통신시스템을 위한 공유메모리 기반 ORB 연동 프로토콜의 설계

Design of Shared Memory-based Inter-ORB Protocol for Communication Systems

장익현*, 조영석**
동국대학교 정보통신공학과*, 동국대학교 컴퓨터 · 멀티미디어학과**

Ikhyeon Jang(ihjang@dongguk.ac.kr)*, Youngsuk Cho(jys@dongguk.ac.kr)**

요약

통신시스템 소프트웨어는 대단히 크고 복잡하기 때문에, 소프트웨어 재사용성, 하드에어 투명성, 응용에 따른 소프트웨어 재구축의 용이성과 고성능을 위한 컴포넌트 기반 구조를 요구하고 있다 이런 요구사항을 만족시키기 위하여, 기존 CORBA IIOP의 성능과 통신방식에 대한 분석을 통해 통신시스템에 적합한 공유메모리 기반의 CORBA 연동 프로토콜을 설계하였다. 설계된 프로토콜은 동일한 인터페이스를 지원하며 동일 시스템 환경에서의 메시지 전송 오버헤드를 최소화시킨다. 다른 프로토콜과의 비교시험 결과 새로운 프로토콜은 약15% - 200%의 성능향상을 보여주고 있다. 따라서 본 논문에 제시된 프로토콜은 통신시스템을 위한 CORBA 기반의 컴포넌트 소프트웨어 개발을 위해 사용될 수 있을 것이다.

■ 중심어 :□코바□인터넷연동프로토콜□공유메모리□통신시스템□

Abstract

Since communication systems software is very large and complex, it requires component based architecture for software reusability, hardware transparency, high performance, and easy software reconstruction in different applications. In order to meet these requirements, we analyze performance and inter-process communication techniques of existing CORBA IIOP, and designed a shared memory-based CORBA inter-ORB protocol that would best fit for communication systems software. The designed protocol supports the same interface and can minimize the message transfer overhead in the same host environment. The test results of our protocol compared with other protocols show that the performance is increased by about 15% - 200%. We are thus assumed that our protocol can be used in developing CORBA-based component software architecture for communication systems.

■ keyword :□CORBA□IIOP□Shared Memory□Communication Systems□

I. Introduction

Due to rapidly developing internet services and growing number of service users, today's network structure must be able to respond immediately to new service demands. Consequently, network system architecture is becoming middleware based open architecture that can provide new services through inter-networking among heterogeneous equipments by using standard interfaces.

The OMG (Object Management Group) defined the standard interface specifications of CORBA which

접수번호 : #081204-003
접수일자 : 2008년 12월 04일
심사완료일 : 2008년 12월 20일
교신저자 : 조영석, e-mail : jys@dongguk.ac.kr

allow clients to provide software services for distributed components regardless of component locations, programming languages, OS platforms, communication protocols, and hardwares[1]. Established in 1991, OMG now has more than 500 members, and CORBA extended its application scope from ordinary applications to real time systems and built-in systems[2].

As the productivity and reusability of software have become more important, CORBA-based software supporting distributed process between OS and applications software has been researched and developed in the field of communication systems software, too.

In the area of communication systems software, there are two reasons for increased requirements for CORBA. First, there is a requirement for the aspects of internal structure of communication systems. The communication systems software is not only very large and complex but also depends heavily on the hardware of target system. In order to enhance its productivity and quality, component architecture is required for software reuse and the transparency of hardware architecture is needed for an easy reconstruction of software for different applications. Although CORBA can be used as a software bus to meet these requirements, it is unpractical to use conventional CORBA as a software bus for communication system because of its low performance.

Second, there is a requirement for open architecture of communication systems. As communication systems software architecture has been evolving into open architecture based on standard interfaces, reducing the time required for software development has become a key factor to system competitiveness.

