

**THE NILE FAST-TRACK IMPLEMENTATION:  
FAULT-TOLERANT PARALLEL PROCESSING OF LEGACY CLEO DATA**

MICHAEL ATHANAS  
*Department of Physics, University of Florida  
Gainesville, Florida, 12345, USA*  
and  
*Department of Physics, Cornell University  
Ithaca, New York, 14853, USA*

DANIEL RILEY  
*Department of Physics, Cornell University  
Ithaca, New York, 14853, USA*

*Representing the Nile Collaboration*

Nile is a multi-disciplinary project building distributed parallel fault-tolerant computing for high energy physics and related fields. Nile Fast-Track is an early prototype of many key design principles of the full Nile project. Implementation is limited to a local area network, in contrast to the full Nile project which is distributed computing over a wide area network. Object-oriented design techniques are employed to produce a test-bed system which is extremely modular. We report on the Fast-Track project design, its status, and future plans.

## **1 Introduction**

The Nile project<sup>1</sup> (*Distributed Computing and Databases for High Energy Physics*, NSF National Challenge Grant) is building a reliable system for wide-area access to data and computing resources. The project's name reflects its goals of transporting information to distant sites and of making villages of computing resources available to all its users.

The facilities at Cornell University including the Cornell Electron-positron Storage Ring (CESR) and the CLEO detector<sup>2</sup> are undergoing major upgrades. The CESR luminosity will increase by at least a factor of five over this period and upgrades to the CLEO detector will further add to event complexity. These two effects will increase data handling requirements by as much as a factor of ten. The laboratory's current methods of computing will not scale to accommodate this expansion. The CPU, storage, I/O, and networking requirements are expected to increase sharply in the next five years, driven largely by the size of the available data samples.

The system being developed by the Nile project will be used by the CLEO high energy physics collaboration and will be applicable to other projects in science,

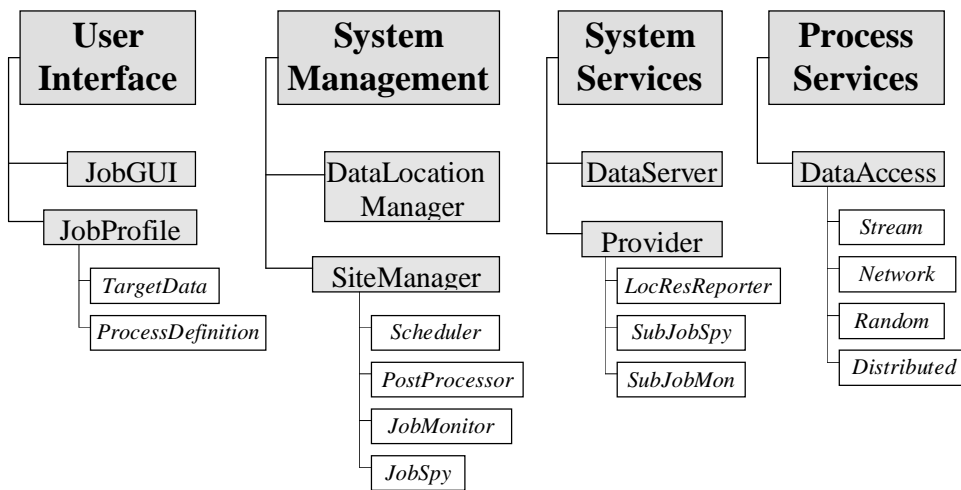
engineering, and industry whose problems can be organized as a sequence of independent events. One aspect of the project agenda is to produce an early prototype system -- the Nile Fast-Track (NileFT). In this paper, we present a description of NileFT and the status of its implementation.

## 2 Nile Fast-Track Design

The Fast-Track system serves as a proof-of-principle and a valuable learning tool for developing the full system. Using an object-oriented design approach makes the system extremely modular. This permits the NileFT system to be used as a test-bed for new ideas for the full Nile system. The NileFT system is more than a pedagogical exercise in system design, as developments in the NileFT system have greatly influenced the full Nile system architecture and design. Furthermore, NileFT will be employed for CLEO production analysis for early 1996. This will allow users to receive immediate benefits from our efforts and provide essential feed-back from the user community.

NileFT is a scaled version of the full Nile system. Self imposed limitations include: an early deadline for a production level system, a system which is compatible with CLEO's existing sequential-based data model, only local area networking, and a homogeneous processing system.

