnesses will thus occupy exactly $n + m$ bits. While version numbers had to be updated at every write operation, cohort sets are only updated at failure detection and/or recovery time.

More formally, our protocol considers replicas and witnesses that reside on different sites of a computer network. Sites can fail and recover as well as be temporarily unable to communicate with other sites. We assume that all sites will either operate correctly or stop exchanging messages with the other sites. Hence Byzantine failures are specifically excluded.

The majority block of a replicated object is the set of witnesses and replicas—let us call them voting entities—that are currently allowed to participate in quorum computations. Any change in the composition of the majority block will require a write quorum and every write quorum will always include at least one current replica.

We associate with each voting entity a *cohort set* representing the set of replicas that (a) were current after the last write or replica recovery in which the voting entity participated and (b) were or are still in the last majority block in which the voting entity participated. Cohort sets thus play a double role: they keep track of which replicas are current and which voting entities are in the current majority set. Hence any update that cannot modify the state of *all replicas* in the current cohort set cannot proceed without simultaneously updating the cohort sets of all the current replicas and electing a new majority block. Hence all replicas that are in the current majority block will always have *identical cohort sets.*

Witnesses do not normally participate in read or write operations unless the operation involves the election of a new majority block. Our protocol assumes that witness failures and recoveries will ultimately be reflected in the composition of the current majority block but does not rely on the instant detection of witness failures or recoveries. As a result, it may happen that the cohort sets of the voting entities in the current majority block contain witnesses that have failed or omit witnesses that have recovered.

To better understand the relationship between cohort sets and the majority block, consider the following scenario. Let O be a replicated object O consisting of two replicas (A_O, B_O) and three witnesses (X_O, Y_O, Z_O). An initial quorum satisfying the two conditions:

$$W_j \cap W_k \neq \varnothing \ \ \forall j, k$$
$$R_i \cap W_j \neq \varnothing \ \ \forall i, j$$

is $|R_i| = |W_j| = 3$.

As long as all sites maintaining voting entities are available and up-to-date, their corresponding cohort set memberships will remain identical and equal to:

$$C_i = \{A_O, B_O, X_O, Y_O, ZO_R\} \ \forall_i \mid i = \{A_O, B_O, X_O, Y_O, Z_O\}$$

The majority block is initially defined with the same configuration as the cohort sets because all the sites are available.

$$MajBlk = \{A_O, B_O, X_O, Y_O, Z_O\}$$

Assume now that the following events happened in the order specified:

a) *Witness X_O becomes unreachable after a network partition:* the cohort sets of the replicated object remain unchanged until a new majority block is elected.

b) *Write access is requested to update O:* the witness failure continues to remain undetected and the cohort sets of the replicated object remain unchanged.

c) *Replica A_O becomes unavailable after a site failure:* both failures remain undetected and the cohort sets of the replicated object remain unchanged.

d) *Write access is requested to update O:* the write protocol detects the failure of replica A_O and requests the election of a new majority block. During the election process, the failure of witness X_O is finally detected and the resulting majority block excludes both replica A_O and witness X_O. The resulting object configuration is:

$$C_i = \{B_O, Y_O, Z_O\} \quad i = B_O, Y_O, Z_O$$
$$MajBlk = \{B_R, Y_O, Z_O\}$$
$$(C_A = C_X = \{A_R, B_R, X_O, Y_O, Z_O\})$$
$$|R_i| = |W_i| = 2$$

This example immediately suggests a small protocol improvement. Observe that a witness failure is never detected until after a replica fails and a write access to the object is requested. Hence failed witnesses could remain in the current majority block for days or even weeks. This is clearly an undesirable situation because it will make read and write quorums harder to achieve. The simplest solution is to send out periodical *probes* checking the status of all voting entities and initiating the election of a new majority block whenever the need occurs. These probes would also speed up the detection of replica failures whenever the replicated object is not frequently modified.

It may happen that successive site failures bring a replicated object in a state where there are not enough replicas to satisfy a write quorum. Updates to the replicated object will then be disabled until enough replicas have recovered to reconstitute a write quorum within the last majority block.

3.2 Availability analysis

Availability is the most common measure of fault tolerance for repairable systems that are expected to remain operational over a long period of time. It is traditionally defined as the fraction of time a system is operational. In the case of replicated data objects, the availability of a replicated object represents the fraction of time that the consistency control protocol will allow access to the object.