The signal based communication systems software has an ineradicable structural weakness by depending on OS, programming language and message transfer protocol for its development because it approaches distributed objects through inter-process communication supported by the operating system. In contrast, CORBA-based development methodology is independent of OS and programming languages. Therefore, CORBA-based development methodology ensures flexibility and scalability in software development. Moreover, when the CORBA-based software development methodology is applied to the development of software for communication systems, it is possible to reduce the time for software development, to flexibly configure systems, and as a result, to improve efficiency in the system maintenance. Also, CORBA can be adapted to a new trend of an open architecture with a paradigm shift of the methodology from the signal-based communication systems software development to the CORBA-based software development.

In this paper, we focus on the first reason that is a requirement for the aspects of the internal structure of communication systems. In order to design a high speed CORBA, we propose a shared memory-based inter-ORB protocol for distributed processing between software blocks running on the same host, which will ensure the development of a high performance CORBA platform for communication systems.

This paper is organized as follows. Chapter 2 analyzes the specifications and internal structure of CORBA. Chapter 3 describes the design of inter-ORB protocol based on shared memory. Chapter 4 analyzes the performance of the proposed model. Finally, we discuss conclusive remarks in chapter 5.

II. CORBA Specification and Internal Overview

CORBA specifications define interoperability protocol to support inter-networking among object request brokers[1,2]. This makes it possible to provide transparent services by defining message formats and communication protocols common to object service exchanges among object request brokers. This paper will analyze related studies on improving the functions and performances of CORBA defined by the OMG. This paper will also carry out performance analysis related to CORBA components, and present the methods to improve CORBA performances.

1. CORBA Specifications

Generally, CORBA specifications define interfaces only. Provided features of CORBA include: object location transparency, management of connection and memory, parameter (de)marshaling, (de)multiplexing of events and requests, error handling, failure endurance, activation of object/server, parallelism, and security functions. [Fig. 1] illustrates the key components in the CORBA reference model that collaborate to provide applications transparency from language, OS, hardware and network protocol[1][9].

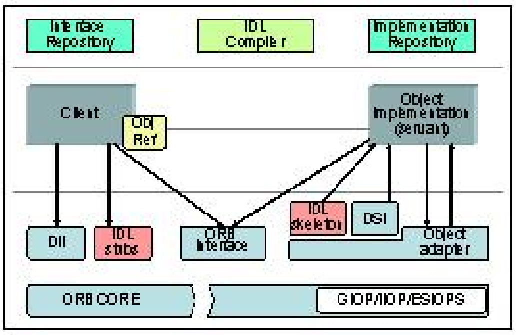

Fig. 1. Key component of CORBA reference model

CORBA specifications define the following four interfaces for systems development[1][2].

1) IDL (Interface Definition Language)

Interfaces for implementation objects show different mappings depending on the programming language. IDL compiler is independent of programming languages so that it can express interfaces for objects. Generated codes must provide the means for providing transparent services for implementation objects.

2) ORB (Object Request Broker)

Client programs using CORBA can transparently request server side implementation codes located on the same or remote system through ORBs. ORBs follow the procedure by intercepting service requests from clients, find portable object adaptors ready to provide object services, call relevant methods including parameters, and receive return values if necessary. In this case, clients can request services without the information on the locations of the service implementation objects, operating system, hardware and the programming language. In this way, ORBs provide location transparency with which services can be provided for objects implemented in heterogeneous systems; independence from programming languages and interoperability by which multi-implemented systems can be supported.

3) POA (Portable Object Adaptor)

POAs bridge object request brokers and implementation objects. They activate implementation objects and control services while maintaining the information on service requests and object status.

4) IOP (Interoperability ORB Protocol)

Generally, inter-ORB communications use GIOP (General Inter-ORB Protocol) and IIOP (Internet Inter-ORB Protocol). Both of them define general inter-networking specifications. The OMG does not define any specific protocol as the transport layer protocol. GIOP defines communication specifications by which service request side can marshal/demarshal its hardware architecture information and parameters.

2. IIOP Specifications

IIOP uses CDR (Common Data Representation) grammar to marshal the messages written by CORBA IDL. Then IIOP transfers the standard GIOP formatted messages to the destination CORBA through TCP/IP. IIOP operations are defined in the OMG Specifications 2.0, and they have following basic functions[1][2].

