Persistent Pointers in the STAR Micro-DST

V. Perevoztchikov
Brookhaven National Laboratory,USA

STAR

What is the PROBLEM?

The problem is in supporting: creating,updating of complex DST, Mini-DST,Micro-DSTs.

- Full experimental DST contains a lot of non physical information. It is not good for analysis. There is no disk space to keep it;
- Mini-DST contains only physical information. But it is still too much for particular physics team. Disk space is problem too;
- Set of micro-DSTs. Each one is good for one team. But a lot of duplicated information. Hard to maintain;
- The evident solution is to read simultaneously a subset of needed micro-DSTs. In this case we can avoid duplication of information.

The last solution is good, but there is a difficulty:

- relationship between objects from the different Micro-DST (files).

Relationship. Typical Example

Track has relationship with Hits, from which it was.
made. In C++ it is natural to use pointers for it.
When Hits are in one file and tracks in another, it is not too trivial.
Indices as a solution?

- We choose C++ as OO tool. Why we should refuse of it's power and return to Fortran era approach? Then it is more logical to return to Fortran. It was not bad language.
- If our Hits are from different collections, one index is not enough. We need second index for collection and must create a collection of collections for indexing them. But these hits collections could be also in different files.
- If we use indexing, we forbid rearranging of collections, shrinking them, etc…

So, we decided to implement persistent pointers.

Object Identifier.

Object Identifier is the possible solution. The

object table[id] == pointer,

allows conversion from id to pointer.

What kind of identifier?

Well known *Universally Unique Identifier* (UUIDGEN utility) is good for it, but too big (32 bytes). It is easy to avoid by using UUID only for the top level objects of the record. All subordinated objects could have 4 bytes EUID (*Event Unique Identifier*).

So each object is world wide uniquely labeled by UUID.EUID.

In ROOT TObject has fUniqueID member, which could be used for that, without additional penalty.

Little bit of History.

Relationship of objects from different files is not a new problem:

- Professional Data Bases , like Oracle, usually support it;
- Objectivity, as far as I know, not;

- ZEBRA, where links functionally are very close to pointers, no support also;
- ADAMO, probably(?) supports it.

- DSPACK (NA49) supports it, indexing pointers during I/O;

- FARFALLA, old C++ I/O system, no support.

Main Features of STAR Persistent Pointer Implementation

- Top object of structure inherited from special class (StXRefMain).It has UUID and contains list sub-structures (Branches).
- Top object of sub-structure inherited from special class (StXRef). It has the same UUID as the main object. Sub-structures could be written in different files or different records of one file;
- Each object could be:
  - Ordinary object. Can not be seen from the other file or record;
  - Pointable object. Must be inherited from special class (StObject). It has 4 bytes ID. Could be seen from the other file;
- The complexity of structure is based on two types of containers:
  - Structural container, which is owner of objects and delete them if deleted;
  - Reference container. Keeps only reference pointers to objects. Never deletes them.

Main Feature continue.

- Each object belongs to only one structural container;
- Each object could belong to several reference containers;
- Containers can contain other containers;

Such structure could be very complicated and it is complicated in STAR case.

Writing:

- Structural container writes the objects and EUID;
- Reference container writes only EUID of objects;

Reading:

- Structural container reads the objects and fills temporary table[EUID]=pointer;
- Reference container reads EUID and gets pointer=table[EUID];

Updating:

- During reading user can add sub-structure and write it. It could be back pointers to old sub-structures in it.

Conclusions

- The ***Persistent Pointers utility*** was developed in STAR on the base of ROOT I/O;
- It is working and will be tested in upcoming production and reproduction in STAR;
- Physicist can read only needed several Micro-DSTs simultaneously, like they were written together.
- Each physicist can add his own Micro-DST with additional information, with no rewriting the old data.
- We can keep only commonly used parts on disks, keeping all the rest on tapes;
- There is no penalty in speed. It is even slightly (5%) faster than standard ROOT I/O.
- It is not big. About 1000 lines of code.