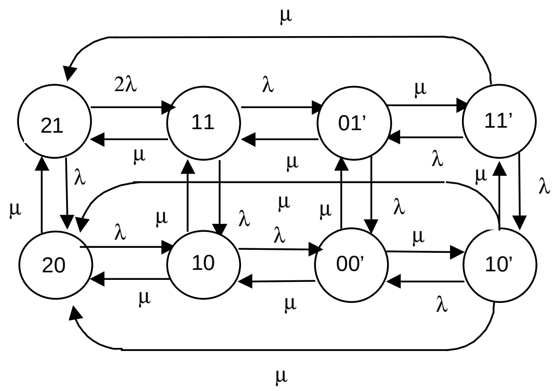

Figure 1. State transition diagram for two replicas and one witness

Our system model consists of a set of sites with independent failure modes connected via a network whose failure rate is negligible. When a site fails, a repair process is immediately initiated at that site. Should several sites fail, the repair process will be performed in parallel on those sites. We assume that site failures are exponentially distributed with mean λ, and repairs are exponentially distributed with mean μ. The system is assumed to exist in statistical equilibrium.

Although the assumption of an independent failure rate λ is reasonable if the sites have independent power sources, the assumption of exponential repair times is harder to defend on general grounds. However, both hypotheses are necessary for a steady-state analysis to be tractable and have been made in most probabilistic studies of the availability of replicated data [7, 12-14].

Figure 1 represents the state transition diagram for two replicas and one witness managed by a dynamic-linear voting protocol using cohort sets. For the sake of simplicity, we will assume that the replicated object is frequently updated and that changes in the status of the two replicas and the witness are quickly reflected in the composition of the majority block. We will further assume that the initial read and write quorums are two voting entities out of three.

We can then represent the state of the replicated object by the number of replicas and witnesses that are available at any time. For instance, state <21> will represent the state of the replicated object when the two replicas and the witness are operational. Edges will represent transitions between states and edge labels represent the rate probabilities of these transitions. We can immediately identify two kinds of transitions, namely *failure transitions* and *recovery transitions*. Failure transitions correspond to the failure of one of the three voting entities: their transition rate is either λ or 2λ. Conversely, recovery transitions represent the recovery of a voting entity that had previously failed and their transition rate is always equal to μ.

Starting from state <21>, we can see that a failure of the witness would bring the replicated object into state <20>. Since there are exactly two replicas available, the protocol will elect a tie-breaking rule and give to one replica (the *dominant replica*) precedence over the other (the *weak replica*). A failure of the weak replica would bring the object into state <10> where updates are still possible even though only one of the three voting entities remains accessible. Conversely a failure of the dominant replica would move the replicated object into state <10'>. State <10'> is an unavailable state, as the second replica cannot form a write quorum. The replicated object will then remain unavailable until it returns to state <20> or to state <11>. Looking back at state <10>, we see that a failure of the last available replica would bring the system to state <00'> where all three voting entities are unavailable. There are three recovery paths possible from state <00'>:

a) the witness recovers first and brings the replicated object into state <01'>; the replicated object remains unavailable as long as neither of its two replicas can be accessed;
b) the dominant replica recovers first and returns the object to the available state <10>; and
c) the dominated replica recovers first and brings the object back to state <01'>.

Starting again from state <21>, we see that a failure of either of the two replicas would bring the replicated object in state <11>. State <11> is an available state and its read and write quorums are exactly one replica. A failure of the witness would bring the object into state <10> where a failure of the sole remaining replica would move it to the unavailable state <01>. A failure of the replica that failed first would bring the object into state <11'>. Note that state <11'> is an unavailable state because the replica that recovered was previously excluded from the majority block when the object entered state<11>. Conversely, the recovery of the replica that failed last would reconstitute the last majority block and bring the replicated object back to state <11>.