1) Object location service

IIOP defines location service for objects in the client/server model. When a client transmits LocateRequest message for access to objects, the server responds with LocateReply message containing information on the relevant objects and lets the client know their location.

2) Connection management

The client configures and manages its connection according to the server address information extracted from the inter-operable object reference created by the server. As single-thread-based CORBA applies connection oriented transfer protocol to service requests and response messages, the sequence of response messages is guaranteed. In contrast, multi-thread-based CORBA shows asynchronous operations without guaranteeing the sequence of response messages. Disconnecting operation may be activated by both the client and the server. After receiving disconnection message, IIOP objects will activate disconnection and cancel allocation of allocate resources on the basis of connection identifiers.

3) Message format

Seven types of messages are defined as shown in [Table 1]. Every message includes 12-byte necessary message header information. Additional header information may be added depending on the types of messages.

Table 1. IIOP message types

Types	Direction	Value	Description
Request	Client⇒Server	0	Service request
Reply	Client□Server	1	Service response
CancelRequest	Client⇒Server	2	Service cancellation request
LocateRequest	Client⇒Server	3	Object location request
LocateReply	Client□Server	4	Object location response
CloseConnection	Client□Server	5	Connection closing
MessageError	Client⇔Server	6	Message error

4) Message transfer

Generally, IIOP supports TCP/IP-based transfer protocols. These protocols are often developed by applying high-speed transfer protocols like ATM depending on message characteristics, data sizes or application fields.

3. Analysis of Methods for Improving CORBA Performance

IIOP based generic CORBA platform is applicable only to the network management like TMN since it does not meet the high performance requirements of communication systems. Consequently, various studies have been conducted to improve CORBA performances. The studies focus on the three following themes[3][4].

1) Optimization of (de)marshaling techniques for a data type conversion

A frequent memory copy during a data type conversion degrades CORBA performances. I/O vectors can be used to optimize CORBA performances. However, this method can not satisfy performance requirements for communication systems.

2) Data compression

It is possible to omit version information or 4 byte magic information from GIOP header. However, this violates CORBA specifications. Compressing data without the header leads to a large fluctuation in performance depending on data sizes and system loads.

3) Optimization of message transfer time

High speed transfer networks like ATM can be used to optimize message transfer time which takes about 70% of the whole processing time. However, a shared memory in the same system, or mechanisms like message queues can be applied.

In the following sections, we will analyze the performances of the CORBA platform depending on its constituents and the transfer methods in order to find best ways to improve its performances.

4. Analysis of CORBA Inter-ORB Protocol Performance

IIOP is used to exchange GIOP messages defined as a set of standard message format combined with common data expression types, in order to ensure the communication among several ORBs on the TCP/IP network. General CORBA communication model uses primarily TCP/IP or UNIX domain socket to ensure communications between two components, as shown in [Fig. 2].

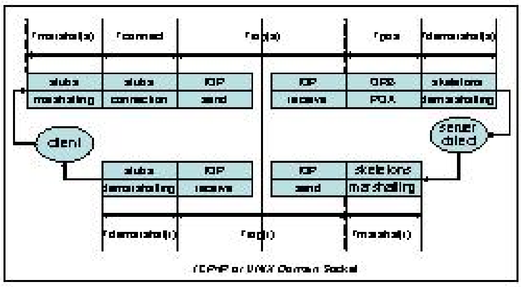

Fig. 2. CORBA communication model

This paper will find optimizing elements by analyzing the performances of the basic CORBA platform. The test environment for this analysis comprises Solaris 2.7, UltraSparc 167MHz and 192MB memory. The objective is to design a high-performance platform for communication systems.

Assuming that client-to-server CORBA service request maintains the packet transfer capability at a constant level, the CORBA performances can be decided by the attributes like the (de)marshaling, the object activation and the message transfer time.

