

A CLIENT/SERVER TAPE ROBOT SYSTEM IMPLEMENTED USING CORBA AND C++

Y. MORITA, K.B. URQUHART[†], Y. WATASE

*Computing Center
KEK, National Laboratory for High Energy Physics, Tsukuba, Ibaraki 305, Japan*

The Common Object Request Broker Architecture (CORBA) is an object-oriented communications framework which allows for the design and development of distributed, object-oriented applications. A CORBA-based implementation of a distributed client/server tape robot system (KIWI Tape Robot) is developed. This approach allows for a variety of data-modeling options in a distributed tape server environment. The use of C++ in the handling of HEP data which is stored in a Hierarchical Mass Storage System is demonstrated.

1 Introduction

Handling and accessing a large amount of High Energy Physics data on magnetic tapes in a distributed environment has been a central issue for KEK UNIX batch users. In CHEP '92, we have reported a client/server file distribution system using 8mm tape robot and a TCP/IP socket based file request queuing mechanism.¹ The system has been used as a convenient and handy solution for user files archiving and restoring for simulation batch users and for network based file system backup for individual workstations.

As the need for a larger storage space and CPU resources increases, KEK Computing Center is now preparing for a new UNIX based workgroup cluster server system with a high performance tape library and a HSM storage manager. To meet the requirement of achieving high performance and large scale I/O in a multi-user, multi-access environment with a various range of file size and I/O speed, we are faced to develop a new layer of file service interface which incorporates the new features of the storage system hardware.

Our objectives of developing a new client/server tape system are as follows: create a C++ library which provides for direct and uniform file serving, provide a transparent and universal user interface which incorporates various schemes of tape file serving mechanisms including migration, staging, direct tape access and HSM, and convert the existing KIWI Tape Handling library into fault-tolerant system.

2 KIWI Tape Robot Testbed

KIWI Tape Robot Testbed, reported in reference 1, consists of an EXAbyte EXB120 tape robot system and four 8mm tape drives. Each drive is connected to a separate workstation and cartridge handling manager (chsd) and staging managers are running separately on each workstation. When a user requests a file to be staged in, file clerk sends a request to mount a tape to chsd, and the file on the tape is copied to the staging

[†] On leave from Simon Fraser University, Canada.

disk from the tape drive by using *rmt* calls or NFS mount. The staging clerk, *chsd*, and tape serving workstations can be configured freely so that the files are served over the network or user's computing workstation can be the same workstation as the tape server workstation. [Figure 1]

User commands for file archiving and restoring are: *chs_rdump*, *As-restore*, and *chs_archive*. Also *chs_stagein* and *chs_stageout* are provided for file staging.

3 KEK's Next Central Computer and Hierarchical Storage System

KEK plans to install anew UNIX-server based central computer server system in January 1996. The system is HITACHI multi-CPU UNIX servers (PA-RISC and Alpha) for 11 separate **workgroup**. Total CPU power will be 15,800 **SPECfp92**, and processors in a same **workgroup** cluster are **interconnected** with an 1.5 **GB/sec** cluster switch. Total capacity of the RAID system is 775 **GB** with an **I/O speed** of 4.5 MB/sec. The system comes with two sets of tape library system with 16 SONY 1/2" helical tape drives. The drive can record up to 42 **GB** per cartridge. Sustained data transfer speed is 12 MB/sec in uncompressed mode. A **file marking feature** provides the positioning of any **files** on tape within 60 seconds. Total capacity of the tape library is 20TB.

To accommodate wide range of **file** size and I/O speed requirement in storing files into this tape library, **two** methods of file access are originally provided: one is to open the files on each tape through function calls *mtopen()*, *mtread()*, *mtwrite()*, and *mtclose()*. This set of function calls can enjoy the sustained I/O speed and the file positioning **performance** of SONY 1/2" helical scan tape drive at its full strength. However the **file access** is limited to the processors which are directly **connected** to the tape drives.

