NILE's Distributed Computing Site Architecture

Fabio Previato, Michael Ogg and Aleta Ricciardi [1]

University of Texas at Austin, Austin TX 78712.

The goal of the NILE project is to manage distributed computing resources, making an arbitrarily large cluster of commodity computers appear to the user as a seamless uniprocessor environment. This paper describes NILE's software architecture.

Key words: Distributed Systems, Fault-Tolerance, CORBA

1 The NILE Project

The NILE project [2,4] is building a distributed computing solution for CLEO III [3]. The goal is to provide a self-managing, fault-tolerant, heterogeneous system of hundreds of commodity workstations, with access to a distributed database in excess of 100 TB. NILE will allow any resource to be used transparently by any member of the collaboration, from anywhere within the collaboration. NILE must outlive its development phase, adapt to and scale with changes in CLEO's computing needs, be easily maintained, and be able to incorporate new software components as they become available; this led us to a distributed architecture based on CORBA. To program replicated objects we are using Electra [5], an ORB built on top of a reliable communication substrate; currently, Electra has adaptors for Isis [1] and Horus [7].

2 NILE Architecture

In NILE's software architecture, each of CLEO's collaborating institutions is a *site*, with a local set of users, and computational and data resources that process *jobs* submitted by local users or other NILE sites. The preference is to use local resources only, but the great disparity in sites' resources will require

[1] The first two authors are supported by the NSF under grant ASC-9318151

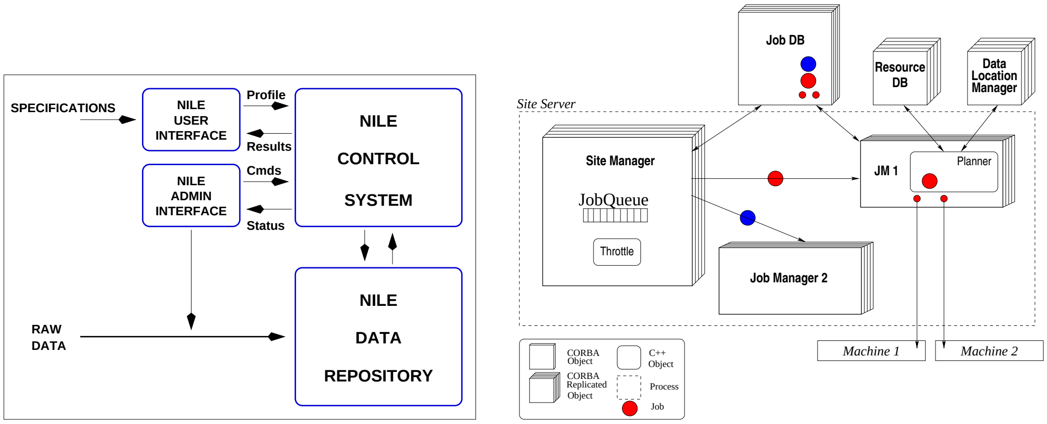

Fig. 1. The Nile Site and Control System Architecture

many jobs to be processed across multiple sites. Nile provides an abstraction of the computational and data resources as a single computing facility that allows users to be unaware of the underlying system complexity. Like a conventional computer, Nile has a *Control System* (OS), a *Data Repository* (storage system) and a *User Interface*. Nile provides this simple model of computation from a set of independent machines with different architectures and OS; each subject to failures. Our architecture ensures that each site remains available, and every job executes consistently to completion, despite the failure of components. Here we describe the Nile Control System.

3 Nile **Control System Architecture**

The Nile Control System (NCS) is a highly-available, fault-tolerant service which directs the computation of users' jobs. For each job the NCS makes an execution plan according to the available resources, then splits the job into a set of subjobs that can be scheduled and executed in parallel. The NCS consists of a set of distributed, replicated CORBA objects. The main objective of our architecture has been to improve system flexibility by decoupling components' dependencies and by enforcing well defined interfaces between them. The NCS contains the following objects.