Table 2. Comparison of execution time by CORBA constituent (unit: ms)

	Ttrans	Tpoa	Tmarshal	Ttotal	Ratio
Character	0.7552	0.0067	0.0207	0.7826	94 %
Long	0.3467	0.0078	0.0201	0.3747	93 %
Float	0.3776	0.0066	0.0203	0.4045	88 %
Double	0.5549	0.0067	0.0207	0.5823	91 %
String	0.5071	0.0063	0.0249	0.5384	90 %
Sequence	0.3208	0.0064	0.0264	0.3536	84 %

We will present criteria for improving performances of the conventional models by providing performance analysis for a service which has three parameter types, in, out, and inout, and returns values of the same type.

[Table 2] shows performance analysis results regarding various data types.

$$Tmarshal = \sum(Tmarshal(s) + Tdemarshal(s) + Tmarshal(r) + Tdemarshal(r)) \quad (1)$$

Where, Tmarshal is parameter marshalling time.

$$Ttrans = \sum(Tiiop(s) + Tiiop(r) + Tconnect) \quad (2)$$

Ttrans is the time spent transferring messages marshaled on the client side to the server side through transfer layer and returning the result values.

Therefore, Ttotal (time needed to process one CORBA service) is,

$$Ttotal = \Sigma(Tmarshal + Tpoa + Ttrans) \quad (3)$$

Tpoa(time used by ORB on the server side) is the time required to interpret requested services, to find the relevant objects in the implementation object repository and to operate services.

The ratio of Ttrans needed to transfer messages through inter-ORB protocol showed a slight difference depending on the data types.

$$Ratio = (Ttrans / Ttotal) * 100$$

Though there may be variations depending on the data size of transferred parameters, more than 80% of the whole time was spent transferring messages, as shown in[5].

[Table 3] shows the average time spent transferring data of different sizes one thousand times by using a shared memory, TCP/IP and UNIX domain socket, as ways of communications in the model of client/server in the same system.

Table 3. Comparison of transfer time by communication model (unit: ms)

	1KB	4KB	16KB	32KB	64KB
TCP/IP	0.0632	0.0662	0.1640	0.3054	0.6106
Unix Domain Socket	0.0310	0.0371	0.1329	0.2618	0.4537
Shared Memory	0.0755	0.0790	0.0784	0.0770	0.0768

As shown in [Table 3], the transfer capability of a shared memory model remains almost the same although the increase of data size is steep. In contrast, the transfer time of TCP/IP and UNIX domain socket increases rapidly as the data size increases.

III. Shared Memory-based Inter-ORB Protocol

To support the characteristics of the communication system that the system functions are distributed in distributed hardware, CORBA has to support specialized functions like real time processing, high reliability and high performance. In general, communications among separate function modules are performed through inter-process inter-networking protocols like sockets, or remote procedure calls. In contrast, CORBA applications use socket-based IIOP as a standard.

Based on the performance analysis in chapter 2, we will propose a shared memory-based IIOP as a way of minimizing message transfer time, and apply the proposed protocol to the development of a high-performance CORBA platform applicable to communication systems. We use ORBit CORBA[6] as a basic platform, which is one of GNOME projects performed with the support of Redhat.

1. Architecture of Shared Memory-based IIOP

Shared memory techniques used in UNIX-based operating systems include System V and POSIX model. System V shared memory techniques require kernel rebuilding in order to change the size of the shared memory. In contrast, POSIX does not require kernel rebuilding. Therefore, we applied POSIX model to implement the proposed protocol.

Following requirements must be met to implement a shared memory-based IIOP.

- A server side IIOP must allocate a shared memory for inter-process communication.
- Relative addresses should be used to share a shared memory by clients.
- A multi-process synchronization must be provided.

