# New Capabilities in the HENP Grand Challenge Storage Access System and its Application at RHIC

*L. Bernardo*[1], *B. Gibbard*[2], *D. Malon*[3], *H. Nordberg*[1], *D. Olson*[1], *R. Porter*[2], *A. Shoshani*[1], *A. Sim*[1], *A. Vaniachine*[1], *T. Wenaus*[2], *K. Wu*[1], *D. Zimmerman*[1]

[1]Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2]Brookhaven National Laboratory, Upton, NY, USA
[3]Argonne National Laboratory, Argonne, IL, USA

**Abstract**

The High Energy and Nuclear Physics Data Access Grand Challenge project has developed an optimizing storage access software system that was prototyped at RHIC. It is currently undergoing integration with the STAR experiment in preparation for data taking that starts in mid-2000. The system was exercised during the RHIC Mock Data Challenges as well as tested under conditions designed to characterize scalability, up to 100 simultaneous queries were tested and up to 10 M events across 7 event components were involved in these queries. The system coordinates the staging of "bundles" of files from the HPSS tape system, so that all the needed components of each event are in disk cache when accessed by the application software. The initial prototype implementation interfaced to the Objectivity/DB. In this latest version, it evolved to work with arbitrary files and use CORBA interfaces to the tag database and file catalog services. The interface to the tag database and the MySQL-based file catalog services used by STAR are described.

keywords tag,tape,MySQL,query,analysis,HPSS,STAR,RHIC,CORBA

## Introduction

The High Energy and Nuclear Physics Data Access Grand Challenge project[1] has developed an optimizing storage access software system in collaboration with RHIC[2]. This system is targeted at optimizing the access to data in tertiary storage in the data analysis environment of large-scale high-energy and nuclear physics experiments. The STAR experiment data handling capabilities are currently being developed and integrated with this data access software. The initial implementation and architecture were reported at CHEP'98[3]. In this paper we report on the developments following the RHIC MDC1 in Sept. 1998.

Figure 1 shows the basic Grand Challenge architecture (GCA). Client processes contain the data analysis algorithms. The CORBA interfaces for the file catalog, event identifiers and tag database permit partitioning the experiment-specific details from the general-purpose servers. Each event is considered to be composed of named components. A single component of a single event is contained within a file. Separate components for the same event may reside in different files, or in the same file.

## STACS implementation

STACS is responsible for determining, for each query request, which events and files need to be accessed, to determine the order of files to be cached dynamically so as to maximize their sharing by queries, to request the caching of files from HPSS in tape optimized order, and to determine dynamically which files to keep in the disk cache to maximize file usage. It uses a specialized index, called a bit-sliced index[4] that is used for quick (real-time) estimation of the number of events that qualify for given a query. This index is also used to determine the set of files that have to be cached for each query, and the set of event IDs that these files contain for that query.

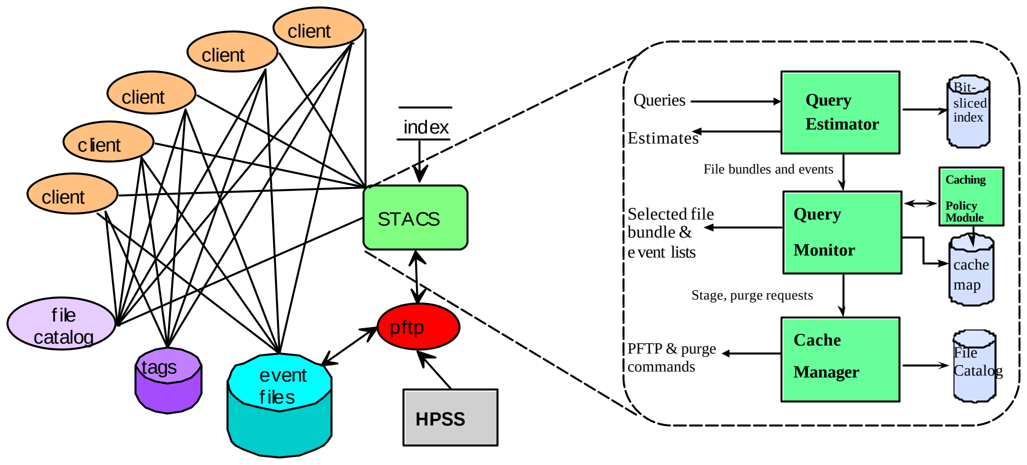

**Figure 1. Illustration of system software architecture with exploded view of Storage Access Coordination System (STACS)**

