Building a Large Location Table to Find Replicas of Physics Objects

Koen Holtman, Heinz Stockinger

CERN – EP division, CH-1211 Geneva 23, Switzerland

Abstract

The problem of building a large location table for physics objects occurs within a number of planned physics data management systems, like those that control reclustering and wide-area replication. To satisfy their efficiency goals, these systems have to make local or remote replicas of individual physics objects, which contain raw or reconstructed data for a single event, rather than replicas of large run or ntuple files. This replication implies the use of a table to resolve the logical, location independent object descriptor into a physical location where an object replica can be found. For modern physics experiments the table needs to scale to at least some 10^{10} objects. We argue that such a table can be efficiently implemented by limiting the freedom of lookup operations, and by exploiting some specific properties of the physics data model. One specific viable implementation is discussed.

Keywords: Object location table, object-oriented databases, object clustering, object re-clustering, data replication

This paper is a shortened version of a long CHEP'2000 conference paper that will be published after the conference.

1 Introduction

The CMS experiment plans to store 1 Petabyte of data per year in an object database system, from 2005 on,which will consist of 10^9 events. A single event will have at least some 10 objects associated with it, each individual object holding some raw or reconstructed data for the event. This means that at least 10^{10} objects are created annually. Each of these objects can potentially be replicated to a Regional Centre. Multiple replicas of an object can also exist at a local site when objects are reclustered or staged from tape to disk. In both of these cases, an object location table (OLT) is required to resolve the logical, location independent object descriptor into a physical location where an object replica can be found.

In an object-oriented database management system, each single object in the database is identified uniquely by a so called object identifier (OID) which can either be physical or logical. Logical OIDs are preferred when data movement comes into play[1]: since there are no permanent addresses stored in the OIDs, the objects can be moved freely. Additionally, object replication and access to different versions can be supported effectively. The price for this flexibility is that the OID has to be mapped in a lookup step to the permanent address of the object. The object location table presented in this paper uses logical OIDs.

The aim of our OLT is to enable and optimise access to a set of physics objects which may be partially replicated. The OLT not only provides lookup operations, but also determines which replica to return in any lookup operation in order to minimise the access cost.

Our OLT is a generic system which can be used for all systems with replication: not only in

systems with wide area replicated data, but also for mass storage systems which stage (replicate) from tape to disk, and for systems that replicate objects while reclustering them, like [3] and [4].

Our OLT system requires that, at least on the small scale, events are iterated over in time order. This implies that individual objects are always accessed, and thus looked up, in the time order of their corresponding events. By internally storing index information about the location of objects in the same time order, the OLT can process many subsequent object lookup operations for every single index page read from disk.

2 The logical OID and the job model

In our system, a logical OID is a unique identifier of a particular object that holds physics data for a particular event. The OID has three fields, as follows.

(*chunk ID***,** *event sequence number***,** *type/version ID* **)**

The first two fields, chunk ID (a chunk consists of many events) and event sequence number, uniquely identify the event to which the object belongs. The last field, type/version ID, uniquely identifies the object among the different objects that exist, and can be created, for the event.

For the purpose of using our OLT system, a physics job is defined as follows. Every job specifies:

1. An event set S of the events which must be visited. We assume that each event in S is represented by its (*chunk ID, event sequence number*) pair.
2. A *job function*, which is a piece of executable C++ code (typically a loadable shared library), to be run for each event.
3. For every different kind of object to be read by the job, the type/version ID of that object.
4. (Optional) If, for some type/version ID t, the corresponding object is not needed for every event in S, then a smaller event set $S_t \subset S$ can be specified, with the understanding that the job function will only need the objects t for the events in S_t.

To execute the job, the system first computes the set C of all chunks that contain one or more objects identified in S. For every chunk in C, the system runs a *sub-job* over the requested events in this chunk. This iteration is always done in event time order, so in the order of increasing event sequence number.

