Implementing Flexible Object Group Invocation in Networked Systems

G. Morgan and S.K. Shrivastava
Department of Computing Science, Newcastle University,
Newcastle upon Tyne, NE1 7RU, England.

Abstract

Distributed applications should be able to make use of an object group service in a number of application specific ways. Three main modes of interactions can be identified: (i) request-reply: a client issues a request to multiple servers and waits for their replies; this represents a commonly occurring scenario when a service is replicated; (ii) group-to-group request-reply: a generalisation of the previous case, where clients are themselves groups; and (iii) Peer Participation: here all the members are regularly multicasting messages (asynchronous invocation); this represents a commonly occurring scenario when the purpose of an application is to share information between members, (e.g., a teleconferencing application). Customisation within each class of interaction is frequently required for obtaining better performance. This paper describes the design and implementation of a flexible CORBA object group service that supports the three types of interactions and enables application specific customisation. Performance figures collected over low latency LAN and high latency WAN are presented to support the case for flexibility.

1. Introduction

Distributed applications are increasingly being designed and implemented using CORBA middleware services. In the context of fault tolerant systems, the provision of an object group service is considered highly desirable, as many fault-tolerant distributed applications can be structured as one or more groups of objects that cooperate by multicasting invocations on member objects. A group is defined as a collection of distributed entities (objects, processes) in which a member entity communicates with other members by multicasting to the full membership of the group. The building of group based applications is considerably simplified if the members of a group can multicast reliably and have a mutually consistent view of the order in which events (such as invocations, membership changes) have taken place. In particular, we require the property that a given multicast be atomic: either all the functioning members are delivered the message or none; an additional property of interest is guaranteeing total order: all the functioning members are delivered messages in causality preserving identical order. Management of replicated data for high availability is a good example of the application of groups; here each member process manages a copy of the data, and given atomic delivery and order, it is relatively easy to ensure that copies of data do not diverge. Another example is a collaborative application (e.g., a conference) where members of the group (conference participants) require delivery of messages in causality preserving identical order. Design and development of process groups with the accompanying membership service has been an active area of research [e.g., 1-6]. In the world of distributed objects, process group ideas can be mapped to object groups, and there have been many recent research efforts to enrich CORBA with an object group service [7-14].

Distributed applications should be able to make use of an object group service in a number of application specific ways. Three main modes of interactions can be identified: (i) request-reply: a client issues a request to multiple servers and waits for their replies; this represents a commonly occurring scenario when a service is replicated, fig. 1(i); (ii) group-to-group request-reply: a generalisation of the previous case, where clients are themselves groups; and (iii) Peer Participation: here all the members are regularly multicasting messages (asynchronous invocation); this represents a commonly occurring scenario when the purpose of an application is to share information between members, (e.g., a teleconferencing application), fig.1(ii).

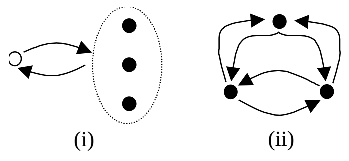


Figure 1: Request/Reply and peer participation

Customisation within each class of interaction is frequently required for obtaining better performance. Consider request-reply interaction between a client and an actively replicated service (in active replication all correctly functioning replicas perform processing). If the client and servers are all connected by high-speed, low latency network, then an efficient way of invoking the replicas would be for the client to multicast to the replicas directly using the underlying total order multicast service (in effect, the client acts as a member of the server group). On the other hand, if the client is separated from servers by a high latency communication path (e.g., WAN, Internet), then this method would be unattractive, and an alternative method that enabled a client to avoid directly multicasting to the replicas would be desirable. Different kinds of customisation might be needed for invoking passively replicated services (in passive replication, a single copy, the primary, performs processing, the remaining members act as backups). Another aspect of customisation is choice of the total order protocol. There are basically two ways of enforcing total order. In the asymmetric version, one of the members of the group assumes the responsibility for the ordering of messages within the group. Such a member is commonly termed a sequencer. In the symmetric version, all the members use a deterministic algorithm for message ordering: this requires the members to exchange, periodically, protocol specific messages amongst themselves to enable message ordering. It has been shown that symmetric protocols tend to be more attractive in situations where all the members are lively, and multicasting regularly (e.g., a conferencing application), so the need for periodically exchanging protocol specific messages just for ordering is eliminated, whereas asymmetric protocols are better in other situations [15]. An application should therefore be able to choose between the two.

In this paper we describe the design and implementation of a CORBA object group service that enables distributed applications to deploy and make use of object groups in a flexible manner as hinted above. We present performance figures collected over low latency LAN and high latency WAN to support the case for flexibility. We have taken a modular approach in the design of the service called the NewTop object group service. First we have implemented a CORBA group communication service that supports overlapping groups (objects can simultaneously belong to many groups) and symmetric and asymmetric total order protocols [13]. Then we have implemented an object invocation layer that uses the multicast service to provide the three specific ways of interacting with object groups mentioned above, together with application specific customisation. As we argue in the next section (related work), existing CORBA object group services do not support all of the functionality or the flexibility provided by our system. In this respect our system represents an advance.

