How to gain control of a 5GB heap in Haskell?

Asked 13 years, 5 months ago Modified 11 years, 2 months ago

Viewed 2k times

48

Currently I'm experimenting with a little Haskell web-server written in Snap that loads and makes available to the client a lot of data. And I have a very, very hard time gaining control over the server process. At random moments the process uses a lot of CPU for seconds to minutes and becomes irresponsive to client requests. Sometimes memory usage spikes (and sometimes drops) hundreds of megabytes within seconds.

Hopefully someone has more experience with long running Haskell processes that use lots of memory and can give me some pointers to make the thing more stable. I've been debugging the thing for days now and I'm starting to get a bit desperate here.

A little overview of my setup:

- On server startup I read about 5 gigabytes of data into a big (nested) Data.Map-alike structure in memory. The nested map is value strict and all values inside the map are of datatypes with all their field made strict as well. I've put a lot of time in ensuring no unevaluated thunks are left. The import

(depending on my system load) takes around 5-30 minutes. The strange thing is the fluctuation in consecutive runs is way bigger than I would expect, but that's a different problem.

- The big data structure lives inside a 'TVar' that is shared by all client threads spawned by the Snap server. Clients can request arbitrary parts of the data using a small query language. The amount of data request usually is small (upto 300kb or so) and only touches a small part of the data structure. All read-only request are done using a 'readTVarIO', so they don't require any STM transactions.
- The server is started with the following flags: +RTS -N -I0 -qg -qb. This starts the server in multi-threaded mode, disable idle-time and parallel GC. This seems to speed up the process a lot.

The server mostly runs without any problem. However, every now and then a client request times out and the CPU spikes to 100% (or even over 100%) and keeps doing this for a long while. Meanwhile the server does not respond to request anymore.

There are few reasons I can think of that might cause the CPU usage:

- The request just takes a lot of time because there is a lot of work to be done. This is somewhat unlikely because sometimes it happens for requests that have proven to be very fast in previous runs (with fast I mean 20-80ms or so).

- There are still some unevaluated thunks that need to be computed before the data can be processed and sent to the client. This is also unlikely, with the same reason as the previous point.
- Somehow garbage collection kicks in and start scanning my entire 5GB heap. I can imagine this can take up a lot of time.

The problem is that I have no clue how to figure out what is going on exactly and what to do about this. Because the import process takes such a long time profiling results don't show me anything useful. There seems to be no way to conditionally turn on and off the profiler from within code.

I personally suspect the GC is the problem here. I'm using GHC7 which seems to have a lot of options to tweak how GC works.

What GC settings do you recommend when using large heaps with generally very stable data?

performance **haskell** **garbage-collection** **memory-management**

edited Jul 8, 2011 at 12:36

asked Jul 8, 2011 at 10:47

Sebastiaan Visser
583 ● 3 ● 8

1 Hmmm.. interesting.. how much RAM the box has on which you are running this server application – Ankur Jul 8, 2011 at 12:15

There is a total of 8GB RAM on my machine. That should be enough. – Sebastiaan Visser Jul 8, 2011 at 12:37

Yeah.. that seems to be enough to avoid page faults – Ankur Jul 8, 2011 at 12:38

1 It would probably be worth it to pass in a large `-H` value to begin with. I don't think this will help with pauses, but it could help load times quite a bit. – sclv Jul 8, 2011 at 14:15

Can you use data types with less overheads? For example, HashMap has a smaller overhead than Map. – tibbe Oct 5, 2013 at 6:33

2 Answers

Sorted by: Highest score (default)

29

Large memory usage and occasional CPU spikes is almost certainly the GC kicking in. You can see if this is indeed the case by using RTS options like `-B`, which causes GHC to beep whenever there is a major collection, `-t` which will tell you statistics after the fact (in particular, see if the GC times are really long) or `-Dg`, which turns on debugging info for GC calls (though you need to compile with `-debug`).

There are several things you can do to alleviate this problem:

- On the initial import of the data, GHC is wasting a lot of time growing the heap. You can tell it to grab all of the memory you need at once by specifying a large `-H`.
- A large heap with stable data will get promoted to an old generation. If you increase the number of generations with `-G`, you may be able to get the stable data to be in the oldest, very rarely GC'd generation, whereas you have the more traditional young and old heaps above it.
- Depending the on the memory usage of the rest of the application, you can use `-F` to tweak how much GHC will let the old generation grow before collecting it again. You may be able to tweak this parameter to make this un-garbage collected.
- If there are no writes, and you have a well-defined interface, it may be worthwhile making this memory un-managed by GHC (use the C FFI) so that there is no chance of a super-GC ever.

These are all speculation, so please test with your particular application.

answered Jul 8, 2011 at 12:45

Edward Z. Yang

26.7k ● 16 ● 86 ● 116

5 Using the compacting GC can also be useful. – Don Stewart Jul 8, 2011 at 13:40

I haven't tested it thoroughly yet, but increasing the number of generations does seem to have a positive effect.
– Sebastiaan Visser Jul 9, 2011 at 12:04

2

I had a very similar issue with a 1.5GB heap of nested Maps. With the idle GC on by default I would get 3-4 secs of freeze on every GC, and with the idle GC off (+RTS -I0), I would get 17 secs of freeze after a few hundred queries, causing a client time-out.

My "solution" was first to increase the client time-out and asking that people tolerate that while 98% of queries were about 500ms, about 2% of the queries would be dead slow. However, wanting a better solution, I ended up running two load-balanced servers and taking them offline from the cluster for performGC every 200 queries, then back in action.

Adding insult to injury, this was a rewrite of an original Python program, which never had such problems. In fairness, we did get about 40% performance increase, dead-easy parallelization and a more stable codebase. But this pesky GC problem...

Share Improve this answer Follow

answered Oct 4, 2013 at 16:56

Finn Espen Gundersen
93 • 6