3 The Object Location Table

Figure 1 depicts the physical representation of the whole object location table in the two top quadrants. In the two bottom quadrants the physical objects and their distribution to different locations is shown. In this example, three different *collections* (a set of replicated objects is called a collection) exist at two geographically distributed sites named CERN and RC1. The site CERN holds two collections, one on tape and the other one on disk. While the collection on tape contains all possible objects of the corresponding chunk and type, the collection on disk only stores a subset of the objects. Another subset is stored in the collection on disk in RC1.

We assume a distributed system with interfaces to the data store at CERN and RC1. Hence, a local copy (replica) of the object location table is available at both sites (see top quadrants). We can distinguish between five different index levels where the first four ones are replicated to all sites where objects can be looked up and the lowest level is only stored where the physical objects are located.

The first two index levels represent the type/version ID and the chunk ID of the corresponding object. The Top Level Collection Index holds the list of collections for a single chunk.

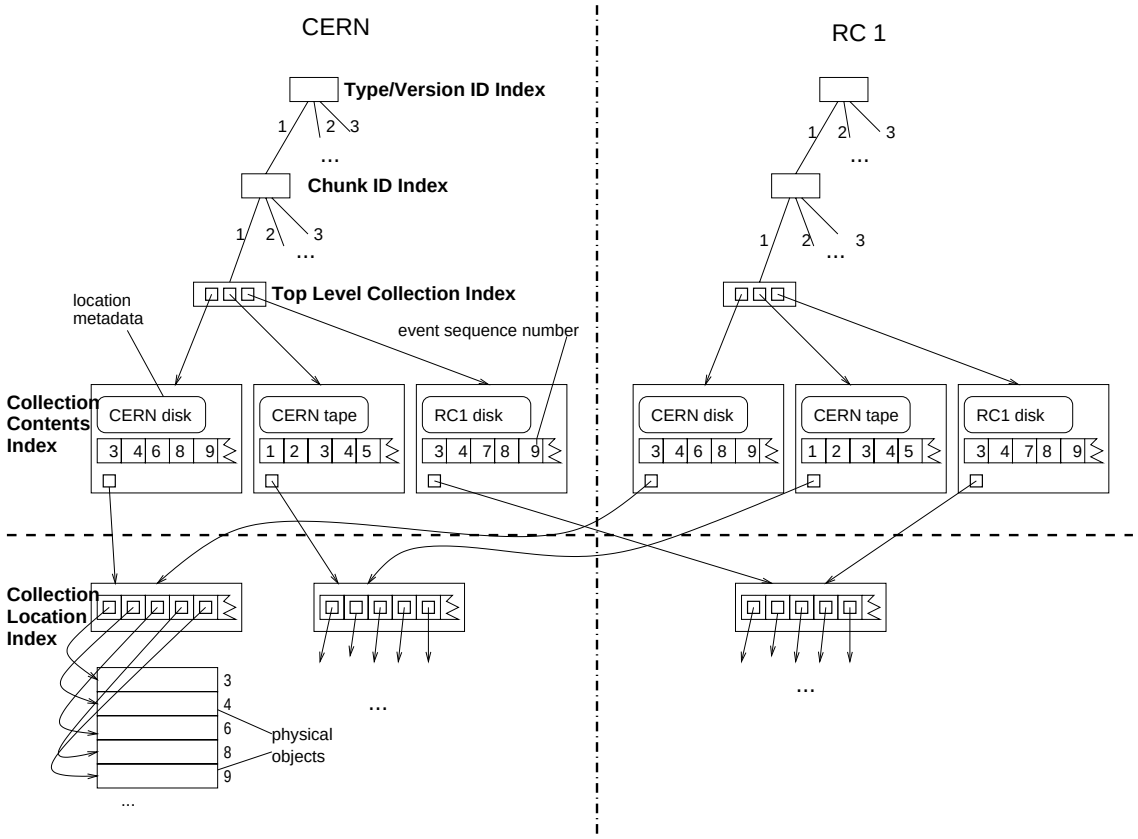

Figure 1: Physical representation of the whole object location table, with different levels of indexes. Arrows indicate location dependent, physical pointers.

