A Quorum Based Approach to CORBA Fault-Tolerance

Gregory Chockler* Danny Dolev* Dahlia Malkhi*

1 Introduction

In this paper we propose an approach based on quorum replication for providing fault-tolerance in middlewares compliant with the Common Object Request Broker Architecture (CORBA) [OMG99]. While various solutions for supporting replication in CORBA exist, all of them are based on techniques borrowed from the group communication world. In this paper we demonstrate that quorum based replication is a viable alternative to this approach, offering improved scalability, availability and load balancing. In addition, quorum systems can be customized according to a wide range of parameters, e.g., to mask Byzantine failures [MR98], thus offering the object creator flexibility to choose the replication framework most suitable for the application needs.

The issue of fault-tolerance support in CORBA has received significant attention in recent years, both in research and standardization. The recently published Fault-Tolerant CORBA (FT-CORBA) specification [OMG00] is a culmination of several years of intensive research dedicated to this topic. Due to the importance of compliance with this standard on the one hand, and the need to reflect on potential improvement to it on the other, we dedicate attention in this paper to the most important aspects of this standard and analyze its applicability in our context. We also discuss the modifications we deem necessary to the standard that would allow implementations based on the quorum replication approach.

Our proposed infrastructure utilizes a new total ordering protocol [CMR01] that we recently developed for maintaining replica consistency using quorum replication. The protocol is entirely client driven and is built on top of a simple and efficient distributed mutual exclusion primitive. Our claims of scalability and high availability derive from the properties of the total ordering protocol and the fundamental features of quorum based replication. In particular, the total ordering protocol does not rely on system reconfiguration in case of failures for ensuring its progress. Instead, it relies on the inherent fault-tolerance of quorum systems, as only a quorum of the replicas needs to be available for the duration of each request. Eliminating constant monitoring of replica failures contributes considerably to the scalability of our solution and results in a more lightweight infrastructure. Additionally, the size of quorums can be surprisingly small, e.g., an order of square root of the total number of replicas. Thus, finding an available quorum is quite realistic in practice, and keeps communication costs low. Finally, a clear separation between the client and the replica side implementations makes our protocol especially suitable for distributed object systems and in particular for CORBA systems. This further reduces the complexity of integration of our protocols at the ORB level.

We are currently implementing the first system prototype that is structured as a CORBA service (see Section 4). In the future, we intend to make it compliant with FT-CORBA as much as possible. This might require implementing parts of our system at the ORB level, as necessitated by the standard (see Section 5).

*School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, ISRAEL.
Email: {grishac,dolev,dalia}@cs.huji.ac.il

2 Related Work

The existing body of research on CORBA fault tolerance is dedicated to augmenting CORBA with support for group communication; for a survey of works, see [AG98]. According to [F98], existing systems can be classified as designed according to one of the following three approaches: the *integration approach*, the *interception approach* and the *service approach*. Below we briefly discuss these approaches and their implications on our system design.

With the integration approach, the ORB is augmented with proprietary group communication protocols. The augmented ORB provides the means for organizing objects into groups and supports object references that designate object groups instead of individual objects. Client requests made with object group references are passed to the underlying group communication layer which disseminates them to the group members. The most prominent representatives of this approach are Electra [LM97] and Orbix+Isis [II94].

With the interception approach, no modification to the ORB itself is required. Instead, a transparent interceptor is over-imposed on the standard operating system interface (system calls). This interceptor catches every call made by the ORB to the operating system and redirects it (if necessary) to a group communication toolkit. Thus, every client operation invoked on a replicated object is transparently passed to a group communication layer which multicasts it to the object replicas. The interception approach was introduced and implemented by the Eternal system [MMN98].

With the service approach, group communication is supported through a well-defined set of interfaces implemented by service objects or libraries. This implies that in order for the application to use the service it has to either be linked with the service library, or pass requests to replicated objects through service objects. The service approach was adopted by Object Group Service (OGS) [FGS98, F98].

Among the above approaches, the integration and interception approaches are remarkable for their high degree of object replication transparency: It is indistinguishable from the point of view of the application programmer whether a particular invocation is targeted to an object group or to a single object. However, both of these approaches rely on proprietary enhancements to the environment, and hence are platform dependent: with the integration approach, the application code uses proprietary ORB features and therefore, is not portable; whereas with the interception approach, the interceptor code is not portable as it relies on non standard operating system features.

