Atlas event data model optimization studies based on the use of segmented VArray in Objectivity/DB.

S. Rolli[1], *A. Salvadori*[2],*A.C. Schaffer*[2,3],*M. Schaller*[2,4]

[1] Tufts University, Medford, MA, USA
[2] CERN, Geneva,Switzerland
[3] Laboratoire de l'Accelerateur Lineaire (LAL), Inst.National de Physique Nucleaire et de Particules Univ.de Paris Sud, Universite de Paris-Sud (ParisXI), Paris, France
[4] Innsbruck University, Austria

Abstract

The current raw data event model for ATLAS is benchmarked for writing and reading performances as well as an extension of it based on the the use of segmented VArray (SegVArray). SegVArray is a multilevel variable-size array with the same interface as the Objectivity/DB VArray class, but containing ooVArray of SVArraySegments, each of them containing an ooVArray of objects that are the elements of the SegVArray.

The advantages compared to the ooVArray class might be manifold as illustrated through the performance benchmarks performed on a toy model as well as the ATLAS raw data event model.

Keywords: database, Objectivity/DB, event data model

1 Introduction

The key software elements which directly concern the computing model which will satisfy the major storage needs of the ATLAS experiment, are the management and the storage of the data, where the two central components for the data storage are: 1) an Object Database Management System (ODBMS) and 2) a Mass Storage System (MSS). RD45[1] investigated the use of commercial ODBMS and MSS in HEP and at the moment the candidate solution is represented by the use of Objectivity/DB[2] coupled to HPSS[3].
The various objects of an event are clustered together to define different event object groups used in the offline analysis: raw data, ESD, AOD, tag[1] . In this paper we will be concerned with raw data. The actual ATLAS raw data event model organizes each channel response as *digits* which are *contained* by an object representing the detector element which produced it, e.g. silicon wafer. Each detector elements provides an *Identifier*, to allow for both identification and data selection. The most natural way to store the large number of digits is through the use of arrays.
Objectivity/DB provides several persistent array data structures:

1) *Fixed-size array*. The size of this array is specified in the DDL file and cannot be changed at run time.

2) *variable-size array (VArray)*. The size of the VArray can be changed at runtime. The contents of the variable-size array is guaranteed to be allocated in adjacent fashion in the persistent storage.

3) `vector<T>`. Support for a persistent version of C++ STL containers has been recently added.

[1]ESD also known as Event Summary Data, contain enough information to allow for reconstruction of the event, AOD, or Analysis Object Data, contain the parts most needed for the analysis and TAG is a very compact information, allowing for fast access of the data

None of them is totally adequate to be used as a digits container. Fixed-size arrays clearly disqualify, the number of digits varying event by event. VArrays have some constrains that make their usage as digits container problematic. Adjacent allocation of storage offers indeed faster element access than non adjacent allocation but it might lead to expensive resize operations. If the size of the array is extended another larger adjacent block of storage has to be allocated and the content of the original array has to be copied to the new place. The effective limit in the size of the VArray comes from the fact that the entire VArray must be read in memory before an operation can be performed on it. This is useful if all the elements are accessed, but it's not always the optimal solution. If only one or a couple of elements are accessed by the application this additional constraint represent a severe performance drawback. The last array class is a persistent version of `vector<T>` and has still sever performance problems.

In this paper we describe an array structure, the multi-level array, that might overcome the previous mentioned problems. The multi-level array data structure splits a large array up into smaller fixed-size arrays called segments. A variable size array refers to the segments. If an entry of the multi-level array is accessed only the variable-size array referring to the segments and the segment containing the entry have to be brought into memory. The size of the segment is a critical

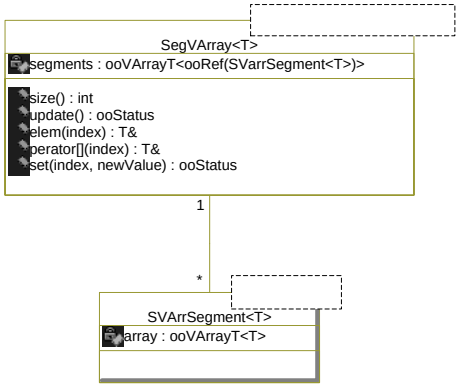

Figure 1: The segmented VArray

parameter. Since the page size is the unit of data transferred between main memory and secondary storage, there is no advantage to make the segment smaller than the page size. On the other hand, if the segment size is larger than the page size, an additional space overhead in the size of half page occurs. This overhead is acceptable only if the segment size is large compared to the size of half page, say if the segment is larger than four pages. Objectivity/DB has developed an unsupported multi-array raArray and the class SegVArray has been written as a modification of raArray. The class diagram is shown in figure 1 . SegVArray contains an ooVArray to the SegVArraySegments. Each SVArraySegment contains an ooVArray containing the objects that are elements of the SegVArray.

The advantages compared to the ooVArray class are:

- if only some objects of the array are read in, only the segments are read that contain these objects, therefore the SegVArray can be used for very large arrays as well.