Principle components of the Fast-Track architecture can be grouped in four categories: user interface, system management, system services, process services (see below). What follows is a discussion of the function and interactions of the principle components in the course of the execution of a physics analysis job: the *NileJob*.



### 2.1 NileJob Submission

The user interface contains the *JobProfile* and the *JobGUI*. The *JobProfile* is a persistent object which uniquely describes a particular analysis job. It contains the *TargetData* object - a description of the data to be processed. *TargetData* may describe sets of data files or a list of events. Details such as the locations of the data are

managed by the system - not the user. This allows for possible system performance boosts such as dynamic data location or duplication over multiple nodes. Also within the *JobProfile* is the *ProcessDefinition* - a description of actions to be performed before, during, and after processing the *NileJob*. The *ProcessDefinition* also contains utilized resource information of previous executions of the analysis which may assist job partitioning and scheduling.

The interface between the user and NileFT system is provided by a command line interface and a graphical user interface, *JobGUI*. *JobGUI* not only provides methods for *NileJob* submission but also methods for monitoring the progress of the *NileJob* in the NileFT system and a method for peeking at intermediate results (histograms in progress).

The *NileJob* partitioning, scheduling, reliable execution, monitoring, and post-processing is managed by the *SiteManager*. Once the *NileJob* is submitted to the *SiteManager*, the *TargetData* object is resolved by the *DataLocationManager*. That is, the location of data represented by the *TargetData* object is made part of the object itself. This new resolved *TargetData* object becomes part of the persistent *NileProfile* maintained by the user.

Fault-tolerance is maintained with the use of the Isis<sup>3</sup> toolkit for distributed computing. Isis provides reliable inter-process communications through process groups (a collection of application services on a network) and guarantees consistency. This permits active replication of key management services for the purpose of fault-tolerance and load balancing. For the first phase of the NileFT implementation, a C++ wrapper class Isis/C++<sup>4</sup> is used to implement reliable distributed objects. Isis/C++ provides a useful encapsulation of the Isis process group facilities but unfortunately is tightly bound to the underlying Isis abstraction.

For the purpose of job partitioning and scheduling, the smallest segment of a job, designated as a *FileAtom*, corresponds to a data file. A *NileJob* contains a collection of *FileAtoms*. To attain optimal system job throughput, it may be efficient to partition the *NileJob* for processing across multiple nodes. This partitioning is the duty of the *Scheduler* subcomponent of the *SiteManager*.

To determine the possibility of executing a *FileAtom* at a particular node, the *SiteManager* requires information about: 1) the location(s) of the data files represented by the *FileAtom*, 2) the current and allocated load of the available nodes, 3) the estimated resources required, and 4) any applicable access control limits. Local node information maintained by the *LocResReporter* subcomponent of the *Provider* includes CPU load, network load, disk usage, and memory usage. The *Providers* communicate to the *SiteManager* through distributed object stubs implemented by Isis/C++. The system design is highly modular and permits experimenting and tuning of various scheduling algorithms.

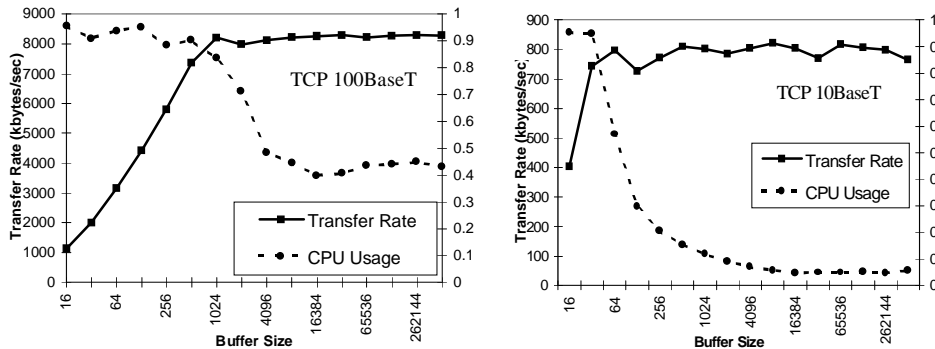
An aggregate of *FileAtoms* scheduled for execution at a particular node is designated a *SubJob*. The actions of the *SubJob* are dictated by the *SiteManager*. So, there is no negotiation of resources by the *SubJob*. The direct creation, execution, and monitoring of the *SubJob* is managed by the *Provider*. The *Provider* also provides a method for the collection and reporting of intermediate results of the *SubJob* through its *SubJobSpy* subcomponent. *SubJob* failure detection and reporting is handled by the *SubJobMon*.