The service approach is less transparent compared to the other two. However, it offers superior portability as it is built on top of an ORB and therefore, can be easily ported to any CORBA compliant system. Another strong feature of this approach is its modularity. It allows for a clean separation between the interface and the implementation and therefore matches object-oriented design principles and closely follows the CORBA philosophy.

In general, each of the approaches outlined above can be adopted for implementing quorum based replication. We opted to first introduce quorum replication support using the service approach, and then implement a compatibility layer for FT-CORBA using the integration framework.

3 Quorum based replication

Quorum systems are known tools for increasing the availability and efficiency of replicated services. A quorum system over a universe of replicated servers (simply, replicas) is a set of subsets, called *quorums*, in which each pair of quorums have a non-empty intersection. Replicated services implemented with quorum systems allow an operation to be performed on any available quorum. Intuitively, operation consistency is preserved because of the intersection property, which guarantees that an operation observes the effects of any previously completed operations.

The efficiency of this paradigm is gained from the need to access only a subset (a quorum) of the replicas; and from the reduction in the overall load on any single replica, stemming from the fact that every replica handles only a fraction of the operations. In fact, quorums can be

surprisingly efficient: For a universe of n replicas, there exist quorum constructions with quorums of size $O(\sqrt{n})$ only, in which each replica needs to handle only a $O(\frac{1}{\sqrt{n}})$ fraction of the overall operations. High availability is provided in quorum replication due to the need to access only a live quorum. This comes at a low cost, since no complicated failure handling is involved. Moreover, quorum systems can be designed with various availability levels, and for various failures types, including resilience to arbitrary (Byzantine) failures [MR98]. For a survey of quorum replication techniques, see [Mal99].

Traditionally, quorums have been used primarily for achieving mutual exclusion or for locking, and for emulation of data with weak guarantees (e.g., safe registers). To support atomic data sharing or transactions, replicated data systems employ additional concurrency control mechanisms such as locking, that come with a heavy price: A coordinator of a transaction may fail holding a lock permanently, or otherwise, if old locks are allowed to be overriden, create inconsistency that must be resolved manually.

Recently, we have devised a protocol that provides atomic replication guarantee for replicated objects using quorums [CMR01]. The protocol implements operation ordering providing *linearizable* semantics [HW90]: Informally, this guarantees that all client operations (even those that are invoked concurrently) appear to execute in some serial order. The ordering protocol utilizes a simple and efficient mutual exclusion primitive for leader election (see [CMR01] for details). This protocol forms the foundation of fault-tolerant replication for CORBA, whose design and implementation is described hence.

4 A CORBA quorum replication service

This section describes our design for CORBA quorum replication using the service approach. With the service approach, the implementation of the replication protocol is encapsulated into a number of service objects implementing various parts of the protocol. The overall architecture is depicted in Figure 1.

A replica side service object, called a *replica proxy*, implements the following three interfaces: TOReplica, Mutex and QuorumManager. The TOReplica interface represents the server side functionality of the quorum based total ordering protocol; Mutex supports the interface defined by the mutual exclusion primitive; and QuorumManager supports methods for manipulating (set and get) parameters of the quorum system through which the object is replicated. Turning regular CORBA objects into replicated objects is done simply by inheriting from the replica proxy interface. Note that both Mutex and QuorumManager can be realized as separate CORBA services for improved modularity and flexibility.

Each object group supported by the quorum replication service is identified by a unique name. We use a CORBA Naming Service [OMG98] for resolving object group names to the lists of members. Each newly created object group G is assigned a naming context under the quorum replication service naming context. Consequently, references to the individual members of G are bound to the names under the G's naming context. Note that this is a reasonable solution for the quorum based systems because in such systems a replica set is not supposed to change frequently and in particular does not change in response to failures.

A client side service object, called a *group proxy*, implements the client side functionality of the quorum based total ordering protocol. Group proxy implements an interface called TOClient. This interface consists of a single method: submit(op). This method takes an operation description (operation name, arguments, etc) as a parameter and returns the result of invoking this operation at the target object group.

