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

Subject: Re: First results of TPC code profiling
Posted by Florian Uhlig on Fri, 31 Dec 2010 09:02:47 GMT
View Forum Message <> Reply to Message

Hi Felix

If the crash happens or not probably depends on the memory accessible on a machine. If you're allowed to use 2GB of Memory then the process will crash when it reaches this limit. On another machine this limit doesn't exist and you will have no crash. I am not speaking about the hardware memory but of the memory dedicated to the process by the OS. You can check this with the following command.

Quote:
> ulimit -a
core file size (blocks, -c) 0
data seg size (kbytes, -d) unlimited
max nice (-e) 0
file size (blocks, -f) unlimited
pending signals (-i) unlimited
max locked memory (kbytes, -l) 256000
max memory size (kbytes, -m) 256000
open files (-n) 8192
pipe size (512 bytes, -p) 8
POSIX message queues (bytes, -q) unlimited
max rt priority (-r) 0
stack size (kbytes, -s) 8192
cpu time (seconds, -t) 60000
max user processes (-u) unlimited
virtual memory (kbytes, -v) 1512000
file locks (-x) unlimited

The interesting information is max memory size. As you can see this value is very small on this interactive machines to prevent people from starting big jobs on this dedicated machine,

By the way there are some small scripts which should help people to find memory problems in the scripts directory of fairbase. Either you add this directory as external packages to pandaroot or you download it standalone. These scripts will not tell you where the actual problem is, but that there is a problem. They are easy to use and since they run in separate processes they will not slow down the root macro, which in my opinion is the real advantage. To isolate the problem you can switch on and of tasks to find the bad guy.

Hope this info helps.

Ciao and happy new year

Florian

Subject: Re: First results of TPC code profiling
Posted by StefanoSpataro on Fri, 07 Jan 2011 12:58:19 GMT
View Forum Message <> Reply to Message

Hi,
I find the plot quite interesting.
About the FairLink, the strange thing I have found some time ago is that the crash did not disappear removing all the set functions from the tpc code (this means leaving all the links empty), but removing the inheritance from the code.

I have tried Tobias dev code but "unfortunately" in the machine I have used I could not reproduce the crash neither with the "standard" code... probably I should use another machine or another seed.

Important question: how have you created such a plot? Which code have you used (and how)? I admit I have not tried yet the code suggested by Mohammad and Florian.

Many thanks for the tests. Have you also tried dpm (which seems to have a higher load)?

Subject: Re: First results of TPC code profiling
Posted by Felix Boehmer on Sat, 08 Jan 2011 10:32:35 GMT
View Forum Message <> Reply to Message

Hi Stefano,

I first tried to use TMemStat, but for some reason I could not even get it to work. I hoped that in this way I would be able to directly associate events and memory consumption.

The plot you see was created with the google performance tools and pprof. It works by replacing the normal libcmalloc for the program you want to examine, thus enabling heap analysis. This can be done by a LD_PRELOAD environment variable, so no re-compiling or re-linking is necessary.

However, it takes a while...

I have not tried DPM yet simply for time reasons. The generator used will of course not change the task behavior. But I do believe that with DPM you will get a lot more TPC objects (all the slow protons!) and will probably run into these kind of memory problems much quicker / more often.

I hope I could help to at least make things a little clearer.

Cheers,

Felix

Subject: Re: First results of TPC code profiling
Posted by StefanoSpataro on Wed, 09 Feb 2011 13:01:07 GMT
View Forum Message <> Reply to Message

Hi,
I am still trying to understand what is going wrong with the TPC digitization.
For this job I am using a i7 with 8Gb of memory, and I am visually monitoring the memory using the System Monitor given by the Ubuntu distribution.

I have run the run_digi_tpccombi.C, with all the detector digitization tasks, simulating 10k protons between 0.2 and 5 GeV/c.

I have seen that the used memory is in general quite stable. When I have started to take a look, it was stable around 750 Mb, without any variation.
After around 1000 events, I have seen that the digitization was stopping on a well defined event, in particular at this stage:

XXXXX electrons arriving at readout
Aggregating drifted electrons into avalanches finished.
XXXXX Avalanches created
0 aggregations done.
XXXXX Signals created
PndTpcElectronicsTask::Exec
Building up padmap ...finished. XXX pads hit
.....

In the middle of TPC digitization, still filling the line with dots ".". This procedure has taken few minutes, and at the end the memory has raised up to 1.3Gb... just this task!
The memory has remained stable at 1.3Gb for a while. After a bit, a "long" tpc event has raised it up to 1.4Gb, then stable again for many events, and after another "heavy" event up to 1.8Gb, then to 2.1Gb, then finally 2.7Gb: increasing only during specific events and in the middle of tpc tasks.

The following is the 1.8Gb->2.1Gb case (the only one I have copied):

********** PndEmcMakeBump, event: 7077 **********
EMC header: fired crystals= 14, digi= 10, Total energy= 0.295201 [GeV], Reconstructed

clusters= 6, Total energy in clusters= 0.290243 [GeV]
23481 electrons arriving at readout
Aggregating drifted electrons into avalanches finished.
23481 Avalanches created
0 aggregations done.
43941 Signals created
PndTpcElectronicsTask::Exec
Building up padmap ...finished. 406 pads hit
.....
2878 Digis created
PndTpcClusterFinderTask::Exec
number of digis: 2878
371 cluster created containing 2878 digis from 2878
Hit array contains 26 hits
PndEmcMakeCluster, event: 7078

From this check, I can argue the following points:

The memory consumption is stable for almost all the events, for all the tasks
For some well defined events the memory jumps up, in the middle of TPC digitization
The memory does not go down after the heavy events, it is not cleaned and it remains blocked

I think it should be important to understand what is making the memory growing inside the "guilty" TPC task, and how to clean the memory to go back to the original size. If not, it is easy to understand that if one has a "limited" memory and not 8Gb like my machine, then there is a crash. In this case, if I would use a 2Gb limited machine, i.e. GSI lxi or my VM -> crash.

If I remember some time ago there was some discussion because in that TPC task many links were created, then maybe also them could be the reason... but I think still that we should find some way to reduce the memory somehow during processing.

I leave the word to the experts...
