Demonstration of NILE

Fabio Previato, Michael Ogg and Aleta Ricciardi [1]
University of Texas at Austin.

We demonstrate NILE running on a network of PCs. NILE manages distributed computing resources, making an arbitrarily large cluster of commodity computers appear as a seamless uniprocessor environment. The demonstration has all the elements of NILE: client GUI, replicated Site objects, and multiple JobProcessors.

Key words: NILE, CORBA, replicated objects, PCs

The NILE project [3] is developing a distributed computing solution to provide a self-managing, fault-tolerant, heterogeneous system of hundreds of commodity workstations, with access to a distributed database. In [6] we describe the NILE Site architecture and in particular the NILE Control System (NCS). In this paper we give further details of NILE's status and describe a demonstration of the system on a network of PCs. We have successfully implemented most of the NCS that is now running on different platforms.

NILE already meets many of its requirements. The system is distributed onto a set of workstations and tolerates process, CPU or network failures. It hides unnecessary details from the user, such as the physical machine and data location. It increases the overall computation power by parallelizing tasks among the available machines. At present, NILE does not support computation across a wide-area network, and is using a very simple Data Repository implementation. To implement NILE we are using Electra [4], a fault-tolerant ORB built on top of Isis [2]. The main development platform is Intel x86 with Linux, but we also have working versions on SUN Sparc and Digital Alpha.

NILE can run on an arbitrary set of machines called *nodes*. Each node can be allocated as a *processor node* and/or a *system node*. Each processor node executes *subjobs*. System nodes run the set of CORBA object groups comprising the NCS. The number, identity and allocation of nodes can change dynamically as a result of administrative intervention or hardware failures, without losing consistency and service availability.

The number of nodes and their allocation is an important parameter. Increasing the number of system nodes increases the robustness of the system by raising the cardinality of each object group. Increasing the number of processor nodes increases the job execution speed-up by raising the level of parallelization. Normally of course, there would be significantly more (*e.g.* hundreds) of processor nodes than system nodes. Figure 1 shows an example of the NILE configuration table that specifies the set of NCS components and their distribution across available nodes. For each component, the table specifies the number of replicas and the set of nodes for its execution.

Assuming sufficient replication for each object group, NILE will run consistently despite any failures of a single process, a whole node or a network link. As long as there is at least one working instance for each object group, then that group is consistently available. The table describes only the initial configuration of the system which can vary at run-time. The availability applies to the *service* that NILE offers – a given subjob might be re-executed as the result of a failure, because subjob execution is not (generally) replicated.

NILE allows users to process a *job* specified by a *profile* object. NILE imposes very few restrictions on the type of job a user can submit. A job is defined by a code specification and a data specification. A code specification can describe binary programs, scripts, source files, libraries of functions or any combination. The user can also specify a set

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

Object group Name	Num. of Replicas	Nodes
ResourceDB	2	`croixfer, glandon`
DataLocMgr	2	`croixfer, glandon`
JobDB	1	any
SiteManager	3	`croixfer, glandon, roselend`
ProcessorFactory	1	`queribus`
ProcessorFactory	1	`roselend`
ProcessorFactory	1	`saisies`

Fig. 1. Example of NILE's configuration table

of system requirements such as machine architecture, OS, libraries, etc. As a result of such flexibility NILE is able to process a wide class of jobs, not just CLEO ones.

The data specification describes which data the job needs. High flexibility has been achieved by defining it as a logical query and/or a set of data locations. A logical query identifies data belonging to the Data Repository and needs to be resolved by the DataLocationManager into a set of data locations. Alternatively, the user can specify a set of data locations identifying the data directly. Each data location is expressed as a sequence of relative URLs, allowing a *data object* to be spread across several *data sources*. Each URL contains the location of the data source, an identifier of the access protocol, and the path name within that data source. A data location can thus identify local files, files on remote servers (*e.g.* ftp, http) as well as data objects inside the Data Repository. Data locations are opaque to the NCS, so it is possible to add transparently any kind of data service. For instance, the current version of NILE uses the file system as a simple Data Repository.