In the next section we describe our layered design and relate our work to existing work on object groups. Section three presents the overview of the NewTop service. Section four describes the design of the invocation layer. Performance figures of our system taken in LAN and Internet settings are presented in section five; these figures illustrate the need for the type of functionality and customisation supported by our system.

2. Approach and Related Work

2.1. Approach

The architecture of our system is depicted in fig. 2. The function of the invocation layer is to use the group communication service to support the three types of object group interactions, namely, request-reply, group-to-group request-reply and peer participation, each one of which can be customised in a specific manner for better performance as indicated earlier and to be discussed further here. The fig. shows how a request-reply interaction between a client and a server group is handled (only a single server is shown). The client application makes its request to the NewTop service; internal to the service, the request is handled by the invocation layer which then uses the group communication service to send NewTop specific message to servers; the message then travels up and down the protocol stack on the server side. The invocation layer employs open and closed groups (see below) to implement request-reply and group-to-group request-reply interactions to enable clients to obtain good performance in high latency as well as low latency networks.

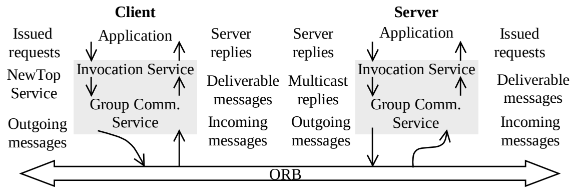


Figure 2: System architecture

Closed group - A client is considered a member of the server group and multicasts requests to each member of the server group directly. Closed groups are appropriate when clients and servers are connected by low latency communication paths (e.g., a LAN).

Open group - A client is not considered a member of the server group and issues requests to just a single member of the server group (that then multicasts the request within the group). Unlike the closed group, clients do not participate in group communication protocols as members of the server group. This makes the open group approach more suitable for use in cases where clients are separated from servers by high latency networks (e.g., a WAN).

The invocation layer achieves open and closed group approaches to client/server group interactions via overlapping of groups. A single group containing members that support some service is identified as a server group. Clients wishing to access the service provided by a server group create a group containing themselves that overlaps with (shares membership of) the server group. A group that contains clients and servers is termed a client/server group. To satisfy open and closed groups, the overlapping of client/server and server groups may be achieved thus: closed group - client/server group contains the client and all the members of the server group (fig. 3 (i)); open group - client/server group contains the client and only one member of the server group (fig.3(ii)).

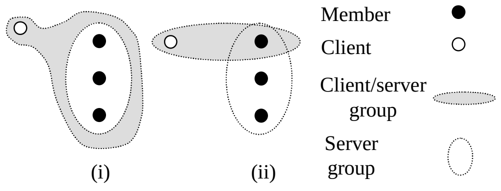


Figure 3: Achieving closed and open groups

Note that in the open group approach, a failure of the server within a client/server group (or the server's disconnection from the client due to some communication failure) will cause the binding between the client and the server to be broken, with the client/server group disbanded. A client can then rebind to some other server within the server group, by creating a new client/server group. In our current design, such actions are handled at the application level in some application specific manner. For example, a client application can be provided with a 'smart proxy' for the server that automatically does the rebinding as suggested here. Rebinding can also be performed at the ORB level, as discussed in section 2.2. In contrast, in the closed group approach, server failures can be automatically masked, without any need for rebinding; this is of course the advantage of making a client and all the servers members of a group.

There is no limit to the number of client/server groups a client may form. Nor is there any limit to the number of client/server groups the members of a server group may participate in. A given group can be configured to use either symmetric or asymmetric total ordering protocol. Furthermore, the open and closed group approaches may be used simultaneously by both clients and members of a server group. This permits implementation of group-to-group request-reply invocations (this will be discussed in a subsequent section).

In contrast to request-reply interactions, peer participation interaction is straightforward to implement (no overlapping groups are required); here the invocation layer simply provides 'one way send' invocation facility. In all, the invocation layer supports the following types of invocation primitives:

One way send - A request requires no reply. The issuer of such a request does not wait for replies and may continue processing;

Wait for first - Only wait for a reply from a single member of the server group;

Wait for majority - Wait for replies from a majority of the server group;

Wait for all - Wait for replies from all members of the server group.

Replies generated from a request are sent to the client directly (closed group approach) or indirectly via a member of the server group (open group approach).

The underlying group communication service has been designed to be suitable for a wide variety of group based applications; objects can simultaneously belong to many groups, group size could be large, and objects could be geographically widely separated. The service can provide causality preserving total order delivery to members of a group, ensuring that total order delivery is preserved even for multi-group objects. Both symmetric and asymmetric total order protocols are supported, permitting a member to use say symmetric version in one group and asymmetric version in another group simultaneously [5].

2.2. Related Work

Enriching CORBA with an object group service has been an active area of research [7-14]. Three ways of incorporating object groups in CORBA have been identified [7,8]. The integration approach takes an existing group communication system and replaces the transport service of the ORB by the group service [9]. Although this is a very efficient way of incorporating group functionality in an ORB, this approach is not CORBA compliant, lacking in interoperability.