- If a segment of a SegVArray fits onto one page, the SegVArray can span several pages without the overhead half page for large objects.

The disadvantages are of course the bad performances of the SegVArray for the case that most of the elements are read from the SegVArray. This is particularly true if operator[] is used to access the information. Reading performance tests have been conducted on several of the models outlined below, using the operator[] to access all the elments of the SegVArray and the results are extremely CPU bounded. On the other hand, the use of an iterator can improve performances dramatically, even in the case of reading all the elements of the SegVArray. Indeed a SegVArrayIterator class has been designed and implemented with the characteristic of a STL random iterator and the results of using it to read all the elements are very good.

In the following sections we present some results relative to reading and writing benchmarks using the original ATLAS event data model, as well as modification of it, based on the use of SegVArray.

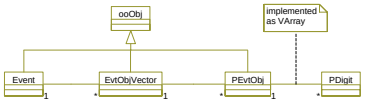

Figure 2: The original ATLAS raw data event model

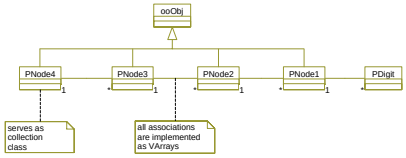

Figure 3: The generic event model modeled after the original ATLAS event model

Table I: Typical fan-out for the ATLAS raw data model components

detector	EvtObjVector	EvtObj		Digit	
	fan-out	size	fan-out	size	average VArray size (byte)
Si	1500	23	7	12	85
TRT	600	23	13	8	100
Calo	40	23	200	8	1600

2 The original ATLAS raw data model

The original raw data event model is shown in figure 2. The digits are contained in the detector element object (PEvtObj), through VArrays. PEvtObj are contained in PEvtObjVectors, through a 1-to-many bidirectional association. There are about 10 EvtObjVector objects for each Event object, one for each detector system.

A more generic event model, modeled after the original one, has been benchmarked, see figure 3. All the associations are implemented through the use of VArray. PNode3 corresponds to the Event class, PNode2 to EvtObjVector and PNode1 to PEvtObj. The new Digit is of a more generic form. Pnode4 corresponds to a generic event collection. The fan-out shown in TableII were used. The page size is 8 KB. Server and client were hosted on two independent machines:

- 2 * Sun UltraSPARC II 399MHz; 512MB memory with Hitachi RAID unit with typical data transfer of 25MB/sec (server and local/remote client);

- SUN UltraSPARC II 270MHz, 192MB memory, Elite23 drives - 9MB/sec (remote server).

The network speed between the two machines have been measured to be about 7.4 MB/sec. The benchmarks results are reported in TableIII: the rates are derived by dividing the the size of the database by the total time taken by the application to access it to write or read (as given by the Unix command time). We would like to notice that in the results reported in Table III the user time is of order 60% to 80% of the real time. The database size is 197.8 MB, corresponding roughly to 0.66MB per PNode3.

Table II: Fan-out for the model of Fig. 3 and the one of Fig. 4

	PNode4	**PNode3**	**PNode2**	**PNode1**
fan-out	**300**	**10**	**100**	**50**

Table III: Original event model benchmarks results

	local	**AMS**
writing	**5.7MB/s**	**3.7MB/s**
reading	**10.1MB/s**	**2.8MB/s**

3 The modified event raw data model

A modified version of the event model is shown in figure 4. The digits have been clustered together and moved to PNode2, in order to obtain larger VArray sizes. The fan out for the nodes is the same as for the original model (TableII). Each digit consists of 3 integer numbers, so the total size in bytes of the VArray is given by : $50 \times 3 \times 4 \times 100 = 60000$. The page size chosen is of 8 KB. The database size is 198 MB, corresponding roughly to 0.66MB per PNode3. The results are summarized in tableIV: once again the rates are derived by dividing the the size of the database by the total time taken by the application to access it to write or read it (as given by the Unix command time). Even in these results the user time is of order 60% to 80% of the real time.
We would like to remind that a VArray of 60000 bytes occupies 9 8KB pages, leading to an overhead of 22%. Would have we chosen a page size of 16KB, the pages occupied would have been 5, leading to an overhead of 36%.

Table IV: Modified event model benchmarks results

	local	**AMS**
writing	**10.8MB/s**	**6.7MB/s**
reading	**13.88MB/s**	**4.6MB/s**

We notice that the half-page overhead for large objects is a sever constraint for VArrays that are larger than the page size, but occupies only several pages. On the other hand very large objects are read very fast (see also [4], Table 1).