The **SiteManager** is the main interface between the NCS and all other clients (users or other Nile sites). It receives job requests (submission, control, status) and arranges for appropriate processing by creating or contacting other objects. A job submission request is issued by users who describe a job's *computational* and *data* requirements within a *profile* object. These requirements include the code to be executed (as binary, source, libraries etc.), dependencies between code components (*i.e.*. initialization, computation, result collection), minimal system characteristics, an approximate cost of the job, and a highlevel description of the data the job uses (independent of physical location and naming). In response to a job submission, the SiteManager allocates a new JobContext by creating a JobManager, storing the JobContext into the JobDataBase and placing the JobContext into a queue of jobs ready to be started. At the end of this phase the SiteManager returns a unique job identifier to the user. A separate thread of the SiteManager manages the queue of pending jobs and starts them according to the current load.

The SiteManager is stateless, so the **JobDataBase** (JobDB) stores the state of jobs in the system (pending, processing or completed) together with context information as partial or complete output locations.

The **ResourceDataBase** (RDB) maintains the current state of the available hardware resources at the local NILE site. It obtains this information from each HostResourceMonitor, which provides status information such as processor type, memory, performance, CPU load average, disk space etc.

The **DataLocationManager** (DLM) is not strictly part of the NCS, but belongs with the Data Repository. It translates a logical data specification (from the job profile) into a corresponding set of physical data objects. This information is used by the Planner to determine the most suitable host(s) on which to run the subjob needing those objects.

The **JobManager** is responsible for managing and processing a single job. It obtains an *execution plan* from the Planner and applies it by dividing the job into a set of smaller subjobs to be executed in parallel. The JobManager monitors the state of each subjob. If a subjob fails, it can recursively request a new plan and split the failed subjob into smaller subjobs. At the end, the JobManager collects and assembles the results, then signals the SiteManager. If the execution plan involves resources at another site, the JobManager will pass the relevant subjobs to the appropriate SiteManagers.

The **Planner** produces an execution plan consisting of a list of subjobs, each having a host machine and a set of data objects. The Planner obtains the necessary information by querying the RDB and the DLM.

The **SubJobProcessor** creates a context and runs a given subjob, controlling its execution. The SubJobProcessor signals the JobManager when a subjob has terminated. Each physical machine has a **SubJobProcessorFactory** that creates and destroys SubJobProcessor objects for that machine.

Each processor is represented by a **HostResourceMonitor** (HRM), which sends usage and availability information to the RDB. Whenever a HRS appears because a new host has been activated or disappears because of a host failure, the RDB automatically receives the corresponding signal and updates itself.

4 Programming with Replicated Objects

Our experience developing NILE has proved that CORBA is a powerful programming model for distributed applications. It allows the use of objects residing on different machines, hiding complex communication details and architecture dependencies. Electra has proven to offer great advantages in adding replication to a distributed object system. Designers and programmers can concentrate on the interaction of CORBA objects, yet still realize the features of fault-tolerance through replicated objects. Nevertheless, in [6] we argue that in complex systems such as NILE, there are still some areas where it is necessary to be aware of the underlying communication layer so that subtle traps can be avoided. Also, although a CORBA-compliant ORB allows a system to be built without regard to the lower layers, when implementing a system with *replicated* CORBA objects, some critical and complex aspects of the group model necessarily appear to the programmer.

References

[1] K. P. Birman and R. van Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, 1994.

[2] D. G. Cassel and M. Ogg. Nile: Distributed Computing and Databases for High Energy Physics. In *NSF Grand and Naational Challenges Workshop*, March 1996.

[3] D. Miller et al. CLEO Collaboration. The CLEO III Detector. Technical report, Cornell preprint CLNS 94/1277, 1994.

[4] `http://www.nile.utexas.edu`. NSF, ASC-9318151.

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

[6] F. Previato, M. Ogg, and A. Ricciardi. Experience with Distributed Replicated Objects: The NILE Project. In *European Research Seminar in Advanced Distributed Systems*, March 1997.

[7] R. van Renesse, K. P. Birman, and S. Maffeis. Horus: A Flexible Group Communication System. *Communications of the ACM*, 39(4):76–83, 1996.