The group proxy object is created by a service object called a *group proxy factory*. This object supports a GroupProxyFactory interface that among others, supports a createGroupProxy(groupName) method that is used to instantiate a group proxy object for the object group designated by groupName. This method is implemented as follows. First, the naming service is contacted and groupName is resolved into a list of references to individual object group members. Then, a method of QuorumManager is invoked on some group member to discover parameters of the quorum system

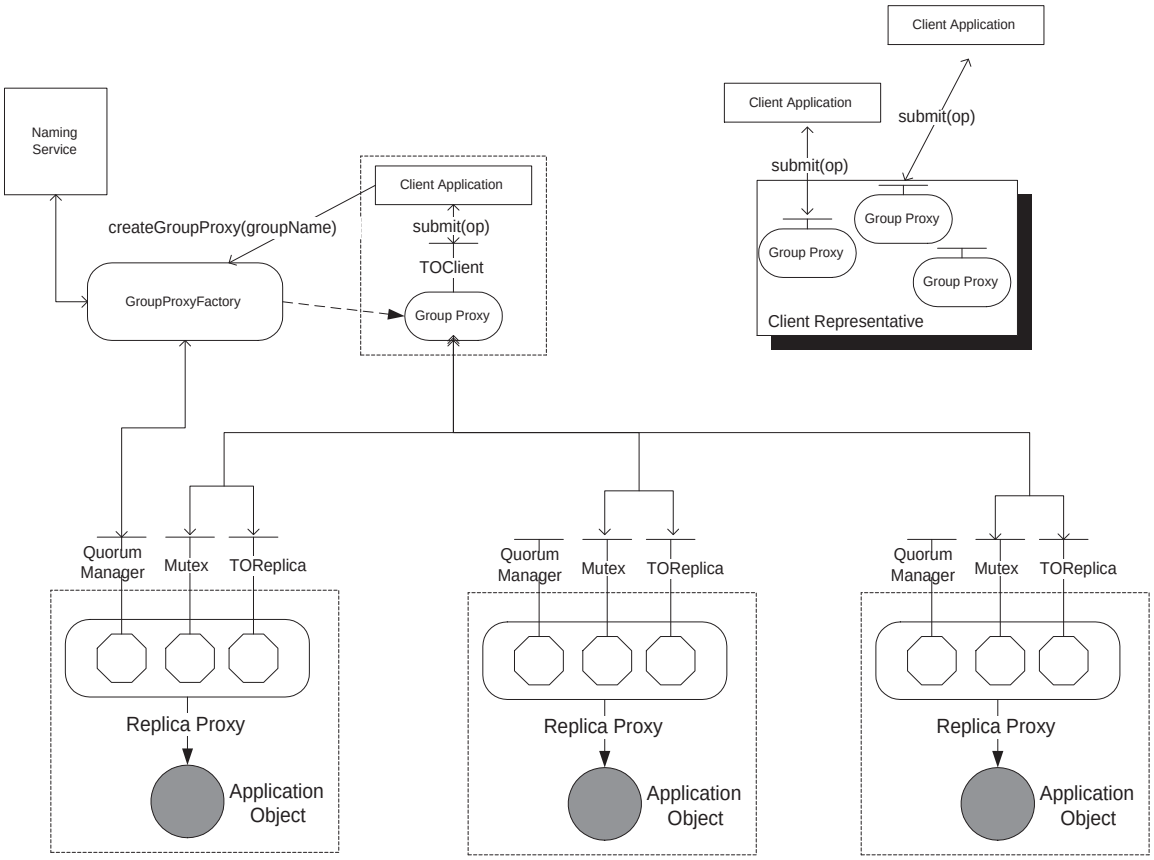

Figure 1: Quorum Replication Service: Basic Components

employed by the group. Finally, a group proxy is instantiated and initialized with the list of replica object references and the quorum system parameters. createGroupProxy(groupName) returns the object reference to a newly created group proxy object.

In order to allow client applications to access object groups through the original object interfaces, the group proxy can be implemented using CORBA's Dynamic Skeleton Interface (DSI). The DSI based group proxy is constructed by the group proxy factory based on the interface definition obtained by invoking the get_interface() method on the object reference of some target group member. The resulting DSI based group proxy is then able to process method invocations made through the original object interface.