The second approach called the interceptor approach also makes use of an existing group communication system; here messages issued by an ORB are intercepted and mapped on to calls of the group communication system. Well known examples of this approach are the Eternal [10,11] and AQuA [12] systems; Eternal uses the Totem group communication system [6], whereas AQuA uses the Ensemble group communication system [3]. Both Eternal and AQuA make use of group communication for supporting object replication only (and not for other uses of group communication, such as collaborative applications). They do so by using the closed group approach, and have been engineered for use in high speed LAN environments, rather than over the Internet. Ignoring for the moment that our system does not use the interception approach (it uses the service approach discussed below), the architecture of our system is different: rather than providing an integrated set of mechanisms for implementing a specific system function suitable in a specific setting (e.g., replication within LAN environments), we have enriched the group communication service with a set of high level invocation and group management primitives that can be used for supporting a wide variety of group based applications, with scope for optimisation based on knowledge of application behaviour and network latency. Naturally, our object group service will need to be used in conjunction with additional subsystems that provide specific functions; for example, in order to support passive replication, some form of state transfer facility would have to be implemented. We have shown elsewhere how a subsystem for replication of transactional objects (that itself uses the CORBA transaction service) can make use of the object group service [16].

The third approach is the service approach: it does not make use of any existing group communication system; rather the group communication system is implemented as a CORBA service from scratch. In addition to being CORBA compliant, the advantage here is that the service is directly available to application builders so can be used for a variety of purposes. This approach was first developed in the Object Group Service (OGS) [7,8], and has been taken in the NewTop service. The NewTop service offers a more comprehensive set of group management facilities than OGS. In particular, OGS does not support overlapping groups or group to group invocations.

NewTop can be adapted to exploit forthcoming enhancements to ORBs. As part of the ongoing development of CORBA, the OMG have recently adopted interceptors, messaging, and fault-tolerance specifications. Availability of ORBs with interceptors will enable the use of NewTop as a multicast transport service as demonstrated by the Eternal system. Exploitation of the messaging service will enable more efficient implementation of multicasting than is possible now. Since at present ORBs only provide one to one communication, multicasting has been implemented by making synchronous invocations in turn to all the members. Multiple threads of execution are used to obtain parallelism and prevent client blocking. Such a measure to prevent blocking will not be required if the ORB supported asynchronous invocation provided by the messaging service.

The forthcoming fault tolerance standard extends the Interoperable Object Reference (IOR) to handle object groups (IOGR - Interoperable Object Group Reference). This is achieved by embedding the IORs of group members within a single IOGR. NewTop can exploit this feature in a number of ways: in open groups, if the client ORB is unable to invoke one of the members of the object group (one IOR is identified as primary and will be chosen first by the ORB), an attempt may be made to invoke another member that is present in the IOGR. As this is executed at the ORB level (possibly with the aid of interceptors), the process is transparent to the client. In a closed group, a multicast may be initiated by the client ORB, sending an invocation to all members present in the IOGR.

Although not a CORBA service, the system described in [17] is worth mentioning. The paper describes a client access protocol for invoking object replicas, without the need for the client to use multicasts. We obtain the same functionality by making use of open groups.

3. Overview of the NewTop Object Group Service

The failure assumptions made by the NewTop service are the same as made in other group services referred to in this paper. It is assumed that processes/objects fail only by crashing, i.e., by stopping to function. The communication environment is modelled as asynchronous, where message transmission times cannot be accurately estimated, and the underlying network may well get partitioned, preventing functioning members from communicating with each other. The actual protocols used in the NewTop service will not be described here, as these details are not directly relevant to this paper; the interested reader is referred to [5].

The NewTop service is a distributed service and achieves distribution with the aid of the NewTop service object (NSO). In the following description, a group member will also be referred to as a client of the NewTop service. Each client is allocated an NSO. Group related communication required by a client is handled by its NSO. Referring to fig. 2 of section 2, the shaded box is an NSO. Only one NSO is required by a client, irrespective of how many groups the client participates in. Communication between a client and its NSO is handled by the ORB. Therefore, the NSO may reside within the same address space, in a different address space, or on a different node in the network to the group member associated with it. The most efficient configuration would be the client and its NSO within the same address space.

Internally, the NewTop service itself has been composed of a group communication subsystem that handles membership and reliable multicasts and the invocation subsystem. The group communication system provides clients with create, delete and leave group operations and causal and total order multicasts. In addition, it maintains the membership information (group view) and ensures that this information is kept mutually consistent at each member. This is achieved with the help of a failure suspector that initiates membership agreement as soon as a member is suspected to have failed. The client can obtain the current membership information by invoking 'groupDetails' operation. View updates are atomic with respect to message deliveries, as in virtually synchronous communication [2]. Message delivery is atomic with two types of ordering guarantees (causal and causality preserving total order) and in case of total order, two types of ordering techniques, symmetric and asymmetric, are supported. In the asymmetric version, one of the members of the group assumes the responsibility for the ordering of messages within the group. Such a member is commonly termed a sequencer. Electing a new sequencer, in case the original one departs from the group, is straightforward as the underlying membership service maintains consistent group views; so any deterministic algorithm can be used. In the symmetric version, all the members use a deterministic algorithm for message ordering.