The other way of utilizing the tape library system is by **OmniStorage** HSM software. Files on the tape library are migrated to the RAID disk system and the disks are served to each user by either locally or by network using **DCE/DFS**. **This** method allows users to **access** their **data** in a transparent fashion, but the total **performance** of the **file utilization** heavily depends on the average access pattern of the files, size of the user files and the caching disks, and the **I/O** speed of the RAID disks. For those who want to utilize the **full** performance of the SONY tape drive in a sequential read or write of large data files over -1 **GB**, the speed of the RAID disks **will** be the bottleneck

4 Design of Abstract File Access Layer

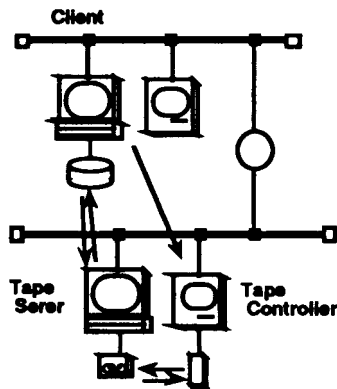


Figure 1: Components of client/server tape system.

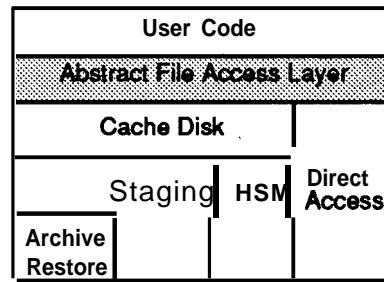


Figure 2: Abstract file access layer

Table 1: **Function list of abstract file access layer for various tape services**

Function	mode	HSM	staging	Direct
Read Hello	Async	media online	stagein	load()
Read Ready	sync	open(dk)	open(dk)	open(mt)
Read	sync	read()	read()	read()
Read Done	sync	close()	close()	close()
Read Bye	Async	media offline	clean up	unload()
Write Hello	Async	media online	- N/A -	load()
Write Ready	sync	open(dk)	open(dk)	open(mt)
write	sync	write()	write()	write()
write Done	sync	closet)	close()	close()
Write Bye	Async	media offline	stageou	unload()

Given the configuration of the next generation **workgroup** server system above, and to **prepare for** a potential request of C++ user interface for the tape library system in near future, we decided to redesign the KIWI **client/server** tape system using **C++** and **CORBA**.²

Several basic design questions are addressed: will our system provide an object persistency mechanism in a generic way, or shall we restrict ourselves to give users the **meta-physical file** images and let the user utilize the **file** image object mapping onto their own data structure? Under heterogeneous machine type **environment**, should we provide a generic data type conversion mechanism? When should the data type conversion occur?

Our approach to these questions is as follows: size of the **HEP** raw data imposes a lower limit on solutions of the storage technologies, that is, we want to utilize the **sustained** hardware performance at its **full** strength. This means the user, especially those who design the data reconstruction program **framework**, **should** be aware of how to utilize the RAID and **helical** scan tape drive technologies in an efficient way. Also the description of the data modeling and providing an automatic data type conversion for a sequential access of tape is too expensive and therefore the question must be left out to each experiment group.

Utilization of the **"file"** image in typical **HEP** programs is relatively straight forward and does not require a technical detail of how **files** are stored on the hierarchical storage system. Figure 2 shows the concept of the abstract file access layer and the table 1 lists **the** abstract functions of **file** access for various types of underlying file storage method. Although the **HSM** provides a clean and transparent interface of **file** system to user, features like **file** pm-caching or scheduled utilization of the caching disks will still be **useful** and efficient for sequential access of large amount of data. By using our abstract **file** access layer, it becomes easy for the user program to debug and test the **file access** mechanism for a small set of data before running a production job for a large set of raw **data**.

5 Client/Server Tape Robot with CORBA

CORBA architecture has a rich set of features for easy client/server development environment for **C++** and can design data transfer interface with other languages such as **Fortran** by means of Interface Definition Language (**IDL**). User-defined exception handling can also be built into the system together with the system-level exception

handling, and it allows to construct a fault tolerant client/server communication mechanism in a simple way. We picked up a CORBA-compliant object request broker product called *ORBeline*¹.

