RECEIVED
JUL 30 1998
OSTI

ANL/HEP/CP-98-38
CONF-980577--

Query Estimation and Order-Optimized Iteration in Very Large Federations

David M. Malon*
for the HENP Grand Challenge Collaboration

Argonne National Laboratory
9700 South Cass Avenue, Building 900
Argonne, IL 60439 USA
malon@anl.gov

4 May 1998

The submitted manuscript has been created by the University of Chicago as Operator of Argonne National Laboratory ("Argonne") under Contract No. W-31-109-ENG-38 with the U.S. Department of Energy. The U.S. Government retains for itself, and others acting on its behalf, a paid-up, nonexclusive, irrevocable worldwide license in said article to reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government.

Abstract

Objectivity federated databases may contain many terabytes of data and span thousands of files. In such in an environment, it is often easy for a user to pose a query that may return an iterator over millions of objects, requiring opening thousands of databases. This presentation describes several technologies developed for such settings:

- a query estimator, which tells the user how many objects satisfy the query, and how many databases will be touched, prior to opening all of those files;
- an order-optimized iterator, which behaves like an ordinary iterator except that elements are returned in an order optimized for efficient access, presorted by the database (and container) in which they reside;
- a parallel implementation of the order-optimized iterator, allowing any number of processes in a parallel or distributed system to iterate over disjoint subcollections of items satisfying the query, partitioned by the database or container in which the items reside.

These technologies have been developed for scientific experiments that will require handling thousands of terabytes of data annually, but they are intended to be applicable in other massive data settings as well. In such environments, significant amounts of data will reside on tertiary storage, accessible via Objectivity's recently-anounced HPSS (High Performance Storage System) interface.

*The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-Eng-38. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

MASTER

DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

DISCLAIMER

Portions of this document may be illegible electronic image products. Images are produced from the best available original document.

When deployed in large-scale physics settings later in 1998, the query estimator will further inform the user of the number of tape mounts required to satisfy the query, and provide rough time estimates for data delivery. The order-optimized iterator will be connected to a cache manager that will prefetch from tape to disk the files needed by the query (known from the query estimation step), and will decide which items to deliver to the user next according to the order in which data become available in the disk cache.

Background

High energy and nuclear physics experiments are on the verge of producing many hundreds (and soon enough, thousands) of terabytes of data annually. Simply storing so much data is a difficult task; making it usable for analysis is even more daunting. In such settings, it is critical that the physicist know whether a query will return 100 or 100,000,000 events before he or she launches it and goes home for the evening, and when an analysis requires a very large number of events, it is essential that the data be delivered as efficiently as possible, even though much of it may necessarily reside on tape.

While the technologies described in this paper are designed for large-scale physics experiments, they are intended to be suitable as well for other applications that

- may touch a very large number of objects in many different databases within a federation;
- can potentially determine which objects are of interest in advance of touching them (e.g., via indexing or via a "tag" database in the parlance of our application);
- are doing *"for each"* analysis: an operation must be performed for each object that satisfies the query, but the order in which the objects are processed is unimportant.

A Note on Tags: In our target applications, the primary objects of interest ("events") may be large, but physicists are often able to decide which events are relevant to their analysis on the basis of the values of a relatively small number (tens or hundreds, but not thousands) of attributes. These attributes collectively constitute an event tag. One strategy under exploration by several physics data projects, including this one, involves keeping such tag objects in a separate database or collection of databases, with each tag containing a reference to (or association with) its corresponding event. Selection queries are issued against these tag databases or indexes built thereon. When a tag satisfies a query, it is the associated event, not the tag itself, that is of interest to the physicist.

Queries and Query Objects

In this architecture, a query may be a selection predicate ("numPions > N"), but it may be a collection of object references (or, equivalently, an iterator over such a collection). Such a collection may be predefined ("Events from Run 217"), or it may be the result of a previous analysis task, in which references to events of interest to the particular physicist have been saved in a personal collection ("myInterestingEvents"). Once selection predicates have been evaluated, these query forms are equivalent—the optimization task of determining the order in which objects should be delivered is the same.

A Query object provides the user interface to query construction, estimation, and execution. It also supports several methods for ascertaining query status.

Query Estimation

The Query Estimator returns a count—approximate or precise—of the number of objects that satisfy a query, and reports as well the number of databases (and containers, if desired) that will be touched. These counts are precise if the query is in fact a collection or iterator supplied by the user; for a query that is a selection predicate, estimates may be approximate or precise at the user's discretion.

Quick versus Full Estimates: For collections of billions of elements, tags consisting of only a few dozen attributes per element yield tag databases with sizes in the hundreds of gigabytes. One consequence is that even when object selection may be accomplished by examining only tag data, this examination may nonetheless be a time-consuming operation. We are experimenting with building approximate indexes that record data values only to within the resolution of a discrete partition of their range. With a bit-sliced, compressed representation, these should be small enough to reside in memory, and will provide upper and lower bounds for the number of objects that satisfy a boolean range query on the indexed attributes. Users will be able to request quick estimates from this in-memory index before deciding whether their queries are sensible in terms of the number of objects that qualify and the time that will be required.