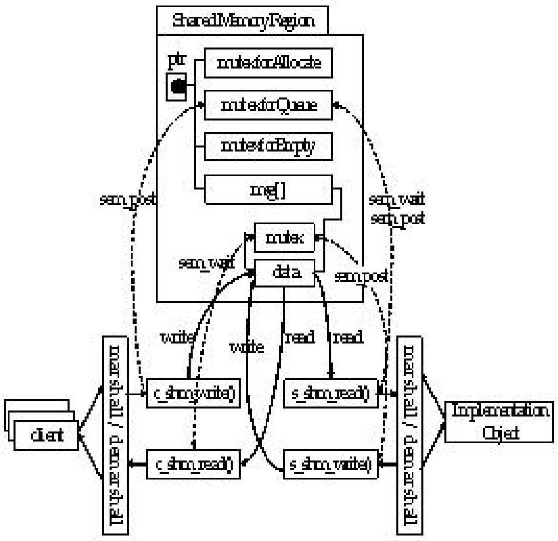

Fig. 3. A Shared memory-based IIOP model

To satisfy these requirements, a server side IIOP should allocate shared memory areas and initialize semaphore variables for the synchronization, as shown in [Fig. 3]. Clients shall insert generated IIOP messages into the message area of a shared memory, change semaphore values, and process them in such a way that they can be processed on the server side. Detailed operation algorithms and status transitions will be described in the next section.

Data structures for the shared memory GIOP message transfer have an entry for GIOP messages, client identifiers, and flags to determine whether the request for the message information and direction service is one way or two ways.

```
typedef struct GIOPMsgSlot {
    sem_t   mutex;              /* signal for client */
    int     clientID;           /* Client Id */
    char    data[MSGSIZ]; /* GIOP msg */
    int  response_expected;  /* 1 way / 2 way */
} MsgSlot;
```

Following is the data structure maintained inside the shared memory generated for communications on the server side.

```
struct ShmIIOPMsg {             /* struct in SHM */
    sem_t  mutexforAllocate;   /* slot allocation */
    sem_t  mutexforQueue;      /* queue control */
    sem_t  mutexforEmpty;      /* queue control */
    int      reason       /* Server failed reason */
    GIOPMsgSlot msg[MAX_CONN];
}
```

ShmIIOPMsg structure is maintained in the shared memory for communications between the server IIOP and the client IIOP. mutexforAllocate is a semaphore used to prevent inter-client conflicts in allocating GIOPMsgSlot. mutexforQueue and mutexforEmpty semaphores are used to control the inserted messages between clients and servers so that other parties can read the messages.

A client inspects the value of reason to determine whether server objects are activated, recovers the allocated resources when the server shows an abnormal stop, and initializes the resources to solve synchronization problems.

Supporting a shared memory-based IIOP model requires an expansion of IOR structures, as shown in [Table 4]. A new profile type for IOP_TAG_UNIORB_SHAREDMEMORY is defined and the path information of the shared memory created by a server is included.