Figure 3 shows an example of **IDL files** which defines the interface of **file** loading operation **between** client and server. Two parameters *DataSet* and *FullPath* are defined for **specifying** the location of the file, and an interface *Load()* defines the actual method of providing the **file** on the tape on each type of tape storage manager.

IDL file: **TapeManager.idl** (define the client/server interaction)

```

//Parameter description:

struct TapeFile {
    string DataSet;
    string FullPath;
}

// Transaction description:

interface TapeManager {
    boolean Load (inout TapeRequest request) ;
}

```

Figure 3: Example of **TapeManager** client/server IDL file.

The **CORBA IDL** compiler takes this **file**, **TapeManager.idl**, and generates four segments of **C++** skeleton code: **TapeManager_client.hh** and **TapeManager_client.cc** for the **client**, and **TapeManager_server.hh** and **TapeManager_server.cc** for the server. **CORBA** client and server operation can be bound by **linking** with the generated files.

One the **IDL** is established, client code makes a call to **_bind()** of **CORBA** Class **TapeManager**. Successful return of the **_bind()** call ensures the client the file is ready, and client **code** is now safe to attach the **FullPath** into whatever the **file** stream user code wants to Use.

Locate() meshed of **KIWIServer**

```

CORBA::Boolean KIWIServer::Locate(const KIWIFile &theFile)
{
    char *fullPath = NULL; /* To hold the full path name to the disk file. */
    :
    <Determine which tape holds the &taut request by the user>
    :
    <Summon the tape robot to mount the tape on a tape drive>
    :
    <Read the dataset from the tape and put it onto a hard disk>
    theFile.Where(fullPath); /* Return the path to the disk file. */
    return(CORBA::TRUE);
}

```

Figure 4: **Locate()** method for the **KIWIServer** subclass

Server side of the **TapeManager** code requires the actual implementation of **Load()** method for serving the **file** on the tape. **Actual way of implementing file service** mechanism may differ for different type of file storage, and therefore may construct a subclass of **TapeManager**.

¹PostModern Computing: <URL: <ftp://labrea.stanford.edu/pub/pomoco/>>

For the files whose **pathnames** belongs to the KIWI tape storage, subclass **KIWIServer** is derived from **TapeManager**. Presence of the **KIWIServer** is announced to the network by a call to **CORBA** basic object adopter (BOA) **impl_is_ready()** method. Actual method of attaching files on tape to a staged file is implemented in **KIWIServer::Locate()**.

3 Summary

We have modified the **client/server** mechanism of existing KIWI tape server system using **CORBA**. Writing client/server system using **CORBA** is simple and straightforward, and communication between them can become more reliable by replicating the server process by using **CORBA** framework. Defining a single **C++** class library of abstract **file** access layer can simplify the actual implementation of **file** and cartridge control methods for various types of storage scheme.

In this layer of software, we choose to map the file image directly into the user data access layer so that the I/O performance of the tape drive would not be hindered by the overhead of data type conversion in the **CORBA**-compliant software. This is mainly due to give a freedom to the user group to choose the data modeling and to avoid data serving overhead associated with **CORBA**. We are planning to measure the actual overhead of this conversion under a cross platform environment, and will provide another layer of software which incorporate the data modeling into the **CORBA** framework. This will also lead us to seek the possibility of our abstract **file** access class into the existing object persistency projects.

References

1. **Y. Morita** et id., *Robotic Tape Server for Distributed UNIX Computing, Proceedings of the International Conference on Computing in High Energy Physics '92, Annecy, France, 21-25 September 1992, CERN.*
2. Object Management Group, *The Common Object Request Broker: Architecture and Specification, Revision 1.1*, OMGTC Document 91.12.1, 1991. See also <URL: <http://www.acl.lanl.gov/sunrise/DistComp/Objects/corba.html>> and <URL: <ftp://claude.ifi.unizh.ch/pub/standards/corba/spec/>>.