CORBA/RMI Issues in the Java implementation of the Nile distributed operating system

F. Handfield[1], *D. Mimnagh*[1], *M. Ogg*[1,2], *L. Zhou*[3]

[1] University of Texas, Austin Texas 78712, USA
[2] now at Bell Labs, Lucent Technologies Inc., Murray Hill New Jersey 07974, USA
[3] University of Florida, Gainesville Florida 32611, USA

Abstract

Nile is a high performance, heterogeneous, fault-tolerant, distributed operating system that gives users transparent access to distributed computer resources for processing high energy physics data. Initially, we implemented the Nile Control System using CORBA. As this implementation was written in Java, with little effort we later re-implemented it using RMI for the distributed object transport. Our experience using Nile in the CLEO experiment has shed new light on service, performance, scalability, and programming issues of these two technologies. We describe the ways we have met challenges from CORBA and RMI and their effects on implementation, performance of resource management and fault-tolerance.

Keywords: distributed computing, Java, CORBA, RMI

1 Description of Nile

Nile has been described elsewhere [1, 2, 3] but for completeness we describe its salient features here. Nile is a distributed operating system that gives users transparent access to scalable distributed computing resources that appear to the user to be a single "virtual computer". Nile is fault-tolerant, meaning that the onus is on the system and not the user to compensate for processor failures or network partitions. The motivations for Nile's design were the computing and storage requirements of the CLEO experiment at the Cornell Electron Storage Ring (CESR), but we have taken great care to ensure that Nile can be used easily not only by other elementary particle physics experiments, but also by other related applications with similar parallelizability.

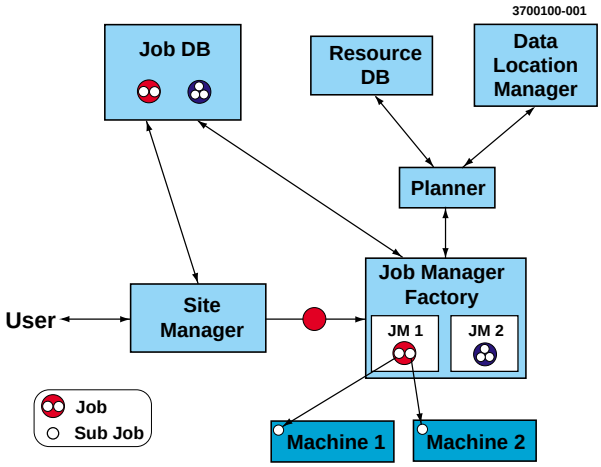

Figure 1: Nile Control System Architecture

The key feature of Nile is that a user's computing Job is broken down into many Subjobs, each of which executes independently and in parallel on as many different processors as are available. After all Subjobs have completed, a Collection task assembles the results so that the user sees the results of only one Job. The architecture of the Nile Control System (NCS) is illustrated in Fig. 1.

The user submits a Job to the Site Manager via a Java Graphical User Interface (GUI). The Site Manager creates a Job Manager instance using any available Job Manager Factory. The Job Manager manages the Job until its completion, using the Planner to obtain CPU and data resources for the job. In turn, the Planner receives information from the Resource Database and Data Location Manager. When resources are available the Subjobs are started on the assigned individual Subjob Processors (i.e., the physical processors). Each Subjob is small enough that the loss of that Subjob and reassignment to another processor in the event of failure does not unduly waste resources. Each Subjob is a leaf in a tree-like data structure that represents the parallelization of the Job. Subjob failure is managed automatically. A failed Subjob becomes a node and is rescheduled as one or more new Subjobs.

The first distributed object implementation of Nile[1] was written in C++ using CORBA [2, 3]. The target ORB, Electra [6], used Isis [5] as its transport. When Isis was no longer available, we decided to abandon object replication, re-implement Nile in Java, but still use CORBA and keep the original IDL object interfaces. At the time, it seemed quite likely that some of Nile's objects would still be implemented in C++, so CORBA appeared to be an appropriate choice. As it turned out, Nile was eventually implemented entirely in Java, so there was no overriding reason to stay with CORBA, except that the IDL object interfaces already existed.

2 Distributed Object Architecture

There are several reasons why Nile (and other similar systems relying on commodity hardware) should use a distributed architecture. Nile is necessarily a distributed system – that is how the parallelization of Subjob Processors is achieved. More pertinent is why the Nile Control System itself should be distributed. This is done for reasons of scalability (arbitrarily many Job Manager Factory objects can be used where each Job Manager instance controls a single job); resilience, by allowing failed components to be restarted as needed, whereas a monolithic architecture would more likely lead to catastrophic failure; separation of function, such as placing databases separately from other components; availability through replication (although this feature was largely abandoned, we wanted to keep the replication option available). Distribution also allows us to add more stringent fault tolerance properties by exploiting the many processors to replicate crucial NCS tasks. Most distributed systems are based on some form of *middleware*, which is a layer of software between the application and the network transport that shields the former from the latter, in particular transparently handling such tedious business as marshalling and unmarshalling, byte swapping, error handling, and raising exceptions. The principal standards-based, or *de facto* standard middleware systems are CORBA [7], RMI [8], and DCOM [9]. Since Nile had already been implemented in Java, the latter was not considered further. While middleware systems handle most of the tedious aspects of distributed systems, they cannot and should not disguise the fact that remote method invocations introduce additional failure modes, requiring different techniques for error handling and failure detection to achieve robust, reliable operation.

[1] There was an earlier version [4] that used a proprietary replicated transport [5] with C++ wrappers.

