

Sending Commands and Managing Processes across the BABAR OPR Unix Farm through C++ and CORBA

G. Grosdidier¹, S. Dasu², T. Glanzman³, T. Pavel⁴
(for the BABAR Prompt Reconstruction and Computing Groups)

¹ LAL-IN2P3-CNRS, Université Paris-Sud, Orsay, France (Currently at SLAC)

² Department of physics, University of Wisconsin, Madison, Wisconsin, USA [‡]

³ SLAC, Stanford, California, USA [§]

⁴ On leave from SLAC, Stanford, California, USA

Abstract

BABAR is operating a Unix farm for the quasi-online reconstruction of B-Factory events. For sending commands through this widely distributed system, we implemented a set of lightweight client/server tools based on CORBA as a communication package within C++ applications. The ACE/TAO library used for this implementation includes a multi-threaded environment for the client side. Details about the requirements and design will be provided, together with performance comparisons against ssh.

Keywords: CORBA, BABAR, ACE/TAO, C++, Parallel Farms

1 Introduction

1.1 The BABAR OPR Farm

The Online Prompt Reconstruction (OPR) farm in BABAR was designed to process raw data events soon after they are available, bunched in some container files, within a latency of about two hours. This Unix farm described in much more detail elsewhere [1, 2], can be sketched in this way: a single master process orchestrates the operation of each processing instance which involves typically 100-200 nodes. This paper is aimed at describing the mechanism and relevant applications used to achieve interprocess control and monitoring.

1.2 The Requirements

In addition to the above basic functionalities, we required this system to be:

- fast: broadcasting a command over the whole farm in a few seconds
- lightweight: limited to executing simple commands (launching a process, returning its log file name, or the contents of the latter)
- flexible, so that one can build sophisticated macros
- robust, so that unreliable nodes do not interfere with healthy ones
- improve process control: all processes on the remote nodes are owned by the same userid, so that inside of the OPR team, anyone could manage any process
- reasonably secure: a limited command library with aliases (true commands are masqueraded), together with an Access Control List (ACL) specific to every command
- reliable: automatically started at boot time, and restarted after crash

[‡] This work was supported in part by Department of Energy contract DE-FG02-95ER40896.

[§] This work was supported in part by Department of Energy contract DE-AC03-76SF00515.

1.3 The Design Components

Given the overall software context in BABAR (Object Oriented plus distributed computing and Unix [3]), and the high level of interaction of C++ with the underlying OS, it was rather straightforward to select this language for the server. But the key choice was really about using CORBA for the message layer [4, 5]. Indeed, the ACE/TAO package [7, 8] was already used in BABAR, and more particularly in OPR [6]. Its selection as a C++ CORBA API was straightforward as well.

Perl was first considered as the language in which to write the client. However, C++ was selected for 3 reasons: to have a uniform programming language for client and server; multi-threading was easily easy in both languages; and, most importantly, C++ offered a reliable CORBA implementation.

Both parts (server and client) are now built and running through:

- Solaris 2.6 using the native C++ compiler (version 4.2)
- ACE 5.04 and TAO 1.04, using the native Solaris threads
- Perl 5.005 for the wrapper script.

We also systematically used containers classes from the Rogue Wave Software Template Library together with the Rogue Wave Tools.h++ library (version 7.08).

2 The Bits and Pieces

2.1 Overview

The Global Farm Daemon (GFD) consists of a “server” application, that runs permanently on every compute node, while a “client” can run on demand on any node node. The client features will be described mainly for the MT¹ version, which actually allows to see the OPR farm (for the command layer) as a single entity. Communications between these two applications are handled through 2-way CORBA calls, conveying simple messages made of string sequences - returned messages can be multi-line strings.

2.2 TAO: the naming services and the tools

Exactly one GFD runs on every compute node. It is identified with a unique name, eg “Gfd-bronco012” refers to a GFD running on node bronco012. There is a TAO Name Server available for the whole BABAR experiment on the SLAC site, which also prevents name duplication. A set of high level TAO Tools is provided as a service library (described in another paper [9]), to provide access to TAO components in a compact way.

2.3 The Server Components

2.3.1 The GFD command library

Every available command is accessible only through an alias. each alias is defined in the command library, together with several attributes:

- the complete command definition
- the name of the subdirectory where logging and error information is stored
- the list of users allowed to access the command (ACL).

This library is currently made of a human readable file, loaded by every GFD during initialization. A special command allows the GFD to reload a new version of this file after an update.

¹Multi-Threaded

2.3.2 Returning information to the client

The server must be a robust and lightweight application and able to handle requests quickly. Thus it normally operates in asynchronous mode (sending a task as a background process, and not waiting for the output). However, the following modes have therefore been made available:

1. commands run as an asynchronous detached process logging its output into a file whose name is sent back immediately to the client;
2. commands run asynchronously, but nothing is sent back, and the log filename is stored for later retrieval;
3. when sent from within an MT client, commands can run synchronously with the result sent back directly to the caller.

Mode 1 is used most of the time, so that some checking can be done over the result of the command. Mode 3 is used when fast feedback is required. Mode 2 is the fastest, sending the command to every node within the smallest time window.

2.3.3 The command processor

The command processor is currently made of several layers (executed in order):

- authentication layer
- command parsing: special commands (direct “system calls”) are caught and processed directly.
- option parsing: several switches allow, for example, verbose log facilities, a no-execute mode (to check the whole chain), an option to specify an alternate log file and another one to provide an alternate command location.
- execution layer: the command string is built and wrapped into a “system()”-like call².

2.4 The Client Side

The client is used for many very different purposes:

- to check the status of the servers
- to launch specific tasks on all the farm nodes simultaneously and to monitor them
- to stop or kill remote processes.

