Nile Fast-Track: Fault-Tolerant Distributed Processing for CLEO

Daniel Riley [a], Michael Athanas [b,a]

Representing the Nile Collaboration [1]

[a] *Department of Physics, Cornell University, Ithaca, NY, 14853, USA*

[b] *Department of Physics, University of Florida, Gainesville, Florida, 32611, USA*

At CHEP95, we introduced the NILE [3] project, whose goal is to manage distributed computing resources making arbitrarily large and arbitrarily many clusters of commodity computers appear to a user as a seamless uniprocessor environment. NILEFT, a 'fast-track' early prototype of many key design principles of the full NILE project, was released to the CLEO collaboration in Fall of 1996. We report on this working architecture and discuss several aspects of the system. We reflect upon what we have learned and how this has affected the full NILE architecture.

Key words: C++, object-oriented methods, parallelization, large systems

1 Design

The NILE Fast-Track (NILEFT) architecture conforms fairly closely to the general structure of the NILE system architecture [8]. Divergences arise because NILEFT is a prototype system, for which some compromises were made in the system architecture in the interests of expediency. In particular, NILEFT is a local area network system only, designed for a highly homogeneous processing environment; it is tied tightly to the existing CLEO II data and storage formats and to the existing body of CLEO II analysis code.

The *SiteManager* is the main interface between NILEFT and its clients. It is responsible for marshaling the computational and data resources available at

[1] Nile and both authors are funded by the NSF under grant ASC-9318151.

a site to run HEP analysis jobs most effectively. The *Job* represents a user request, and responds to status and change requests from the user. The *Planner* component of the *SiteManager* creates *SubJobs* of the *Job* as computational resources become available to process portions of the *Job*.

The *Provider* is the representative of an individual processor and is responsible for executing *SubJobs* assigned to that processor. The *LocalResourceReporter* collects and monitors resource statistics for an individual processor. The statistics are made available to the *SiteManager* for scheduling and to the system operators for monitoring.

Since NileFT uses existing CLEO data formats and files, the *DataLocationManager* is structured to manage those data. The *DataLocationManager* resolves queries sent to it (by the *SiteManager*) to collections of *FileAtoms*, each of which corresponds to an indivisible unit of data. A single *FileAtom* may correspond to a (possibly replicated) file on local disk or remote *DataServers*, a user data file, a data tape, or the portion of an event list contained in any of the preceding types. The *LocalDataLocationManagers* keep track of the files on each server and maintain an index of the event locations within those files.

The typical CLEO analysis job supplies a function to be called by the analysis framework for each record type in the event stream. NileFT generalizes the physics analysis job to a set of *Actions* for each record type and certain system events (*e.g.*, *Job* and *SubJob* initialization and completion). *Actions* are loaded from user-supplied shared libraries by the central *NileNavigator*. For a typical analysis job, the system-provided *Actions* that summarize log files and sum histograms are attached to the *Job*-completion system event. This is a more flexible system, which has also substantially decreased program development time by reducing program link time, but it is still closely tied to the structure of conventional CLEO analysis jobs.

2 Implementation

One of the significant requirements of the Nile project is to build a *reliable* distributed system. In NileFT, reliability is achieved using the Isis Toolkit [2], which provides reliable, consistent inter-process communication. The key objects of the NileFT architecture, the *SiteManager* and *DataLocationManager*, are replicated using Isis facilities, and the *SiteManager* also uses the Isis logging facility. A thin C++ wrapper layer [4] is used to encapsulate the Isis services in an object framework. The C++ wrappers provide a useful encapsulation of the Isis facilities, but are very tightly bound to the Isis toolkit. Since the wrapper layer IDL does not handle user data types, we pack C++ classes into Isis messages, overloading the `<<` and `>>` C++ operators to provide a convenient and type-safe interface. We found the use of an interface definition language to be a useful discipline that promoted careful design and reduced the number of incompatible interface changes and complete system rebuilds. Full NILE is using Electra [7], a CORBA-compliant ORB supporting replicated objects, which provides better and more portable facilities for type-safe group communication of C++ objects.

An abstract authentication class handles user- and server-to-server authentication. The installation at Cornell uses Kerberos 5 [5] as the underlying authentication system, but other implementations are possible. The traditional Unix solution of reserved ports and `setuid` programs was rejected as too insecure and Unix-specific. We found that the point-to-point bias of GSSAPI [6] made its use in a process-group oriented system awkward.

The NILEFT *Planner* preferentially places subjobs to allow local access to the data, avoiding the additional overhead of network data access [1]. Since the network is highly homogeneous, the *Planner* normally considers only whether data are local and the available compute power.

3 Status

NILEFT is currently in use for production CLEO data analysis on a farm of six Digital Alphas running the Digital UNIX operating system, interconnected by a shared 100-megabit Ethernet. Three smaller Alphas serve as a development and test facility. Since the current farm is fairly small, all six systems are used as both data and compute servers, with the the CLEO ROAR-format data sets striped across thirteen 9GB disks distributed among the six data servers.

Development of NILEFT as a research prototype is ramping down at this time. We are using the lessons learned in building and using NILEFT to build the final NILE system. However, NILEFT is also a production system, so we will continue some low-level development to enhance its utility for production CLEO data analysis.

4 Lessons Learned

As a successful prototype system, NILEFT had a substantial impact on the design of the full NILE system. The basic NILE object architecture was developed via an iterative design process, using the implementation experience gained in building the NILEFT to refine and validate the model.

The prototype was particularly useful in clarifying and simplifying the interface between job management and data management, and the introduction of the *FileAtom* concept played a significant role in simplifying the *SiteManager's* handling of the data. While the requirements are different due to NILEFT's use of existing data formats, NILEFT demonstrated the need for a uniform abstract representation of the data for the *SiteManager* to manipulate, which the NILE architecture generalizes substantially from the *FileAtom* concept.

Building a working production system has also given us early insights to the administrative and authentication requirements for the system. These are simpler in the NILEFT case due to its lack of wide-area support, but the lessons learned will provide the basis for the administrative, authentication, and authorization features of the ultimate NILE system.

Finally, production experience with the NILEFT system has convinced us that reliability and fault-tolerance are desirable features of a distributed processing system, and will become more so as systems become larger and more complex.

References

[1] M. Athanas and D. Riley. The Nile Fast-track Implementation: Fault-tolerant Parallel Processing of Legacy CLEO Data. In *Computing in High Energy Physics '95*, pages 64–68, Rio de Janeiro, September 1995.

[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] O. Hagsand, H. Herzog, K. Birman, and R. Cooper. Object-oriented Reliable Distributed Computing. In *Proc. 2nd International Workshop on Object Orientation in Operating Systems*, 1992.

[5] J. Kohl and C. Neuman. The Kerberos Network Authentication Service (V5). RFC 1510, IETF Network Working Group, September 1993. http://ds.internic.net/rfc/rfc1510.txt.

[6] J. Linn. Generic Security Service Application Program Interface, Version 2. RFC 2078, IETF Network Working Group, January 1997.

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

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