Subject: First results of TPC code profiling
Posted by Felix Boehmer on Thu, 30 Dec 2010 15:01:21 GMT
View Forum Message <> Reply to Message

Dear colleagues,

following up several complaints about crashes inside the TPC code (e.g. the PndTpcElectronicsTask) I started looking into what the code is doing more closely.

Let me summarize my observations so far:

I still am not able to reproduce any crashes
The memory footprint development over time is consistent with the step-wise behavior I explained in my last post and in the last evo meeting (TClonesArray's memory management is most likely the reason).
At single points in time memory consumption shows 'catastrophic' behavior, blowing up to several GB
This seems *not* to be directly linked to the event size - I saw events with 200k PndTpcSignals that seemed to cause problems, but I also saw events with 500k Signals without any increase of memory load
I was not able to isolate a single event leading to this so far

Please find attached a printout of a heap analysis I performed running the TPC digitization macro. Unfortunately the graph is a bit difficult to read, but I am sure you will find your way around the plot . Listed for every consumer (box) is the total amount of memory it and its children use.

From that plot I think we can learn a few important things:

A big part of the memory is occupied by the output TBuffer, but the size seems to be constant over time as far as I could see from looking at different time-slices of the profiling information
The TPC tasks themselves (on the left, ExecuteTasks branch of the FairRunAna) seem to behave rather nicely
Something *very bad* is happening inside the FairMultiLinkedData (PndTpcElectronicsTasks is the one affected in the plot): Apparently a copy of the std::set inside the FairMultiLinkedData is temporarily using almost half a GB of memory (bottom left of the plot). Over the following time slices of the analysis this goes up to 1.7 GB (!), then drops again to 200, ...

So this seems to be the culprit responsible for the memory problems we see. As far as i can remember Tobias said he was going to remove the set structure (development branch) I didn't try this so far, since these heap analysis are taking ages to perform. Unfortunately I was not able to contact Tobias during the last days to discuss these findings, but I am sure we will solve the remaining questions together during the next weeks.

In my opinion these results confirm that the problem lies with the FairLinks (that guess was not very far-fetched as everybody seemed to agree that the crashes disappear when the FairLinks are not used), probably in combination with the fact that the TPC simulations creates a LOT of objects. The crashes themselves I still was not able to reproduce in many runs, and I tend to

believe that the reason might have been more on the technical side (remembering the discussion about GSI nfs I/O leading to TTree buffer crashes we had in the last evo meeting...).

So my personal agenda now will be to clean up the TPC code a bit, for now removing as many bookkeeping and referencing structures as possible. I will also try the development branch of Tobias and see if the memory load behavior improves.

Cheers

Felix

File Attachments

1) digi.hprof_3349.0048.heap.pdf, downloaded 610 times