Table 4. IOR structure of shared memory- based IIOP model

Object ID	Profile Count	Profile Type	Shared Memory Name Path	Object Key

2. Analysis of State Transition and Operation Algorithms

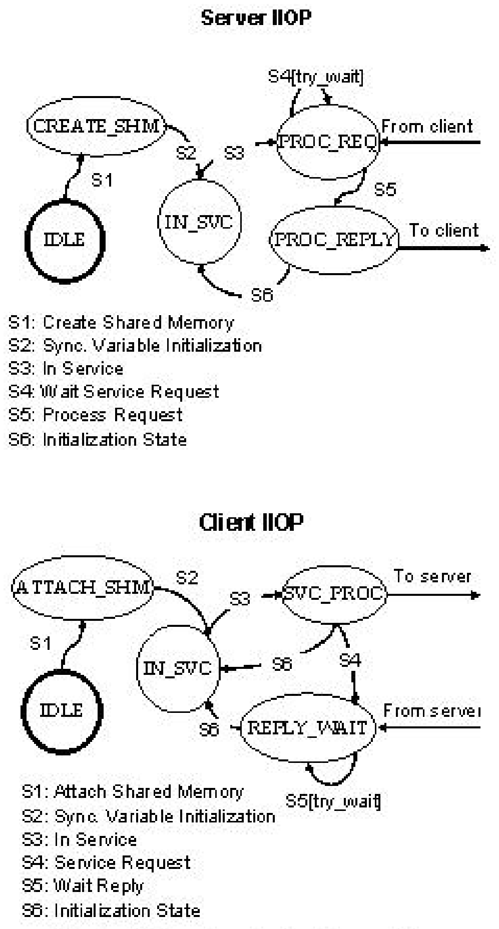

Fig. 4. IIOP protocol state transition

As shown in [Fig. 4], a server side shifts from initial IDLE to CREATE_SHM to create a specific size of the shared memory. If a shared memory already exists, a server side shifts to IN_SVC by deleting the existing shared memory and creating a new shared memory. A server side shifts from IN_SVC, where semaphore variables for a multi-process synchronization of service requests from multi-client processes are initialized, to PROC_REQ, where service requests from clients are waiting. When semaphore values, which confirm messages, are inserted from clients, the server side shifts first to PROC_REPLY go as to process relevant services and send responses, and next to IN_SVC to process new service requests after response messages are transmitted.

A client shifts to ATTACH_SHM to share the shared memory created on the server side from initial IDLE, and changes its state to IN_SVC in order to initialize synchronization variables. If message slots, defined in order to prevent conflicts among multi-clients, are provided to a client, the client shifts to SVC_REQ where messages of service requests can be created and transferred to the server side. If responses from the server are requested, the client shifts to REPLAY_WAIT, waits for responses from the server, processes responses, and changes the state to IN_SVC. However, if the request is a one-way service, the client repeats the transition to IN_SVC where services can be started again.

Following potential problems were found in the implementation of the shared memory-based inter-ORB protocol. First, clients should not be allowed to access the shared memory until the server side inter-ORB protocol creates a shared memory area. Second, new services must not be started until the server side inter-ORB protocol returns responses. Third, if the execution is interrupted while ORB performs the implementation objects, the information on the deactivation of the implementation objects must be transferred to the shared memory to prevent semaphore errors.

To solve these problems, we used a technique with which the server side inter-ORB protocol checks the value of keepalive and initializes semaphore information. The value is provided by a client side when client programs are interrupted by a user or their services are interrupted by program errors while implementation objects are executed.

2.1 Server Side IIOP Algorithm

1) ① A server side inter-ORB protocol allocates shared memory areas, creates data structures to exchange messages with clients, and initializes them.
2) ⑤ Wait for service requests from clients.
3) ⑥ Read GIOP messages in the shared memory buffer.
4) ⑦ Change semaphore variables for synchronization to prevent clients from making new service requests.
5) ⑧⑨ Interpret GIOP messages and process requested services.
6) ⑩-⑫ Create response messages, if necessary, and insert them into shared buffer. Change semaphore values so that clients can read them.
7) Repeat steps ④~⑬.

```
// Server Side IIOP
Declaration of variables
SharedMemoryBuffer {
    Lock[N] MUTEX_LOCK;
    msg[9][28] GIOPMessage;
}
SharedPtr *Ptr;

Initialization
      Ptr = MemoryMap(NULL, fd, flags);
      Semaphore Initialize (Lock[N])
Action
①: Make Server Shared Memory Buffer Create
②: While loop
③: do
④:       Semaphore Trywait(Lock)
⑤:       Wait for Service Request from Client
⑥:       Read GIOPMessage from SharedBuffer
⑦:       Semaphore Post(Lock)
⑧:       Decode GIOPMessage
⑨:       Handle Incoming Message
⑩:       if Reply_required then
⑪:           Send Reply GIOPMessage to Client
⑫:           Semaphore Post(Lock)
⑬:       fi
```