Object group proxies can be instantiated either in the address space of a client application or in a separate server called a *client representative*. The advantage of accessing object groups through client representatives is in reduced contention for the mutex as client requests addressed to a particular object group are mediated by the same group proxy object. The disadvantage is in higher communication cost and reduced fault tolerance as the client representative introduces an extra hop between the client application and the object replicas.

Another possible use of client representatives is in providing transparent access to object groups for clients outside the quorum replication domain. This can be done by configuring some client representatives as naming contexts representing the quorum replication domain in the external naming service. In order to support this functionality client representatives have to (at least partially) implement the CosNaming:NamingContex interface of the CORBA Naming Service.

5 FT-CORBA with quorum replication

In this section, we outline how parts of our service-based implementation can be integrated into the ORB to realize an implementation compatible with FT-CORBA. We start with a short overview of the FT-CORBA standard, illuminating issues relevant to our design of FT-CORBA with support for quorum replication.

The Fault Tolerant CORBA Specification. FT-CORBA supports the notion of an *object group* which is used to designate a set of the object replicas. An object group is represented and addressed by an *Interoperable Object Group Reference (IOGR)*. An IOGR is an extension of CORBA's Interoperable Object Reference (IOR), which is the standard way to address CORBA objects. IOGR can be viewed as composed of multiple IORs (profiles) each of which contains a TAG_FT_GROUP component that uniquely identifies the object group it belongs to. Typically, each IOR encapsulated into an IOGR addresses an individual object replica. Alternatively, it might be the IOR of a gateway that allows clients supported by non-fault-tolerant ORBs to access replicated objects.

In order to facilitate management of large scale fault tolerant applications, FT-CORBA introduces the notion of a *fault tolerance domain*. Each fault tolerance domain is managed by a single logical entity called a *Replication Manager*.

The standard provides flexibility in choosing the actual replication mechanisms supporting fault tolerance through a set of *fault tolerance properties*. Among these properties, a *ReplicationStyle* parameter determines the type of replication employed, which could be ACTIVE, COLD_PASSIVE or WARM_PASSIVE. Additional parameters, *MembershipStyle* and *ConsistencyStyle*, control whether membership maintenance and replication consistency are provided by the objects themselves (APPLICATION CONTROLLED) or by an infrastructure of the fault tolerance domain (INFRASTRUCTURE CONTROLLED).

Within a fault tolerance domain, failures of object replicas are monitored and propagated through a hierarchical infrastructure of *Fault Detectors*. Failure notices of the fault detector are collected by a *Fault Notifier*, that communicates fault notifications to the Repication Manager and other objects that registered for such notifications.

The standard recommends mechanisms for implementing some combinations of fault tolerance properties. In particular, it explicitly recommends view synchronous group communication for ACTIVE replication with INFRASTRUCTURE CONTROLLED membership and consistency. This approach for implementing FT-CORBA was indeed carried in the Eternal system [MMN98], the only full implementation of the standard we are aware of. Eternal employs the Totem group communication sytem [MMABL96]. Other styles of replication are left open in the standard for design by implementors. An attempt to partially implement FT-CORBA with WARM-PASSIVE replication in an APPLICATION CONTROLLED manner was recently made in DOORS [GNSY00].

Implementing FT-CORBA using the integration approach. One of the most important building blocks introduced by FT-CORBA is the Interoperable Object Group Reference (IOGR), which is used to address object groups. Here, we should point out that FT-CORBA constrains any implementation which makes use of object groups referenced through IOGRs to necessitate either the integration or the interception approaches, and preclude service level implementation. There are at least two reasons for this: First, CORBA does not specify any means available outside the ORB to create the CORBA::Object datatype from the IOP::IOR datatype and vice versa. Thus, creating an object group and assigning IOGR to it cannot be implemented by an external CORBA service. Second, several implementation modules should be able to manipulate internal IOGR components. This cannot be done at the application level as the CORBA standard dictates that the CORBA::Object datatype has to be opaque outside the ORB.