In a group communication system a member is often required to stay lively within a group to avoid being suspected by other members. This usually takes the form of a member periodically sending "I am alive" or "NULL" messages during periods it has no application level messages to send. In NewTop, after a member has neglected to send a message for a period of time, the NewTop time-silence mechanism will send a "I am alive" message. A client of the NewTop service creating a group may decide if the group is to be lively or event driven:

·*Lively* – time-silence mechanism and failure suspicion is active throughout the lifetime of a group; the duration of the time-silence period is specified at the creation time. Such a configuration would be most appropriate in peer group settings.

·*Event* – The time-silence mechanism is only active when application dependent messages exist within the NewTop service environment. Once all these messages are delivered to group members the failure suspicion and time-silence mechanisms are shutdown. The appearance of further application dependent messages wakes up these mechanisms. Such a configuration would be most appropriate in request-reply group settings.

4. Flexible Object Group Invocation

In this section we describe how the invocation layer implements one way send, wait for first, wait for majority, and wait for all for a client invoking a group of servers. Replies generated from servers are sent to a client directly (closed group approach) or indirectly via a member of a server group (open group approach). Implementation using the closed group approach (fig. 3(i)) is relatively straightforward and will not be described here. Instead we will concentrate on the implementation using the open group approach; for a more detailed description, see [18].

4.1. Open group approach

The client forms a client/server group containing itself and only one member of the server group. As client requests are directed at only a single server, a mechanism that will propagate such messages throughout the server group and collect replies ready for returning to the client is necessary. This mechanism is described, with reference to fig. 4, where all the requests/replies are causality preserving total order multicasts.

(i) *Receiving client request* - A request sent within a client/server group is received by the server. This server is considered to be the request manager for this particular client (fig. 4(i)).

(ii) *Distributing client request* - The request manager multicasts the request within the server group (fig. 4(ii)). This is achieved by the request manager acting as a client and issuing the incoming invocation as a new invocation (of the same type, e.g., wait for first, wait for all).

(iii) *Receiving server replies* - Each member of the server group multicasts replies within the group (fig. 4(iii)). This would be the case when each member is generating replies, as in active replication. A variation on this behaviour is when only one member generates the reply; this will be discussed in the next subsection.

(iv) *Returning server replies to client* - Server replies are gathered by the request manager (one, majority or all) and returned to the client (fig. 4(iv)). No reply is sent when the client invocation is of type one way.

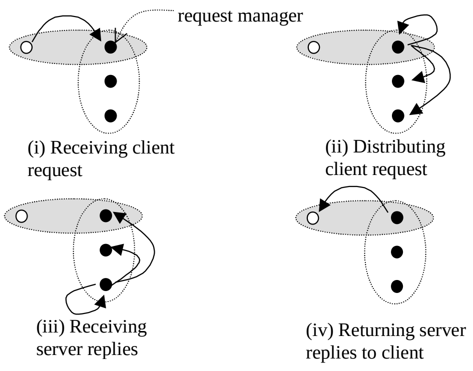

request manager
(i) Receiving client request
(ii) Distributing client request
(iii) Receiving server replies
(iv) Returning server replies to client

Figure 4: Client invocations in open groups

As observed earlier, a failure of the request manager will cause the binding between the client and the server to be broken, with the client/server group disbanded. A client can rebind to some other server within the server group, by creating a new client/server group. Consider this scenario further. Assume that the request manager fails as the servers are multicasting their replies (during the stage depicted in fig. 4(iii)). The server group will be reformed with the request manager removed, and no reply will be sent to the client. Client retries can be handled by the new request manager without causing re-execution, provided retries contain the same call number as the original call and servers retain the data of the last reply message (enabling the request manager to resend the reply). These are 'standard' techniques used in any RPC implementation. Logic for this can be provided either in the invocation layer or at the application level (in client and server stubs): in the current design, we have chosen the latter option. Note that client retries can be handled transparently using the IOGR feature of ORBs as discussed in section 2.2.

4.2. Optimisations

In the above scheme, clients can select any member of the server group for forming a client/server group (fig. 5(i)); total ordering of forwarded requests ensures that all the servers are delivered requests in identical causality preserving order. However, a request received by a request manager becomes deliverable only after it has been delivered through a multicast; this delay can be eliminated at the request manager if only a single request manager is used by all the clients (fig. 5(ii)). This optimisation will be termed restricted group optimisation.

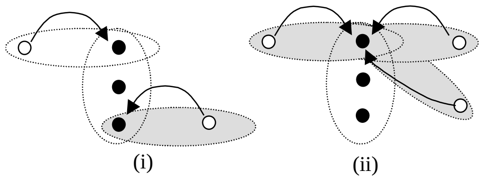


Figure 5: Single request manager optimisation

Further optimisation to the restricted group is possible, termed asynchronous message forwarding, when the client is expecting a reply from a single server (wait for first). The request manager, rather than making ‘wait for first’ calls on the servers (step (ii) of the previous section) makes ‘one way send’ invocation and simply returns a single reply itself. Combining the restricted open group and asynchronous message forwarding approaches as discussed here is particularly attractive for supporting passive replication. The request manager may assume the role of the primary; receiving, processing and replying to client requests. The remainder of the server group are passive members, receiving (but not necessarily acting upon) client requests. The asymmetric ordering protocol would be most suitable in this setting, with the role of the sequencer, request manager and primary all undertaken by the same group member.