4 An example of sub-job execution

Our OLT automatically determines which replicated objects to access in order to minimise the access cost. This optimisation process happens at the level of the execution of a single sub-job. As an example to show the optimisation issues in executing a sub-job, we take a sub-job that should access the objects with some type/version ID 80 for four events in a set S_5 in chunk 5. As the chunk ID and type/version ID are fixed for the whole sub-job, we can represent the identity of the objects needed with the event sequence numbers in their logical OIDs only. Say we need the set

$O = \{2, 4, 5, 8\}$

Say that there are four collections relevant for the job. The contents of these collections can again be represented as sets of event sequence numbers.

collection 1 = { 1,2,3,4,5,6,7,8,9,10 } located at CERN on tape
collection 2 = { 2, 5, 8 } located at CERN on disk
collection 3 = { 2, 4, 8, 10 } located at CERN on disk
collection 4 = { 2, 4, 8, 10 } located at RC1 on disk

Say the (sub)job runs at RC1. We can now ask the question: which replicas should be read by the sub-job to obtain the best overall performance? Looking at the information above, in this case the answer is obvious:

Read object 2 from collection 4

Read object 4 from collection 4
Read object 5 from collection 2
Read object 8 from collection 4

Our OLT system automatically calculates the best answer at the start of the sub-job by comparing the set O of objects needed to the contents of all relevant collections. To make these comparisons, only the indexing structures shown in a topmost quadrant of figure 1 are needed. The replicated objects themselves are only touched when the sub-job starts its iteration over the events.

In calculating the optimum, a *cost function* is used for each collection in question, to calculate the costs of reading a specific number of objects from that collection. Collections at different locations (CERN or RC1), or with different access characteristics (disk or tape), have different cost functions. The optimisation process will be fully described in the long version of this paper.

5 Conclusions

The use of replication implies the use of a table to resolve the logical, location independent object descriptor into a physical location where an object replica can be found. For modern physics experiments the table needs to scale to at least some 10^{10} objects. Our OLT obtains its scalability by limiting the randomness of the lookup operations that are performed on the table. There is a 'locality of access' on the lookup table, and this ensures that we can process many subsequent object lookup operations for every single index page read from disk. Two distinct methods are used to get this locality of access. First, locality of access is obtained by encoding chunk IDs and type/version IDs directly into the OID, and by running sub-jobs over single chunks. Second, inside the sub-jobs, there is a locality of access in lookup operations at the 'Collection Contents Index' (CCI) level in figure 1 is ensured by requiring event iteration to happen in time order, in the same order at which the event sequence numbers are ordered inside the CCI representation. Apart from scalable lookup operations, the OLT also supports scalable update operations: an insert operation for a new (replica of a) set of physics objects can be implemented by inserting some new CCIs into the tree structure of figure 1, CCIs initialised to point to the new objects.

Large parts of the OLT system have been implemented in prototypes. The prototyping results are discussed in the long version of this paper which will be published after the conference.

References

1 Eickler, Andre, Carsten Gerlhof, Donald Kossmann. A Performance Evaluation of OID Mapping Techniques, Proc. of the Conf. on Very Large Data Bases (VLDB), Zurich, Switzerland, September 1995.

2 Holtman, Koen, Julian Bunn, Scalability to Hundreds of Clients in HEP Object Databases, Proc. of CHEP '98, Chicago, USA 1998.

3 K. Holtman, P. van der Stok, I. Willers. Automatic Reclustering of Objects in Very Large Databases for High Energy Physics, Proc. of IDEAS '98, Cardiff, UK, p. 132-140, IEEE 1998.

4 Holtman, Koen, Peter van der Stok, Ian Willers. A Cache Filtering Optimisation for Queries to Massive Datasets on Tertiary Storage. Proc. of DOLAP'99, Kansas City, USA, November 6, 1999.