Thus, compliance with the standard currently requires modification of low-level infrastructure in order to implement quorum based replication for FT-CORBA. The quorum replication service described in Section 4 can be converted into an FT-CORBA compliant implementation by inte-

grating several parts of the service at the ORB level and augmenting the infrastructure with a *Replication Manager (RM)*.

Thanks to a clear separation between the client and the replica side implementations of our protocols, the necessary modifications are confined to the client and servant specific modules while the core ORB mechanisms are left intact. More specifically, the group proxy code becomes the part of the client stub, whereas the code implementing the TOReplica and Mutex interfaces is incorporated into the object skeleton.

To enable the conversion, an implementation of the QuorumManager interface becomes a part of the *Replication Manager* functionality. This requires augmenting the FT-CORBA standard with new properties that are needed for describing quorum systems. As in FT-CORBA, the Replication Manager is responsible for creating object groups and assigning Interoperable Object Group References (IOGRs) to these groups.

In a typical scenario, an object group launcher application contacts an RM and requests it to create object replicas at the specified locations through the create_object() method. In response, an RM creates replicas by invoking auxiliary factories at the requested locations, collects replica references, allocates a unique group identifier, constructs an IOGR and returns it to the launcher. The launcher then invokes other RM methods to associate a quorum system with this IOGR.

Whenever a client application invokes a method on an object group through an IOGR, the client stub first contacts the RM to get a description of the quorum system employed by the object group. It then extracts replica IORs from the IOGR and proceeds with the rest of the protocol. If quorum system parameters could be incorporated into the IOGR, it would save the extra interaction with the RM.

References

[AG98] G. Agha and R. Guerraoui editors. High Availability in CORBA. *Special Issue of Theory and Practice of Object Systems* 4(2): 71–115, April 1998.

[CMR01] G. Chockler, D. Malkhi, and M. K. Reiter. Backoff protocols for distributed mutual exclusion and ordering. In *Proceedings of the 21st International Conference on Distributed Computing Systems*, April 2001. To appear.

[FGS98] P. Felber and R. Guerraoui and A. Schiper. The implementation of a CORBA object group service. *Theory and Practice of Object Systems*, 4(2):93-105, 1998.

[F98] P. Felber. The CORBA Object Group Service. A service approach to object groups in CORBA. *PhD Thesis, Ecole Polytechnique Federale de Lausanne*, 1998.

[GNSY00] A. Gokhale and B. Natarajan and D. C. Schmidt and S. Yajnik. DOORS: Towards High-performance Fault-Tolerant CORBA. In *Proceedings of the 2nd International Symposium on Distributed Objects and Applications (DOA '00)*, OMG, Antwerp, Belgium, September 2000.

[HW90] M. P. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems* 12(3):463–492, July 1990.

[II94] IONA and Isis. An Introduction to Orbix+ISIS. *IONA Technologies Ltd. and Isis Distributed Systems, Inc.*, 1994.

[LM97] S. Landis and S. Maffeis. Building reliable distributed systems with CORBA. *Theory and Practice of Object Systems*, 3(1), 1997.

[Mal99] D. Malkhi. Quorum systems. Chapter in *The Encyclopedia of Distributed Computing*, Joseph Urban and Partha Dasgupta, editors, Kluwer Academic Publishers. To be published. `http://www.cs.huji.ac.il/~dalia/pubs/quorums.ps.gz`.

[MMABL96] L. E. Moser and P. M. Meliar-Smith and D. A. Agarwal and R. K. Budhia and C. A. Lingley-Papadopoulos. Totem: A fault-tolerant multicast group communication system. *Comm. ACM*, 39(4):54-63.

[MR98] D. Malkhi and M. Reiter, Byzantine quorum systems. *Distributed Computing* 11(4):203-213, 1998.

[MMN98] L. E. Moser and P. M. Meliar-Smith and P. Narasimhan. Consistent object replication in the Eternal system. *Theory and Practice of Object Systems*, 4(2):81-92, 1998.

[OMG98] CORBAservices: Common Object Services Specification, December 1998.

[OMG99] Object Management Group. The Common Object Request Broker: Architecture and Specification, 2.3 edition, June 1999.

[OMG00] Object Management Group. Fault Tolerant CORBA Specification, *OMG Document ptc/2000-04-04*, April 2000.