4.3. Group to Group Invocations

The open group approach presented earlier provides an economic way of extension to group to group invocations. One scheme is illustrated in fig. 6; here a client invokes group gx; members of gx make another call to gy. Each member of the client group (gx in this case) uses the open group approach to invoke the server group (gy), using the same request manager. Another group, termed a client monitor group, containing gx and the request manager is created. The request manager expects the call request to come from all the members of gz (except itself), and filters them out, except one, and forwards it to members of gy. Each member of gy multicasts its reply within gy. The request manager returns the replies to all the members of gx by multicasting within gz.

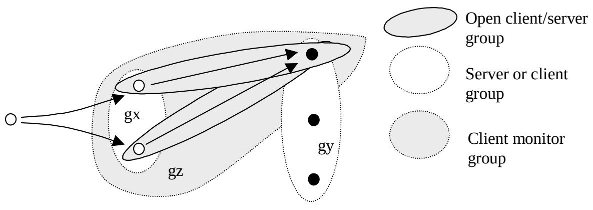


Figure 6: Implementing group to group invocations

The scheme described above is one of several ways of implementing group to group invocations. Another way would be to use request managers at each group to perform message distribution. In the design presented here, the aim has been to minimise inter-group (between gx and gy) multicasts. The only such multicast is from the request manger to members of gz; this is necessary to ensure atomic delivery to all the functioning members of gx.

4.4. On the use of Overlapping Groups

Isis system was first to use overlapping groups [2]; it supported closed groups for client-server interactions. We have taken this approach a step further and described the use of overlapping groups for supporting closed groups, open groups and for supporting group to group invocations. The AQuA system [12] also uses overlapping groups in a variety of ways for replica management. The group communication protocols used in NewTop have been designed to cope with overlapping groups in an efficient manner [5]. In the open group approach, we have relied on the use of a client/server group for invoking a single server. Since no multicasting is involved, a client can in principle invoke the server directly, without using the group system. Although this is possible (and used in [17]), we have chosen the former approach because it has the power of preserving causality (if any) between multiple client requests. This is illustrated with the help of fig. 7.

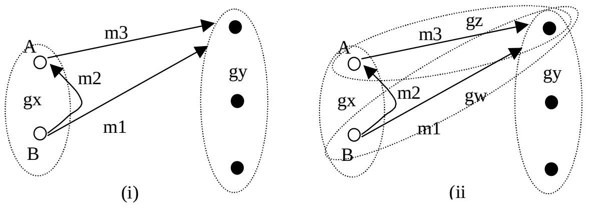


Figure 7: Ordering of related client requests

A group gx consists of two members (A and B). B issues an open group request to gy (m1). B then sends a message in gx (m2). During the processing of m2, A issues an open group request to gy (m3). We want to ensure that request m1 is serviced at gy before m3. This will be the case if requests m1 and m3 are sent using client/server groups (fig. 7(ii)).

5. Performance Evaluation

This section describes the experiments carried out to assess the benefits of customisation facilities made available to application developers. That is, we assess the advantages of being able to select between closed and open groups as well as two ways of enforcing total order (asymmetric or symmetric) for a given group. Two classes of experiments were carried out:

Request-Reply - A client issues a request to multiple servers and waits for their replies, fig. 1(i). Inter-server as well as client-server communication could be via a high speed LAN (local distribution) or Internet (wide area distribution). Performance of closed and open group approaches (figs. 8(i) and (ii) respectively), and the restricted group with asynchronous message forwarding optimisation (discussed in section 4.2) depicted here in fig. 8(iii)) have been evaluated under the two classes of ordering protocols.

Peer Participation - All the members are regularly multicasting by using the asynchronous method invocation operation, fig. 1(ii). Performance of locally distributed and widely distributed groups under the two classes of ordering protocols have been evaluated.

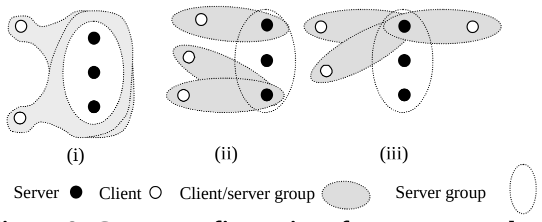


Figure 8: Group configurations for request-reply interactions

The two network environments used in the experiments were: (i) LAN: Pentium Linux machines connected by 100 Mbits fast Ethernet; and (ii) WAN: Pentium Linux machines were geographically separated by large distances, communicating via the Internet; the machines were located in Newcastle (United Kingdom), London (United Kingdom) and Pisa (Italy). The ORB used in these experiments was omniORB2 [19]. In all the experiments, group members reside in the same address space as their NSOs.