2.2 Client Side IIOP Algorithm

1) ① Client side IIOP attaches shared memory created by the server side IIOP to its process and creates data structures.
2) ② Allocate a channel for client processes to prevent inter-client conflicts.
3) ④ Create GIOP messages for service requests.
4) ⑤ Acquire a semaphore to insert created messages into the data area of a shared memory buffer.
5) ⑥-⑧ Paste messages in the data area of the shared memory and change the semaphore counter.
6) ⑨ Disable the relevant channel and terminate services in case of a one-way service request.
7) ⑩ Wait for response messages from the server if responses are required.
8) ⑪ If a response message is received, process relevant messages.
9) ⑬ Disable the relevant channel when services are provided.

```
// Client Side IIOP
Declaration of variables
SharedMemoryBuffer {
    Lock[N] MUTEX_LOCK;
    Channel[N] Integer;
    msg[9][28] GIOPMessage;
    keelAlive Integer;
    reply_expected Integer;
}
SharedPtr *Ptr;
Initialization
      Ptr = MemoryMap(NULL, fd, flags);
Action
①: Make Client Shared Memory Buffer Create
②: Channel Allocation for Client Processes
③:
④: Construct GIOPMessage
⑤: Semaphore TryWait(Lock)
⑥: Write GIOPMessage to SharedBuffer
⑦: Semaphore post(Lock)
⑧: Semaphore wait(Lock)
⑨: if oneway is not set then
⑩:       Wait for ReplyMessage from Server
⑪:       Handle ReplyMessage
⑫: fi
⑬: Channel Close
```

Ⅳ. Performance Analysis

In order to conduct a performance analysis, we use data types defined in the OMG specifications for server programs written by null function. The test system consists of a server and four clients. We compared CORBA throughputs by requesting services to all four clients simultaneously and by receiving results from them. Since the conventional generic and commercial CORBA platforms do not support a shared memory-based inter-ORB protocol, the comparison of its performances was conducted on the same platform of Solaris 2.7, UltraSparc 167MHz and 192MB memory.

First, omniORB(3.0.4)[6] developed by Lucent Technology, and MICO(2.3.5)[7] developed in Germany for open releases were compared to ensure the reliability of platforms. In comparing the time spent to call functions one thousand times employing various data types of five clients[8], the performance of the CORBA based on our protocol (uniORB) achieved a 250% improvement over MICO implemented by C++, and a 15% improvement over omniORB for each data type, as shown in [Fig. 5].

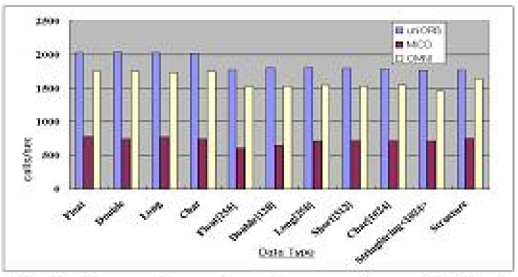

Fig. 5. Comparison of performance in multi-client

Second, the performance of the proposed shared memory-based inter-ORB protocol was compared with that of the platform-based on TCP/IP and UNIX domain sockets.

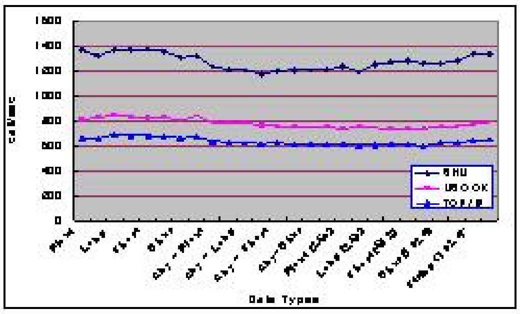

Fig. 6. Comparison of throughput by data types

The first test measured time spent on a client calls for one thousand times of services of various data types. The results by data types are shown in [Fig. 6]. This figure shows that the shared memory-based inter-ORB protocol improves its throughputs by 50% higher than the UNIX domain socket, and by 200% higher than the TCP/IP based protocol.

We can conclude that there is no distinction between marshalling/demarshalling time for all data types and the time required for the ORB to find and to activate relevant objects in every methods. However, our protocol does not incur any overhead by not copying data inside the kernel, as contrast with TCP/IP-based protocol.

The second test is aimed at comparing throughputs by data sizes. A client changes the data sizes of the character types to be transferred. The sever side implementation codes compare the size of the data received and return its result values. [Fig. 7] shows the time spent on service requests/responses of one thousand times.