Once the user has defined the job, she submits it to the NCS through the SiteManager and receives a *JobID*. She can then control the job by querying its *status* (*e.g.* `PROCESSING, COMPLETED, FAILED`, etc.), and can kill it and remove it from the system. She can see the decomposition into subjobs, and for each subjob see its parent, status, processor node, subset of data, transaction history, etc. Finally she obtains the results produced by the job.

Since the interaction between the user and NILE is via the SiteManager, which is a CORBA object, the interface to NILE is an instance of the more general problem of interfacing to CORBA objects. In [5] we discuss possible solutions. NILE provides users with both graphical and non-graphical interfaces for defining, submitting, controlling and browsing results. To make such an interface available on the largest class of systems, NILE does not require that any of its components be run on the client side. The user interface will either be a text-based remote terminal or a Java-enabled Internet browser.

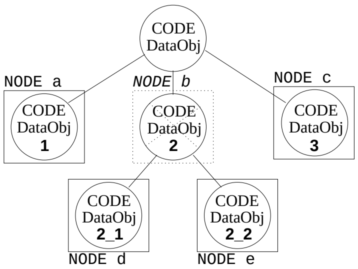

Fig. 2. Job Splitting example: subjob 2 failed and has been re-planned.

NILE minimizes the execution time of a job by splitting it into a set of subjobs, running them in parallel on the available processor nodes. For each job, the Planner object produces an *execution plan* which is a list of *subjobs* specifying for each one the processor node and the set of data to be used. The JobManager starts the subjob on the corresponding processor node. If a subjob fails for any reason, the JobManager will ask the Planner to make a new plan for that subjob, possibly obtaining a new set of subjobs (Figure 2). At the moment the Planner object uses a very simple round-robin policy to assign data subsets and processor nodes to subjobs, but one can easily *plug-in* policies or even new Planners [1] as long as they respect the same interface.

To show NILE's flexibility and low requirements, we demonstrate the system running on a small network of laptops running Linux, but still exploiting fault-tolerant features. Each machine works as a system and as a processor node, meaning that every CORBA object group composing the NCS contains two replicas, except for the JobProcessorFactory object which runs individually on each node (Figure 3). This simple environment allows us to demonstrate that the system resists process failures simulated by killing processes and hardware failures simulated by physically unplugging one node. Conversely, the system seamlessly incorporates the addition of new nodes.

Users can submit, control and retrieve job results from a remote Windows95 machine running an Internet browser but none of NILE's components. We demonstrate the system processing CLEO jobs accessing CLEO data from the local file system. Jobs are split into a set of subjobs running in parallel on each node. We demonstrate how NILE can save computational resources by rescheduling only failed subjobs instead of the whole job.

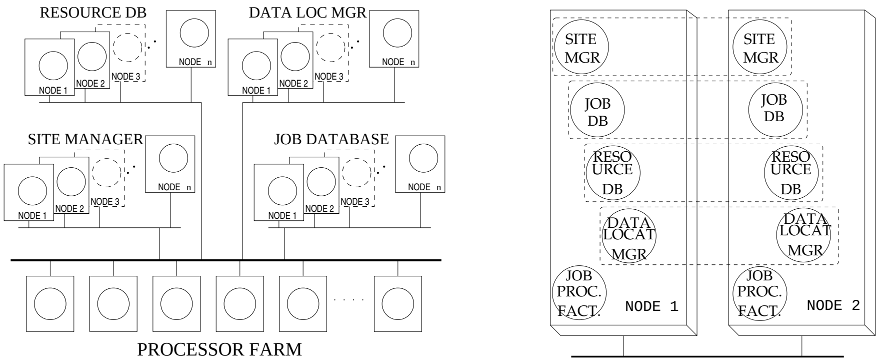

Fig. 3. NCS configuration (a) for the general case, (b) on a 2-node network.

References

[1] A. Amoroso, K. Bhatia, K. M. Marzullo, and A. Ricciardi. Scheduling Resources for Wide-Area, High-Performance Scientific Computing. In *Cluster Computing Conference*, 1997. (to appear).

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

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

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

[5] M. Ogg, F. Previato, and A. Swaminathan. Building User Interfaces for CORBA Objects with Java . In *Computing in High Energy Physics*, 1997. (to appear).

[6] F. Previato, M. Ogg, and A. Ricciardi. Progress with Nile. In *Computing in High Energy Physics*, 1997. (to appear).