5.1. Request Reply Interactions

Two performance metrics are of interests: for a client we wish to know the RPC time for invoking a service, whereas for a server we wish to know the throughput (number of requests serviced per second). To measure server throughput, clients were configured to issue requests as frequently as possible: as soon as a reply is received, another request is issued. The server used in this experiment is a CORBA object that simply returns a pseudo random number when requested to do so by a client. Client numbers were increased gradually from one to twenty. At each of these increments each participating client is timed for 1000 requests, and the average is taken.

Given the above scenario and assuming negligible computation time for a service (as is the case here), we would expect servers to become saturated (reach maximum throughput) with only small number of clients if the clients are connected by a low latency path to the servers; at the same time, RPC times would increase as the client numbers increased. On the other hand, if the clients are connected by a high latency path to the servers, then we would expect the server throughput to increase as the number of clients increased, and the RPC times would not be affected that much.

The server group consisted of three members in all these experiments with all groups designated as event driven; the following client/server group configurations were used:

(i) *Low latency*: clients and servers were all on the same LAN;

(ii) *Low and high latency*: servers were located on the same LAN in Newcastle; clients were equally distributed between London and Pisa; and,

(iii) *High latency*: servers and clients were geographically separated between Newcastle, London and Pisa.

5.1.1 Non-replicated service

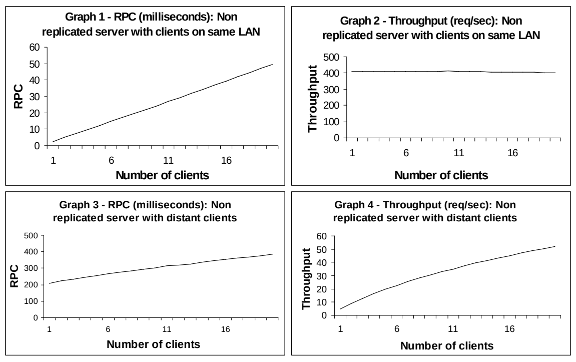
To enable comparative analysis of the performance figures, CORBA RPC times without the use of the NewTop Object Group Service were first obtained. The figures obtained are shown in table 1.

The experiment was repeated, however, communication was achieved via the group service. Performance figures within the LAN environment are shown in graphs 1 and 2 and those over the Internet in graphs 3 and 4.

CORBA RPC	***Timed request (milliseconds)***	***Throughput (requests per second)***
Client and server on distinct nodes in LAN	**0.9**	**1111.11**
Client in Pisa and server in Newcastle	**78.0**	**12.82**
Client in London and server in Newcastle	**81.0**	**12.34**
Client in Pisa and server in London	**86.0**	**11.62**

Table 1: Performance of CORBA RPC

The first observation to be made is that the RPC time of a single client making a call via the NewTop service (2.5 msec, LAN and 209 msec, Internet) is around two and half times the performance of a single client making an RPC without the NewTop service. This drop in performance is inevitable and may be explained by the manner with which messages are handled; this message passing process is shown in fig. 9.



A request by a client for a pseudo random number is received by the client's NSO (m1). The client's NSO then multicasts this message to the replica group (m2). The server's NSO receives this message and queues this message as pending. Messages in the pending queue that satisfy ordering and delivery guarantees are then delivered to the NSO's associated application object (pseudo random number generator object) (m3). This delivery takes the form of an invocation. Results from the invocation are then queued by the server's NSO in the multicast pending list (m4). A thread is then created which handles the multicasting of messages held in this list. Finally, the client's NSO receives this return message (m5) and queues it in the messages pending list, awaiting delivery back to the client object of the NSO (m6). Assuming the client (server) NSO to be in the same address space as the client (server), request-reply message pairs m1-m6, m3-m4 will not generate any network traffic. On the other hand, message m2 as well as message m5 are each a CORBA RPC. As we have remarked earlier (section 2.2) expected availability of asynchronous messaging (and multicast services) in next generation ORBs will remove the main source of the inefficiency.

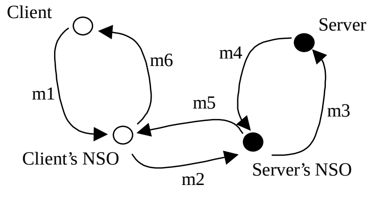


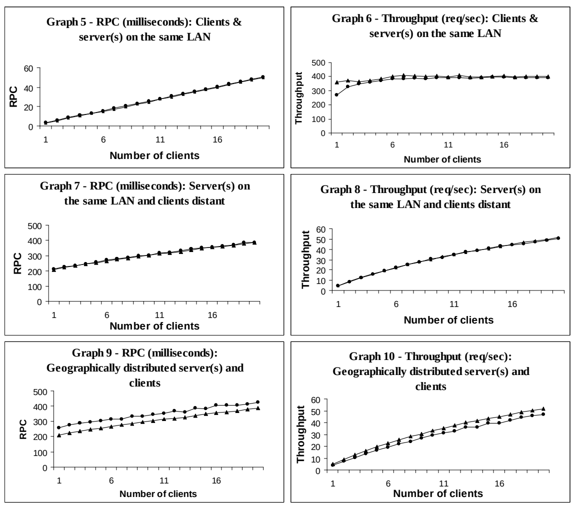
Figure 9: Message passing between NSOs and application objects

As the number of clients is increased, the behaviour obtained is consistent with the explanation of expected behaviour given earlier; so in the LAN environment, a single client is able to saturate the server, whereas the throughput of the server in the Internet environment increases with the number of clients.