In most cases, the client sends a specific command (along with one or more parameters) to many nodes at the same time. However, it is sometimes useful to send a command (eg to kill a given process on a given node) to only one node. Therefore, two versions of the client program currently coexist which are both accessible through a Perl wrapper: a single-threaded version, for reaching a single node, and a multi-threaded version, which tackles a string of nodes simultaneously in one shot. In both versions, the client executes a non-blocking loop to contact each GFD. Thus, there are no delays if a misbehaving GFD is encountered. The functions achieved by the client program are:

- selecting the target GFDs and collecting their TAO IORs
- sending the selected command alias, options and relevant information through a CORBA call to every target GFD
- receiving the returned data from the same CORBA call (if any)
- processing the returned information (if any)

The multi-threaded version saves resources, mainly memory and TAO name server calls, compared with the single-threaded one. However, the client program requires much more subtle and thoughtful coding. Some of the more frequent traps seem to be:

²The genuine “system” call available in the standard C library is not MT safe on both Solaris 2.6 and 2.7.

- Ensure that TAO is correctly initialized at run time to be MT safe (check the “-ORBResources tss” options).
- Avoid using special CORBA types (CORBA::Object_var or CORBA::String_var) outside of the message handling section itself, as these types cannot be declared static.
- Every utility or tool used in the thread code itself must be checked for MT safety, or moved elsewhere.

3 A few Technicalities

3.1 Performance

Issuing the “uname -a” command to 100 nodes synchronously requires, in elapsed time:

- 230 sec. when using “ssh -x”;
- 25 sec. when running the single threaded client (the Perl wrapper achieving the loop over the nodes);
- 7.6 sec. when running the multi-node client (the latter achieved the loop over the nodes sequentially);
- 4.7 sec. when running the multi-node client in MT mode.

3.2 Some Code Excerpts

3.2.1 The IDL declaration for the CORBA interface

```
interface OprFarmDaemon {
    typedef sequence<string> psList;
    string start(in psList argList, out string outstring); }
```

3.2.2 A snippet of the Client program

```
[...]
// farmDaemon will be the reference of the target GFD
OprFarmDaemon_var farmDaemon = OprFarmDaemon::_narrow (obj, env);
char *returnData, *optionalData;
// Creates the string sequence containing command+parameters+options
psList argListL(argList);
// Issues the CORBA call
returnData = farmDaemon->start(argListL, optionalData);
cout << returnData << endl;
[...]
```

3.2.3 Several snippets showing the Server code

```
// Class declaration
class OprFarmDaemon_i : public POA_OprFarmDaemon {
public:
[...]
```

```
// Methods
char* start(const OprFarmDaemon::psList &argList,
            CORBA::String_out outstring,
            CORBA::Environment &_tao_env=CORBA::Environment::default_environment());
private:
```

```

    // Data member
    string _ustring; }
    // Class definition
    // Constructor
OprFarmDaemon_i::OprFarmDaemon_i (...) {
    _ustring = string(" ", 8192); }
    // Methods
char* OprFarmDaemon_i::start(const OprFarmDaemon::psList &argList,
    CORBA::String_out outstring, CORBA::Environment &_tao_env) {
    // Processing the command ... then returning ...
    string ostring = "Some optional Info to return";
    outstring = CORBA::string_dup(ostring);
    string rstring = "Main output to be returned";
    _ustring = CORBA::string_dup(rstring);
    return (char*) _ustring.c_str(); }

```

4 Conclusions and Future Evolution

The current versions of GFD components have been successfully in production for two months. The server part has proven to be robust and reliable. The command library has already been extended several times without requiring redesign. No memory leaks have been found ! The multi-threaded version of the client is also robust and scalable - it has been tested successfully with up to 250 target nodes at one time.

However, our experience while using CORBA within the ACE/TAO environment deserves some remarks. It requires a significant learning curve of the order of weeks, not days. The documentation is also weak (this is hopefully getting better). Both of these items must be taken into account when considering the use of CORBA within a large project.

Above all, the technique of using CORBA for this application has proven to be successful. Therefore, we now plan to:

- run the GFDs also on all OPR servers for monitoring
- extend its use to a new structure called “Global Farm Manager” to drive and coordinate the entire farm
- propose to use the GFD in the DAQ system for starting and monitoring processes.

References

- 1 T. Glanzman et al., “The BABAR Prompt Reconstruction System”, Paper No. 52, CHEP’98, Chicago, Autumn 1998.
- 2 T. Glanzman et al., “The BABAR Prompt Reconstruction System, or Getting the Results out Fast: an evaluation of nine months experience operating a near real-time bulk data production system”, Paper No. 288, these proceedings.
- 3 T. Schalk, “BABAR Software Overview”, Invited talk, this conference.
- 4 S. Yang, CORBA evaluations for the BABAR Online System”, Paper No. 189, CHEP’98, Chicago, Autumn 1998.
- 5 S. Metzler et al., “Production experience with CORBA in the BABAR experiment”, Paper No. 290, these proceedings.
- 6 S. Dasu et al., “Event Logging and Distribution for the BABAR Online System”, Paper No. 74, CHEP’98, Chicago, Autumn 1998,

- 7 D.C. Schmidt, "An Architectural Overview of the ACE Framework: A Case-study of Successful Cross-platform Systems Software Reuse",
USENIX login magazine, Tools special issue, November 1998.
(<http://www.cs.wustl.edu/~schmidt/ACE.html>)
- 8 D.C. Schmidt et al., "The Design and Performance of Real-Time Object Request Brokers",
Computer Communications, Volume 21, No. 4, April, 1998.
(<http://www.cs.wustl.edu/~schmidt/TAO.html>)
- 9 S. Dasu et al., "Event Logging and Distribution for the BABAR Online System",
Paper No. 138, these proceedings.