The *SiteManager* provides a *JobMonitor* method for handling *SubJob* failure. Upon failure, a prescribed action is performed such as resubmission, report generation, or failure isolation. Upon the completion of the *NileJob*, results in the form of log files, histogram files, and other processed data are merged into a coherent form by the *PostProcessor* subcomponent of the *SiteManager*.

## 2.2 Data Access

NileFT provides versatile data access methods to the *SubJob*. Good system performance is achieved by optimizing access latencies, data throughput, and CPU loads. Event data can be accessed by one or a combination of: local stream access, network access, or random event access. Local stream access is the simple extraction of logical records (events) from a local disk file. This is typically the fastest and incurs a minimal CPU load due to local optimized hardware.

Network access is the extraction of logical records from a remote source. Performance benchmarks have been explored for both TCP and UDP on 10BaseT and 100BaseT networks. A comparison of the TCP performance benchmarks is illustrated below where the transfer rates in kbytes/second are plotted as a function of buffer size (squares). Also shown is the user CPU load (circles). The optimal data throughput and minimal CPU load is achieved when data buffers are larger than 4kbytes.

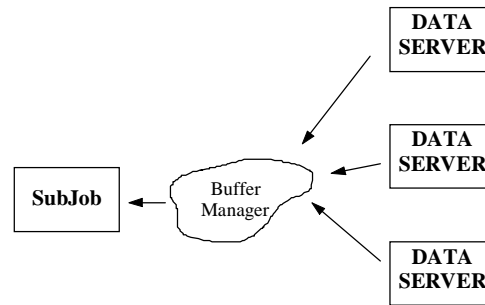


This optimization is achieved in NileFT by limiting inter-node data transmission to bags of logical records with a minimal 4kbyte size.

In later stages of a user's analysis, individual physics events of interest have been identified rendering stream access of the entire data stream wasteful and inefficient. NileFT provides methods of random event access. This eliminates the need for individual skim files and insures against the distribution of invalid data. This is a somewhat formidable task since the sequential-oriented data model of legacy CLEO data must be perpetuated. In this model, compression and format records are embedded in the stream, so logical record order must be maintained.

The implementation of event random access is a two step process. A simple key based database maintains information relating a {run number, event number} key to an offset in the corresponding data file pointing to the logical record of interest. The logical record is extracted directly from the data file and returned to the user. For the first phase of this project, we are using the common *ndbm* database interface<sup>5</sup> to the DEC, Gnu, and Berkeley packages.

The network access method is extended to accommodate multiple network data servers. In this case, a single *SubJob* processes data from multiple servers (see figure below). The multiplexed connection is asynchronously managed by a *BufferManager* object. Signaling and communications in this multi-threaded object are managed by Isis. For a high speed network, this method provides an important low latency transfer mechanism.



### 3. Project Status

Many principle components are completed and ready for system integration. The main pieces of the *JobProfile* are in place while the *JobGUI* remains in the design phase. An early version of the *SiteManager* is complete; however, test and evaluation is dependent upon the as of yet incomplete *Provider*. The data access methods (local stream access, network access, random event access, and distributed data access) are completed and tested.

The design and implementation of the Fast-Track system are on schedule for a production version available to the CLEO collaboration for early '96. Future plans of investigation include evaluating distributed object brokers such as Electra, Orbix+Isis. We will also investigate encapsulation of the legacy CLEO dataset into an object store and/or object-oriented data base.

### Acknowledgments

The Nile collaboration is Alessandro Amoroso, Michael Athanas, Paul Avery, Kenneth Birman, Andrew Calkins, David Cassel, Chris Hawblitzel, Ray Helmke, George Hoffman, Wade Hong, Theodore Johnson, David Kreinick, Keith Marzullo, Aleta Ricciardi, Daniel Riley, Mark Rondinaro, and Eric Rothfus. We wish to acknowledge and thank the National Science Foundation, National Challenge Project (award 9318151).

### References

1. For more information on Nile, see <http://www.nile.utexas.edu/Nile> .
2. For more information on CLEO and CESR, see <http://w4.lns.cornell.edu/>
3. For more information on Isis, see <http://www.isis.com/>
4. O. Hagsand, H. Hersog, K. Birman, R. Cooper "Object-oriented Reliable Distributed Programming" 1992
5. M. Seltzer, O. Yigit, "A New Hashing Package for UNIX", USENIX - Winter '91 - Dallas, TX