5.1.2. Optimised group invocation

We first consider the group invocation requiring a reply from a single server (wait for first) and present the performance of the scheme depicted in fig. 8(iii) incorporating the restricted open group with asynchronous message forwarding optimisation that is ideal for passive replication. This will enable us to directly compare performance of group invocation with that of non-replicated invocation discussed in the previous subsection: in both the cases, the client invokes a single server; the only additional work required for group invocation is that the request manager is required to forward the request to all the members. Since this is performed asynchronously, we would expect the performance of optimised group invocation to closely match that of the non-replicated invocation. This is indeed the case when the asymmetric ordering protocol is used, with the role of the sequencer, request manager and primary all undertaken by the same group member.

Legend for graphs - Optimised open asynchronous ● Non-replicated server ▲



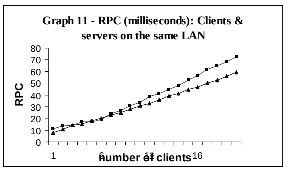
The performance figures (including those of non-replicated server obtained earlier) are presented in graphs 5 through to 10 for the optimised asynchronous open group combination.

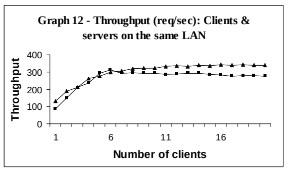
5.1.3. Closed and open group invocations

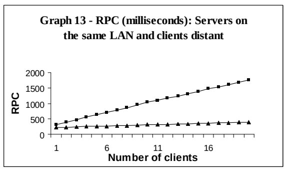
We next compare the performance of request reply interactions using closed groups and open groups. In the closed group approach, a server failure is automatically masked, whereas in the open group approach, clients bound to that server are required to rebind. In the absence of failures, the open group approach is expected to perform better than the closed group approach, although within low latency networks, the difference between the two is not expected to be significant. The performance figures, for the case of the server group using asymmetric ordering protocol and clients invoking ‘wait for all’ are shown in graphs 11 through to 16. We note that when clients are separated from servers by high latency paths, the open group approach is most attractive.

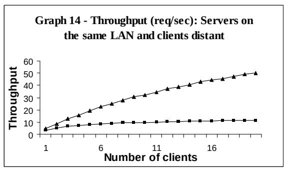
Figures obtained for the case of the groups using the symmetric ordering protocols have been omitted here to save space, but we can make two observations: (i) the closed group approach does not perform well, because it gives rise to extensive protocol related multicast traffic amongst all the members for ensuring order; and (ii) under the open group approach, there is little to chose between the two; this is because message ordering is performed within one group only, and it does not matter which technique is used.

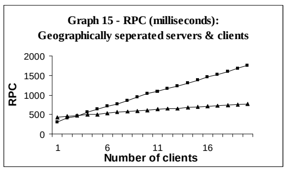
Legend for graphs - Closed ■ Open ▲

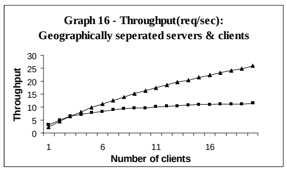












5.2. Peer to Peer Interactions

The peer participation scenario is commonly associated with applications wishing to share information across a number of participants. GroupWare applications are typical examples of such applications (e.g., teleconferencing, shared whiteboards, Internet Relay Chat (IRC)). Members simply distribute messages (via multicast) throughout the group using one way send; the body of the message consists of a CORBA string type of 100 characters in length. All members issue multicasts as frequently as possible. Performance is measured by assessing how long a multicast takes to become deliverable at all members within a group from the time of the multicast’s issue. The time taken for 1000 multicasts from each member to become deliverable at every other member of the group is measured. This results in a figure per client (as all clients are multicasting). Each of these figures are then divided by a 1000 to gain a figure for the time taken for a single multicast. The resultant figures are then summed to allow a throughput figure for the group to be gained.

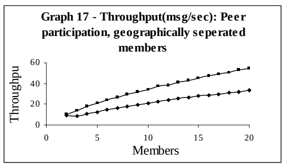
The group was designated as lively; two group configurations were used:

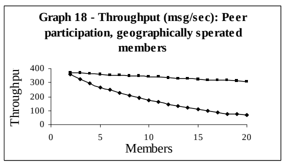
(i) *Low latency*: members were distributed over the same LAN; and,

(ii) *High latency*: members were distributed between Newcastle, London and Pisa.