To obtain the availability of the replicated object, we first solve the equilibrium conditions for the eight system states and compute then the probability of being in any of the four accepting states, namely <21>, <20>, <11> and <10>. After simplification we obtain:

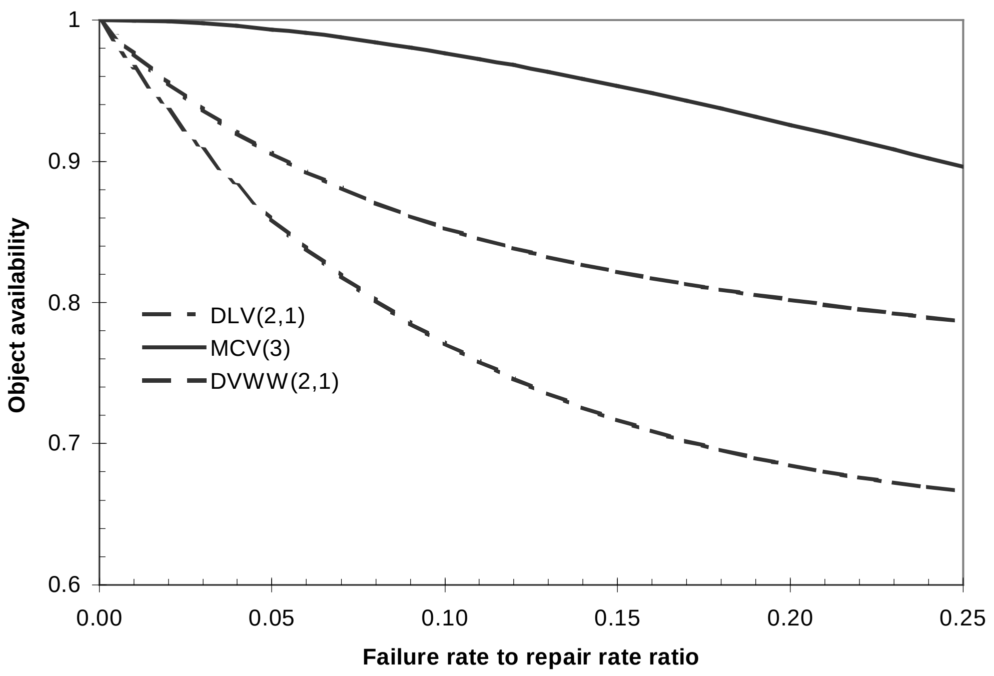

Figure 2: Compared availabilities of replicated objects with three entities managed by dynamic voting with witnesses, majority consensus voting and voting with witnesses.

$$A_{DLV}(2{,}1) = \frac{2 + 9\rho + 17\rho^2 + 11\rho^3 + 2\rho^4}{(2 + 3\rho + 2\rho^2)(1+\rho)^3} \qquad (3)$$

where $\rho = \lambda / \mu$ is the failure rate to repair rate ratio.

Note that equation (3) is also the availability of a replicated object consisting of three replicas managed by the DLV protocol. Hence replacing one of the three replicas managed by the DLV protocol by a witness has no effect on the availability of the replicated object.

Figure 2 compares the availability of two replicas and one witness managed by our protocol (DVWW(2, 1)) with those of a three replicas managed by majority consensus voting (MCV(3)) and two replicas and a witness managed by a static voting protocol (VWW(2, 1)). We selected a range of values for ρ between 0 and 0.25. The first value corresponds to a voting entity that never fails while the second corresponds to a voting entity that is available 80 percent of the time. As one can see the our protocol performs much better than its static counterparts even though it storage requirements are two thirds of those of three replicas managed by majority consensus voting.

3.3 Architectural model of our new object replication components

As we stated before, the OMG's joint revised submission does not properly support active replication with voting protocols. It simply assumes the only active replication style is through a group communication protocol, although it recognizes the active style with voting will be needed as an extension. Also notice that network partitions are not well supported by the specification. Our proposition supports network partitions by default since it is based on a voting algorithm.

We can adopt the current FT CORBA specification to the voting protocols. This is possible by using some schemes specifically for group communication for other, slightly changed, purposes. Also, as expected, the specification provides support for the definition of new fault tolerance properties that apply to specific replication styles. It is possible to define properties that apply to all object groups defined as voting object sets. We are using the TAG_GROUP component to define the voting sets (majority block and cohort sets). This component provides the flexibility for the dynamic part of our protocol as well as the creation/deletion of entities and the upgrading/downgrading of witnesses and replicas respectively. The purpose of the group version can be slightly changed to represent our majority block and/or the cohort sets. It is defined as a Java integer that we can use for our bitmap implementation.