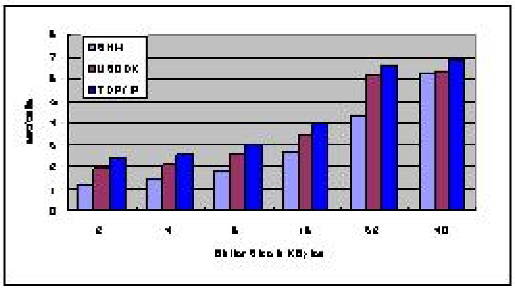

Fig. 7. Comparison of throughput by data size

Shared memory-based inter-ORB protocol shows more improved throughput when data size is small. The reason for this : when data size is large, the time required to transfer the data is relatively longer than the time required to process ORB and to marshal/demarshal.

The performance analysis in this paper shows that the shared memory-based inter-ORB protocol is better fit for the application that has enough system memory and frequently processes small-sized data.

V. Conclusion

The shared memory-based inter-ORB protocol proposed in this paper is designed to develop a higher-performance CORBA for communication systems. The designed protocol supports the same interface and can minimize the message transfer overhead in the same host environment.

Through performance tests with various data types, we can see that our protocol achieved a maximum 250% improvement over MICO, and 15% improvement over omniORB. Our protocol allows to design component architecture of communication systems software with high performance.

To practically apply the CORBA platform based on our protocol to communications software, further research is required on the optimization of memory usage, on development of services like notification, and on the shared memory processing techniques to allow tens of clients to access one server simultaneously.

참 고 문 헌

[1] OMG, *The Common Object Request Broker: Architecture and Specification*, Minor rev. 2.3.1, OMG Document Formal/99-10-07, Oct. 1999.

[2] http://www.omg.org/technology/documents/corba_spec_catalog.htm

[3] I. Ahmad and S. Majumdar, "Achieving High Performance in CORBA-based Systems with Limited Heterogeneity," Proc. Int. Sym. on Object-Oriented Real-Time Distributed Computing, pp.350-359, May 2001.

[4] A. S. Gokhale and D. C. Schmidt, "Optimizing a CORBA Internet Inter-ORB Protocol (IIOP) Engine for Minimal Footprint Embedded Multimedia Systems," IEEE Journal of Selected Areas in Communications, Vol.17, No.9, pp.1673-1706, Sept. 1999.

[5] S. L. Lo, D. Riddoch, and D. Grisby, *The omniORB version 3.0 User's Guide*, AT&T Laboratories Cambridge, May 2000.

[6] http://orbit-resource.sourceforge.net

[7] Puder and K. Römer, *MICO - User and Reference Manual*, TR-98-031, Int. Computer Science Institute, Berkeley.

[8] P. Tůma and A. Buble, "Overview of the CORBA Performance," Proc. 2002 EurOpen CZ Conf., Znojmo, Czech Republic, Sept. 2002.

[9] A. S. Krishna, D. C. Schmit, K. Raman, and R. Klefstad, "Optimizing ORB Core to Enhance Real-time CORBA Predictability and Performance," Proc. Int. Symp. on Distributed Objects and Application, Sicily, Nov. 2003.

저 자 소 개

장 익 현(Ikhyeon Jang) 정회원

▪1984년 2월 : 서울대학교 계산통계학과 (이학사)
▪1986년 2월 : 한국과학기술원 전산학과 (공학석사)
▪1998년 2월 : 한국과학기술원 전산학과 (공학박사)
▪1986년 3월 ~ 1999년 2월 : (주)데이콘 책임연구원
▪1999년 2월 ~ 현재 : 동국대학교 정보통신공학과 교수

<관심분야> : 컴퓨터통신, 분산시스템, 인터넷응용

조 영 석(Youngsuk Cho) 정회원

▪1978년 : 서강대학교 철학과 (문학사)
▪1988년 : Louisiana State University (정보학석사)
▪1994년 : Louisiana State University (컴퓨터학 박사)
▪1980년 ~ 1984년 : 한국 후지쯔(주), Systems Analyst
▪1989년 ~ 1994년 : Louisiana State University, Computer Analyst
▪1994년 ~ 현재 : 동국대학교 컴퓨터·멀티미디어학과 교수

<관심분야> : 소프트웨어 재사용, 소프트웨어 개발 방법론, 설계패턴, 객체지향 방법론, 정보검색