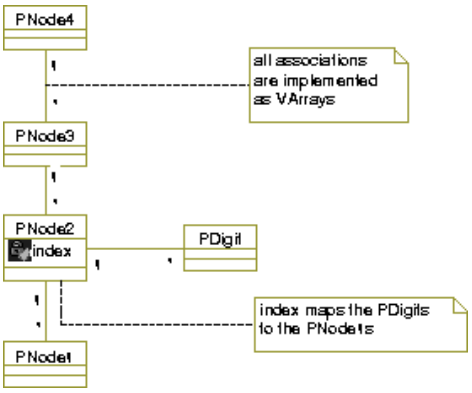

Figure 4: The modified Event Model

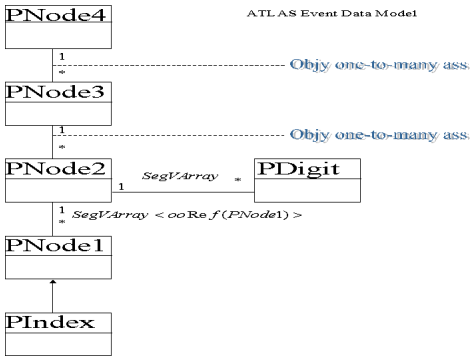

Figure 5: The new Atlas Event Model

Table V: Fan-out for the SegVArray-based event model

PNode3	PNode2	PNode1	PDigit
100 (25)	10	5*I; I:1-10	+=5*I; I:1-10

4 The new ATLAS raw data model

In the new event model the digits have been been clustered together as in the modified model and moved to PNode2, this time through a SegVArray. As it is shown in figure 5, PNode2 contains also a SegVArray<ooRef(PNode1)> where actually the class PIndex is stored, such that it is possible to have a map of the digits indexes corresponding to the various detector elements. The associations between PNode3 and Pnode4 and Pnode3 and Pnode2 have been implemented using Objectivity/DB one-to-many bidirectional associations. A more realistic fan out for the nodes is reported in table V, such that now the size of the data stored in the SegVArrays of Digits varies between 900 bytes and 76500 bytes. Different databases were created corresponding to a different number of pages per segment. The page size is always 8KB.

The results for reading the digits, locally or via AMS using an iterator, are reported in table VI and VII: in the case all the digits are read, the iterator over the digits is used, while in the case one digit is read per PNode1, the iterator over SegVArray<ooRef(PNode1)> is used, to select the corresponding digit. This might imply an effective larger volume of data read, in respect to not using the iterator over SegVArray<ooRef(PNode1)>, but the fact that the size of the digits array is variable doesn't leave us any other choice. The rates are derived by dividing the the size of the

Table VI: Benchmarks results on the new ATLAS event data model

Pages per segment	**1**		**2**	
	Local	**AMS**	**Local**	**AMS**
reading(1) (MB/s)	**8.52**	**3.27**	**9.0**	**0.95**
reading(2) (MB/s)	**9.49**	**4.60**	**6.21**	**2.43**
(1) 1 digit per node1; (2) all the digits; Database size PNode4	552 MB 50 (25 PNode3 each)		540 9 (100 PNode3 each)	

Table VII: Benchmarks results on the new ATLAS event data model

Pages per segment	**4**		**8**	
	Local	**AMS**	**Local**	**AMS**
reading(1) (MB/s)	**12.48**	**3.72**	**10.68**	**3.29**
reading(2) (MB/s)	**11.10**	**5.71**	**7.62**	**5.11**
(1) 1 digit per node1; (2) all the digits; Database size PNode4	565 MB 10 (100 PNode3 each)		543 MB 11 (100 PNode3 each)	

database by the total time taken by the application to access it to write or read it (as given by the Unix command time). The user time is of order 15% to 30% of the real time.
Performances improve when only a few elements are accessed by a local application, except when the number of pages per segment is 1 (Table VI, column 1). In this case in fact accessing just one digit per detector element, or all of the digits is equivalent from the point of view of the pages brought into memory. The size of an event (PNode3) is roughly 0.50MB. More detailed studies of storage overhead will be performed in the future.

5 Conclusions

In this paper we have reported on preliminary studies on using variable size array (Objectivyt/DB VArray) and multi-level variable size arrays (Objectivity/DB SegVArray) to model the ATLAS raw data event model. We have presented some reading rates in the case all the digits are accessed by the application or only few of them. Performances improve especially when only a few elements are accessed by a local application. When the access is done remotely accessing only few elements is more time consuming, due also to the internal page caching mechanism in Objectivity/DB using AMS.
We notice that we are still well within an estimated budget of 1MB/sec which might be sufficient to reconstruct an ATLAS event in a standard way. Further studies are necessary in order to optimize the use of these class of arrays, and the event model itself. To this extent we plan to use the large volume of ATLAS simulated raw data.

References

1 RD45 Collaboration: Using an Object Database and Mass Storage System for Physics Analysis. Technical Report CERN/LHCC 97-9, CERN 1997.

2 `http://www.objectivity.com`.

3 `http://www.sdsc.edu/hpss/hpss.html`.

4 W.D.Dagenhart, K. Karr, S. Rolli and K. Sliwa, CDF/DOC/COMP_UPG/Public/4522 available at:`http://130.64.8.4/CDF-OBJY/cdf4522_objy_status_rep.ps`.