Our implementation uses the TAG_INTERNET_IOP profile to address gateways, consequently communicating with each of the voting set members throughout our new object replication voting protocol. A quorum multicast protocol is used by the gateway to provide a reliable, totally ordered, message delivery service. We also implemented the Infrastructure-Controlled membership style since it is the style that gives the protocol more control producing more accurate evaluation results.

In summary, we have introduced some of the aspects of our implementation and how to map them from the current FT CORBA specification to a voting scheme. Our replication manager uses the FT::ACTIVE_WITH_VOTING as the ReplicationStyle property, the FT::MEMB_INF_CTRL property to represent the membership style and for the consistency style we implemented the FT::CONS_INF_CTRL, among other specified/proprietary properties. We had to simulate some properties and IIOP extensions since no current ORB supports some of the newly specified semantics. We intend to present our results in a future paper.

4 Implementation issues–a prototype

We present preliminary results of some implementation issues we will have to address at a future time. In this section we study the performance of CORBA-compliant applications executing the group communication algorithm and a simple prototype version of our proposed replication control protocol. A concrete example is discussed and the results of the study are introduced.

4.1 A concrete example

Our preliminary hypothesis is that our algorithm will perform better than the initial/current FT CORBA solution algorithms, while maintaining satisfactory object availabilities. We claim that reducing the network overhead by substituting some of the replicated entities with witnesses and reducing the data sent throughout the network will increase the performance drastically during a period of time. We recognize that, in a sense, all the processing in each replica is virtually in parallel, but since we are reducing the work that must be performed for a reply to be accepted, we achieve better results.

The objective of this initial prototype is to prove that our assumptions are correct. We want to verify that reducing the number of client/server requests actually reduces the overhead in the network and performs faster. At the same time, the size of the data passed between the client applications and the replica servers also affects the performance, as well as the network overhead and the throughput.

We ran our client/servers setup in different situations to get a reading on the performance of both algorithms, using CORBA static method invocations. Our client performs 1000 access requests to the replicated object during the elapsed time. Then the program calculates the average response time in milliseconds by dividing the elapsed time by the number of request. This way we measure, under the CORBA environment, the access request/reply differences between the group communication algorithm and our initial prototype.

In the case of the group communication protocol, the algorithm passes a small object among the client and the servers, which contains a simple attribute of type integer. This same object, in addition to the small metadata entities representing the cohort sets and the majority block, are used for the prototype.

Our test case consists of five entity servers and one client application. Our prototype uses the minimum quorum requirements, which are two replica entities and three witnesses. We created three different scenarios, (1) client and servers exist in the same host, (2) client and servers execute in two, heterogeneous hosts, and (3) client and servers execute in multiple, heterogeneous hosts across the network, each process running on its own host.

4.2 Performance

The client application making all the replicated object access requests always executes in the same machine, a 266MHz Pentium II with 64 MB RAM running Windows NT 4.0. The servers communicate with the client across a 10 Mbit/s Ethernet Token Ring, executing on either a Sun Enterprise 2 workstation running Solaris 2.6, or a Sun Ultra 2 running the same version of Solaris. Each Sun workstation has 256 MB of memory. Both client and servers are Java applications, using the Java 2 SDK's ORB.

As Table 1 indicates, the prototype of our algorithm performs faster than the group communication protocol under the same conditions. The calculated margin of error is about 10 percent, even using the same setup. Notice that remote invocations are slightly faster than local ones, this is because in the remote scenarios the servers are running in parallel. The prototype simply outperforms the simple group communication implementation. Even though the servers are independent processes running in independent hosts (scenario 3), the probabilities of jobs being delayed are higher with five servers than with three. This clearly delays the reply to the client, making it, under the prototype implementation, accessible to the client faster. There are other reasons for this behavior, those will be studied and examined in future research.

Table 1. Three case scenarios for simple implementations of our prototype and a group communication algorithm

Scenario	**(1)**	**(2)**	**(3)**
Our Prototype	6.59 ms	3.525 ms	2.434 ms
Group Communication	10.656 ms	5.177 ms	3.285 ms

An interesting observation occurred when we interchanged an Ultra 2 for an Ultra 1, keeping the rest of the configuration unchanged. The simple implementation of the group communication protocol performed within 14.391 ms., under the test case scenario (3). The prototype of our algorithm maintained the original values, averaging 2.63 ms. The reason is simply because the group communication must wait for all the servers to reply, while our prototype simply waits for the majority of servers to reply. This is a considerable difference between both test implementations.