The symmetric ordering scheme is superior to the asymmetric ordering scheme in peer to peer interactions. In the LAN environment the volume of messages that result from persistently sending asynchronous multicasts has resulted in a deterioration of performance in both the symmetric and asymmetric protocols as group membership rises. This deterioration is more extreme in the asymmetric protocol than the symmetric protocol. This indicates that the sequencer is receiving more messages than it can handle. The sequencer is a bottleneck. This bottleneck effect explains why the asymmetric performance deteriorates significantly as group membership rises. This deterioration is not evident in the symmetric scenario as the handling of messages and multicasting is more evenly spread throughout the group. In the Internet scenario, due to the large message transit times involved, the sequencer does not present a bottleneck. However, the cost of redirection is evident; the performance of the asymmetric protocol is approximately half that of the symmetric protocol.

Legend for graphs - Symmetric ■ Asymmetric ♦





6. Concluding Remarks

We have taken a modular approach in the design of the NewTop object group service described here. First we have implemented a CORBA group communication service that supports overlapping groups (objects can simultaneously belong to many groups) and symmetric and asymmetric total order protocols. Then we have enriched the group communication service with a set of high level invocation and group management primitives that can be used for supporting a wide variety of group based applications, with scope for optimisation based on knowledge of application behaviour and network latency. For the case of request-reply interactions, we have implemented open and closed group approaches.

The closed group approach works well within low latency networks but does not scale to high latency networks (wide area distribution), where the open group approach is most suitable. The optimised version of the open group approach works well in all the settings and almost matches the performance of its non-replicated counterpart. Finally, experiments indicate that the asymmetric ordering protocol is more appropriate for groups that are used in request/reply mode, whereas the symmetric protocol is more suitable in peer to peer interactions.

Acknowledgements

G. Morgan was supported by EPSRC CASE PhD studentship with industrial sponsorship from HP Laboratories, Bristol.

References

[1] Amir, Y., et al, "Transis: A Communication Sub-system for High Availability", Digest of Papers, FTCS-22, Boston, July 1992, pp. 76-84.

[2] K. Birman , "The process group approach to reliable computing", CACM , 36, 12, pp. 37-53, December 1993.

[3] M. Hayden, "The Ensemble system", PhD thesis, Dept. of Computer Science, Cornell University, 1998.

[4] D. Dolev and D. Malki, "The Transis approach to high availability cluster communication", CACM, 39 (4), April 1996, pp. 64-70.

[5] P. Ezhilchelvan, R. Macedo and S. K. Shrivastava, "NewTop: a fault-tolerant group communication protocol", 15th IEEE Intl. Conf. on Distributed Computing Systems, Vancouver, May 1995, pp. 296-306.

[6] L.E. Moser, P.M. Melliar-Smith et al, "Totem: a Fault-tolerant multicast group communication system", CACM, 39 (4), April 1996, pp. 54-63.

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

[8] P. Felber, "The CORBA Object Group Service: a Service Approach to Object Groups in CORBA", PhD thesis, Ecole Polytechnique Federale de Lausanne, 1998.

[9] S. Maffeis, "Run-time support for object-oriented distributed programming", PhD thesis, University of Zurich, February 1995.

[10] P. Narasimhan, L, E. Moser and P. M. Melliar-Smith, "Replica consistency of CORBA objects in partitionable distributed systems", Distributed Systems Eng., 4, 1997, pp. 139-150.

[11] L.E. Moser, P.M. Melliar-Smith and P. Narasimhan, "A Fault tolerance framework for COBRA", Proc. of 29th Symp. On Fault Tolerant Computing, FTCS-29, Madison, June 1999.

[12] M. Cukier et al., "AQuA: an adaptive architecture that provides dependable distributed objects", Proc. of 17th IEEE Symp. on Reliable Distributred Computing (SRDS'98), West Lafayette, October 1998, pp. 245-253.

[13] G. Morgan, S.K. Shrivastava, P.D. Ezhilchelvan and M.C. Little, "Design and Implementation of a CORBA Fault-tolerant Object Group Service", Distributed Applications and Interoperable Systems, Ed. Lea Kutvonen, Hartmut Konig, Martti Tienari, Kluwer Academic Publishers, 1999, ISBN 0-7923-8527-6, pp. 361-374.

[14] S. Misra, Lan Fei, and Guming Xing, "Design, Implementation and Performance Evaluation of a CORBA Group Communication Service", Proc. of 29th Symp. On Fault Tolerant Computing, FTCS-29, Madison, June 1999.

[15] L. Rodriguez, H. Fonseca and P. Verissimo, "Totally ordered multicasts in large scale systems", 16th IEEE Intl. Conf. on Distributed Computing Systems, Hong Kong, May 1996, pp. 503-510.

[16] M.C. Little and S K Shrivastava, "Implementing high availability CORBA applications with Java", Proc. of IEEE Workshop on Internet Applications, WIAPP'99, San Jose, July 1999.

[17] C. T. Karamanolis and J.N. Magee, "Client access protocols for replicated services", IEEE Transactions on Software Engineering, Vol. 25, No. 1, 1999, pp. 3-22.

[18] G. Morgan, "A middleware service for fault tolerant group communications", Phd thesis, Dept. of Computing Science, University of Newcastle upon Tyne, September 1999.

[19] www.uk.research.att.com/omniORB/omniOB.html