As shown in Figure 1, the STACS has 3 main components: 1) The Query Estimator (QE), that uses the index to determine what files and what events are needed to satisfy a given range query. 2) The Query Monitor (QM), that keeps track of what queries are executing at any time, what files are cached on behalf of each query, what files are not in use but are still in cache, and what files still need to be cached. The Query Monitor consults an additional module, called the Caching Policy module, 3) The Cache Manager[5], that is responsible for interfacing to the mass storage system (HPSS) to perform all the actions of staging files to and purging files from the disk cache.

The communication[6] between the STACS components and the Client modules are via CORBA interfaces. The Client modules (described in a later section) communicate with STACS by issuing query requests, asking for estimates of the numbers of events, and time to execute the query, issuing an execute request, and getting the information about files when they are cached.

In the extension from single-component to multi-component events, each event is partitioned into several components, such as "tracks", "hits", and "raw". We introduced the term "file bundle" to refer to the ordered set of files, one for each component, that needs to be in cache *at the same time* to process events whose components are in these files. For example, a query requests components c1 and c2. Files F13, F206 are 2 (of many) files for components c1 and c2, respectively. F13 and F206 contain event components in common for events {E7, E36, E102} for a particular query. Then, we refer to the ordered set <F13, F206> as the "file bundle" and to the set {E7, E36, E102} as the set of events associated with that file bundle for that query. The same file can appear in a another file bundle for a given query. Thus, <F13, F301> could be another file bundle for the same query.

The ability to select a subset of the components in the query required an extension to the query language to specify components, for example:

```
SELECT tracks, hits
WHERE glb_trk_tot>0 & glb_trk_tot<10 & n_vert_total<3
```

Now, the Query Monitor has to have all the files of a bundle in cache before it can return to the Client. It needs to check if any of the files of the bundle are already in cache and request caching the remaining files of the bundle from tape. This is achieved by selecting policies on which file should be in cache at any one time. We have developed a methodology that is based on the following 5 policy components: a) assigning file weights and bundle weights, b) servicing queries, c) bundle caching policy, d) file purging policy, and e) pre-fetching policy.

## Scalability testing

Scalability testing was done for the purpose of finding areas where the system can potentially break as the number of events, files, and queries increases. A test dataset was set up for about 10 million events, each partitioned into 7 components, organized into some 4700 files, totaling about 1.6 TB. The tests were performed by launching a large number of queries (about 100) 5 minutes apart. The queries were for 2 to 7 components each, and ranged from 10s to 100s of bundles each. The total amount of time to run such a test to completion was about 18 hours. It was run on an HPSS system shared by other users. The total amount of disk cache available to STACS was 100GB. This was sufficient to hold 200-300 files at a time. The queries had a small overlap of the files, so that the cache was continuously purged to make space for new files. The limit for concurrent PFTP ranged from 4 to 7. At one test, the same 100 queries were launched two more times, concurrent with the first run, so as to have about 250 queries active at the same time. All tests were successful, and the system has been running for up to a week without any failure in our test.

## Client interface

The client interface consists of: a GCA_Resources service, a query object, and an order-optimized iterator. These components are wrapped by an experiment-specific layer of software to provide a view of data access that matches the experiment's analysis or reconstruction framework model. In this section we describe the application-independent client interface.

The GCA_Resources service initializes all client code connections to STACS. Communication with STACS components is CORBA-based, but the GCA_Resources service insulates the client from the need to know about or deal with CORBA-specific constructs. The GCA_Resources service also handles configuration initialization, customization, and serves as a factory for query objects.

The query object is the means by which clients make event selections and specify components of qualifying events. Queries may be constructed from selection strings and predicates: *query = queryFactory->newQuery(select_string, predicate_string)*. An event collection may also serve as a query, with a sequence of event IDs taking the place of the predicate string. The query object provides access to the functionality of the STACS Query Estimator to examine quantities like the number of qualifying events, the number of files that will be delivered, the total number of bytes to be transferred, etc., allowing the client to understand the scope of her query prior to execution. Clients must explicitly invoke the query object's execute() method to set in motion the machinery of data delivery.