5 Conclusion

None of the existing proposals for managing replicated CORBA objects can allow write access to the replicated object in the presence of network partitions without requiring explicit monitoring of each communication path. We have presented a new architecture for fault tolerance in large distributed systems with replicated CORBA objects. Unlike existing proposals, our architecture uses voting to guarantee the consistency of the replicated data. Hence it allows updates in the presence of any network partitions without making any assumptions on the topology of the communication subsystem. Our architecture is also the first to integrate several recent advances in voting protocols, namely witnesses, dynamic-linear voting and voting without version numbers. As a result, it requires fewer replicas and incurs less communication overhead than the best existing replication control protocols.

We have also presented a testbed implementation of our architecture within the FT CORBA framework with some simulated components as well as a Markov analysis of the availability of a replicated object managed by our dynamic-linear voting protocol with witnesses.

More work needs to be done to evaluate the reliability of replicated objects managed by our new protocol, complete the implementation of our testbed and compare its performance with those of FT CORBA object-group communication implementations.

Acknowledgments

This research was supported in part by the Texas Advanced Research Program under grant 003652-0124-1999.

References

[1] D. Davcev and W. A. Burkhard, "Consistency and Recovery Control for Replicated Files," in Proc. *10th ACM SOSP Symposium*, 1985, pp. 87-96.

[2] A. Duda, "Analysis of Multicast-based Object Replication Strategies in Distributed Systems," in Proc. *13th ICDCS Conference*, 1993, pp. 318-338.

[3] P. A. Felber and R. Guerraoui, "The Implementation of a CORBA Group Communication Service," *Theory and Practice of Object Systems*, vol. 4, no. 2, pp. 93-105, 1998.

[4] Gifford, "Weighted Voting for Replicated Data," in Proc. *7th ACM SOSP Symposium*, 1979, pp. 150-161.

[5] R. A. Golding and D. D. E. Long, "Quorum-Oriented Multicast Protocols for Data Replication," Technical Paper UCSC-CRL-91-21, University of California, Santa Cruz, 1991.

[6] S. Jajodia and D. Mutchler, "Enhancements to the Voting Algorithm," Proceedings 13th VLDB Conference, 1987, pp. 399-405.

[7] D. E. Long and J.-F. Paris, "A Leaner, More Efficient, Available Copy Protocol," in Proc. *8th IEEE SPDP Symposium*, 1996, pp. 400-407.

[8] S. Maffeis, "Adding Group Communication and Fault-Tolerance to CORBA," in Proc. *USENIX COOTS Conference*, 1995, pp. 135-146.

[9] B. E. Modzelewski, D. Cyganski and M. V. Underwood, "Interactive-Group Object-Replication Fault Tolerance for CORBA," in Proc. *3rd USENIX COOTS Conference*, 1997, pp. 241-244.

[10] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "Exploiting the Internet Inter-ORB Protocol Interface to Provide CORBA with Fault Tolerance," in Proc. *3rd USENIX COOTS Conference*, 1997, pp. 81-90.

[11] Object Management Group, "Fault Tolerant CORBA", JRS, ftp.omg.org/pub/docs/orbos/99-10-05.pdf, Object Management Group, Framingham, Mass., (1998).

[12] J.-F. Pâris, "Voting with Witnesses: A Consistency Scheme for Replicated Files," in Proc. *9th ICDCS Conference*, 1986, pp. 606-612.

[13] J.-F. Pâris, D. D. E. Long, "Voting with Regenerable Volatile Witnesses," in *Proc. 7th ICDE Conference*, 1991, pp. 112-119.

[14] J.-F. Pâris, D. D. E. Long, "Voting without Version Numbers," in *Proceedings 1997 IPCCC Conference*, 1997, pp. 139-145.

[15] J. Seguin, G. Sergeant, P. Wilms, "A Majority Consensus Algorithm for the Consistency of Duplicated and Distributed Information," in *Proceedings 1st ICDCS Conference*, 1979, pp. 617-624.