Implementation of "full" estimates is more straightforward. One approach is to maintain a multimap (using, for example the transient STL available with Objectivity/DB 5.0) with the keys being the database ids extracted from object references and the values being object references within that database, sorted by database and container (and page and slot, if desired).

With the help of the Storage Manager, the Query Estimator can determine from this list of databases which of them currently reside on tape and which are available on disk. In future implementations, the Query Estimator will also estimate total data retrieval time based upon this information and the concurrent query load.

Storage Manager

The Storage Manager maintains knowledge of which databases are on disk and which are on tape at any given time. Because queries pass through an execute() interface, the Storage Manager can, in principle, determine all the files that all concurrent queries will need. This allows the Storage Manager to do intelligent prefetching—it attempts to optimize tape access within a query, and among concurrent queries as well.

Order-Optimized Iterator

The Order-Optimized Iterator is the means by which programs iterate over elements in a large collection. It has exactly the same interface as an ordinary iterator such as the template class `d_Iterator<T>` in ODMG 2.0. Only its implementation is different: it returns elements in a retrieval order that has been optimized to account for physical clustering and caching on disk. Its intended use is as the mechanism by which physics codes iterate over events satisfying selection queries when those events must be retrieved from tertiary storage.

In its implementation, the iterator talks to a Storage Manager, and gets a (sub)list of references to objects that both satisfy the query and reside in a single disk-cached database. When the user invokes, for example, iter.next(currentRef), a reference is returned from this sublist. When the sublist is exhausted, iter.next(...) causes the iterator to get a new sublist from the Storage Manager (which has, presumably, prefetched another database if necessary from tertiary storage).

Note that order optimization occurs on two levels: first, object references are delivered having been presorted by database (and container, and so on); second, *which* database's object references are delivered next depends upon what is available in the disk cache. The fact that the architecture knows the complete list of database ids that the user requires means that the Storage Manager can attempt to prefetch databases to be served to the user later (via the iterator) while the application program is dealing with its current list of object references.

An Example

```
//Step 1:
   Query* myQuery = QueryFactory::newQuery("Vanilla Query",
                                            "attr1 > a AND attr2 <= b");

/****************************************
           begin optional stuff

//Step 2 (optional):
   myQuery->doQuickEstimate();
```

```
   if (myQuery->minNumEvents() > tooMany) // even lower bound is too high?
      doSomethingElse();

//Step 3 (optional):
   myQuery->doFullEstimate();
   if ((myQuery->numEvents() > tooMany) || (myQuery->numSeconds() > tooLong))
      doSomethingElse();

          end optional stuff
****************************************/

//Step 4.
   myQuery->execute();

//Step 5.
   OrderOptIterator* myIter = OOIteratorFactory::create(myQuery->token());

   d_Ref<Event> thisEvent;  // or an appropriate object reference type

// Step 6.
   while (myIter->next(thisEvent)) {
//    ... physics analysis goes here
   }
```

Parallelism

A unique Query Token is assigned to each executing query, and is used in the construction of the Order-Optimized Iterator. (See Step 5 in the example above.) When the iterator asks for the next sublist of object references from the Storage Manager, it passes this token to identify *which* query's next sublist should be returned–essential in a multiquery environment.

This approach makes parallelization straightforward: one simply broadcasts the query token to all collaborating processes, and initializes their iterators with it. All processes in the parallel application will thus ask for sublists corresponding to the same query. The Storage Manager serializes these requests, ensuring that each sublist is delivered to exactly one of the processes. The result is a disjoint partition of the query workload among parallel processes, with each process dealing with a different subset of the databases containing objects of interest.

Status and Future Work

While a prototype C++ implementation of the work described herein has been built and tested, much of the real work of building a system remains. The bit-sliced index

technology and the cache manager with HPSS connections both exist, but have not as of this writing been integrated into the order-optimized iteration environment. Interfaces are certain to change; we are aware, for example, of both the Object Data Management Group's Query object definition in its C++ binding, and the Object Management Group's Query Services Specification. All of our components will need to operate in a multi-ORB CORBA environment. Preliminary integration of CORBA versions of the architectural components will take place in May 1998. The litmus test for this architecture will occur in August 1998 as part of the Relativistic Heavy Ion Collider's first Mock Data Challenge. We have yet to connect this architecture to more than a few gigabytes of physics data, and we know we have much to learn about indexing, prefetching, data clustering (and reclustering!) and cache management. We're looking forward to it.

Acknowledgments

This work is one part of a larger Grand Challenge Application in High Energy and Nuclear Physics, funded by the U.S. Department of Energy. The design is the work of many people, and its ultimate implementation will be the work of many more. The project spokesman is Douglas Olson at the Lawrence Berkeley National Laboratory (DLOlson@lbl.gov). Please feel free to contact him or the author for additional information.

References

[1] Object Management Group, *CORBA/IIOP 2.2* (OMG, March 1998).

[2] R.G.G. Cattell et al, *The Object Database Standard: ODMG-2.0* (Morgan Kaufmann, San Francisco, 1997).

[3] The HENP Grand Challenge Application Website, http://www-rnc.lbl.gov/GC/default.htm