Every query has, upon construction, a unique token assigned to it by STACS. The token is also used to initialize the order-optimized iterators; the means by which events are ultimately delivered, one at a time, to client applications.

The GC order-optimized iterator supports both ODMG-style[7] iteration and the interface of an STL forward iterator. Iteration over events should look to a client exactly the same as it would without the GC; only the order of event delivery is different. The following block illustrates one supported style of iteration:

```
// iterator is initialized with this query's token, and a
// pointer to GCA_Resources for access to remote STACS components
// and configuration parameters:
    OrderOptIter iter(query->token(), &GCA_Resources);
    while (iter.not_done() {
// inside the while block, *iter points to the current event
        usercode(*iter);   // process an event
        ++iter;
    }
```

Internally, the GC components know nothing of "events"; they transmit and receive CORBA sequences of opaque structs, whose shapes and definitions are provided by the experiment; an object id as returned by the order-optimized iterator may be {run_number, event_number} for one experiment, an Objectivity/DB ooRef for another. (A cast or conversion operation may therefore be useful, e.g., usercode(make_d_Ref(*iter)) or usercode(make_experiment-specific-Ref(*iter)).

Parallel iteration is possible by initiating multiple iterators in different processes (possibly on different compute nodes) with a single query token.

## STAR file catalog and tag database

The STAR experiment[8] has adopted the MySQL database to keep records of data files and their production history. Records for all files are kept in one database table – fileCatalog. For every file the concept of file "producer" is used (data acquisition run, geant simulation and event reconstruction jobs). A file's primary instance (in HPSS) is stored in the fileCatalog table. Records of other possible instances (copies) of the same files are placed in a separate table. The information specific to each producer is kept in other tables. Every producer has only one record in one of these tables. Since each of producers typically creates many output files, reference to that record in the fileCatalog table is used to resolve this one-to-many relationship. The perl scripts that run the production jobs directly update the database.

During the event reconstruction a set of structures containing selected event information is saved as the tag component of the event. These event tags (about 500 in STAR) are used to preselect events for the time-consuming end-user analysis. The STAR event tags consist of overall event summary tags, daq/online tags, and a set of useful physics tags.

The file cataloguing scripts process this tag component for each file and store them in a separate tag table in the database. Although MySQL is a fast database, queries will be more efficient if there is enough space to keep all the table data in the database server memory. Since tags from one million events will result in a 2 GB table, the memory-resident bit-sliced index in STACS is a more efficient way to achieve high performance. The MySQL tag database is used to provide the tag information to the GC IndexBuilder.

To interface GC software for use in STAR only two components have to be modified: IndexFeeder and fcFileCatalog. At the GC initialization stage the IndexFeeder component reads STAR tags from the MySQL database and forwards them to the IndexBuilder. During the run-time the multi-threaded fcFileCatalog server takes requests for file information from other GC software components, delegates them to the STAR MySQL database server and provides the requested results back.

## References

[1] http://www-rnc.lbl.gov/GC/
[2] http://www.rhic.bnl.gov/
[3] http://www.hep.net/chep98/
[4] "Multidimensional Indexing and Query Coordination for Tertiary Storage Management", A. Shoshani, L. M. Bernardo, H. Nordberg, D. Rotem, and A. Sim, Eleventh International Conference on Scientific and Statistical Database Management (SSDBM'99). (http://gizmo.lbl.gov/~arie/papers/download.papers.html).
[5] "Access Coordination of Tertiary Storage for High Energy Physics Application", L. M. Bernardo, A. Shoshani, A. Sim, H. Nordberg, Eight IEEE Symposium on Mass Storage Systems (MSS 2000). (http://gizmo.lbl.gov/~arie/papers/download.papers.html).
[6] "Storage Access Coordination Using CORBA", A. Sim, H. Nordberg, L. M. Bernardo, A.Shoshani, D. Rotem, 1999 International Symposium on Distributed Objects and Applications (DOA'99). (http://gizmo.lbl.gov/~arie/papers/download.papers.html).
[7] Object Database Management Group, http://www.odmg.org/
[8] http://www.star.bnl.gov/