3 Nile's CORBA Implementation

The IDL to Java language binding [7] exactly specifies the Java interfaces and classes generated by an IDL compiler. Therefore, it should be possible to use any available Java ORB without changing the application code. In practice, some parts of the CORBA specification were under-specified, requiring small adjustments to work around differences between ORBs. The main issues encountered programming a CORBA application in Java were:

- The CORBA object model, at least compared to the flexibility of Java, is rather restrictive. For instance, there is a complete separation of data structures and interfaces[2]; interfaces do not extend, and classes do not implement, `java.io.Serializable`, making persistence (a trivial operation in Java) rather cumbersome; all classes are `final` and member data are `public`, thus making it more difficult to achieve flexible object models.
- Certain CORBA operations, such as `ORB.resolve_initial_references()`, are under-specified meaning that some ORB-dependent code is necessary, or else such functions should be avoided altogether.
- Any optimization when a CORBA object is used locally is not part of the specification.
- CORBA's Name Service, CosNaming, is a single point of failure. Attempts have been made to build a replicated Name Service [11] while still implementing the CosNaming interface. However, this is not part of the CORBA standard.

Notwithstanding these difficulties, a successful and robust implementation of Nile was achieved. The fundamental fault-tolerant requirement, namely restarting failed Subjobs, was met. Nevertheless, there was a lingering deadlock problem, which we were reasonably convinced was a bug in the ORB, that we were not able to resolve satisfactorily.

4 Nile's RMI Implementation

Remote Method Invocation, RMI, has been part of the Java specification since JDK 1.1. It has at least two distinct advantages over CORBA: RMI uses normal Java object semantics, and so the full capability of Java can be used; since RMI is part of the Java language the process of filing a bug report is less ambiguous, should a problem arise.

The RMI implementation of Nile was a prototype of limited functionality. However besides performing extremely well while running a CLEO Monte Carlo farm [12], it provided useful insight into the simplifications that could be achieved with RMI. A particular insight was gained from the structure of the Subjob tree. In the CORBA implementation, nodes and leaves of the tree implement the same interface – nodes are local within the Job Manager, whereas leaves are remote in the Subjob Processors. In fact, CORBA tries to mask the distinction between a local and remote object, which is not necessarily a good thing [13]. In particular, whereas a local object should be very light, and therefore suitable for building into a deep and wide tree, a remote CORBA object may be very heavy (in terms of resources such as memory and threads) and therefore very unsuitable for such a purpose. The RMI version does not have this problem, because all Subjob objects are local. If a tree and a distributed architecture are both to be kept, the solution is that non-remote objects must be truly local.

5 Future Work

Investigation of RMI has led us to consider Jini [13] which appears to mitigate several fundamental issues encountered building reliable distributed systems in general and Nile in particular. The

[2] This restriction largely goes away with the CORBA 2.3 "Objects by Value" [10] specification.

Jini Lookup Service offers many benefits: it provides the functionality of the Name Service; is transparently replicated, removing an insidious single point of failure; use of attributes allows it to be used as a general purpose object database. Jini's leasing mechanism and distributed events would be very useful in detecting and communicating failures.

Managing CLEO III data processing and analysis will be the next step in Nile development. To accomplish this we will improve scheduling and data management, and add comprehensive security and wide-area capability.

6 Conclusions

We have successfully implemented and used Nile as a CORBA application, written in Java. While CORBA has been mostly satisfactory, it has raised some issues of distributed object systems. A prototype RMI system largely corrects these problems, though its functionality is somewhat limited. Our work with RMI leads us to believe that we can re-architect critical parts of Nile, using technologies such as Jini, so that Nile has all the advantages of a distributed system with very few, if any, of the disadvantages.

References

1 M. Ogg, G. Obertelli, F. Handfield, and A. Ricciardi. Nile: Large-scale distributed processing and job control. In *Proc. Int'l Conf on Computing in High Energy Physics*, Chicago, IL, September 1998.

2 F. Previato, M. Ogg, and A. Ricciardi. Nile's distributed computing site architecture. In *Proc. Int'l Conf. on Computing in High Energy Physics*, Berlin, paper F360, 7–11 April 1997.

3 A. Ricciardi, M. Ogg, and F. Previato. Experience with distributed replicated objects: The Nile project. *Theory and Practice of Object Systems*, 4(2):107–117, 1998.

4 M. Athanas and D. Riley. The Nile fast-track implementation: fault-tolerant parallel processing of legacy CLEO data. In *Proc. Int'l Conf. Computing in High Energy Physics*, pages 164–168, Rio de Janeiro, 12–18 September 1995.

5 K. P. Birman and R. van Renesse. Reliable distributed computing with the Isis toolkit. *IEEE Computing Society Press*, 1994.

6 S. Maffeis. *Run-Time Support for Object-Oriented Distributed Programming.* PhD thesis, Universität Zürich, 1995.

7 Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.2*. February 1998.

8 Sun Microsystems Inc. *Java^{tm} Remote Method Invocation Specification, Revision 1.7*. December 1999.

9 N. Brown and C. Kindel. *Distributed Component Object Model Protocol – DCOM/1.0*. Internet Engineering Task Force, January 1998.

10 Object Management Group. *CORBA/IIOP 2.3.1 Specification*. October 1999.

11 L.C. Lung, J da Silva Fraga, J-M. Farines, M. Ogg, and A. Ricciardi. CosNamingFT – A Fault-Tolerant CORBA Naming Service. In *18th IEEE Symposium on Reliable Distributed Systems (SRDS99)*, pages 254–262, October 1999.

12 R. Baker, L. Zhou, and J. Duboscq. AMUN: A practicle application using the nile control system. In *Proc. Int'l Conf on Computing in High Energy Physics*, Padova, Italy, February 2000. (to appear).

13 K. Arnold, B. O'Sullivan, R. W. Scheifler, J. Waldo, and A. Wollrath. *The Jini^{tm} Specification*. Addison-Wesley, 1999.