How to read gprof output

This is a brief little tutorial I wrote on reading the output of gprof, a Profiling tool available on most unix systems. (Profiling measures what pieces of code in a program consume the most time)

(note that gprof is not a GNU tool. The 'g' probably stands for "call Graph" profiler) You'll need to check your system's documentation (e.g. `man gprof`) for exact instructions on getting gprof to work. Usually it just involves compiling and linking with `-pg`, running your program, and doing `gprof gmon.out > oopack`)

Here's a 300K sample of output from gprof on the Dec Alpha if you want to take a look at it. This particular report is from a run of AOLServer 2.2.1 which involved fetching index.html 53,623 times.

There's 4 parts to gprof output:

- Built-in documentation: Short form of everything here, and more.
- Call-graph: Each function, who called it, whom it called, and how many times
- Flat profile How many times each function got called, total times involved, sorted by time consumed.
- Index: Cross-reference of function names and gprof numbers.

When I first start looking gprof output, I go to the flat profile section. There it's usually black-and-white who the big time consumers are. You'll notice that each function has a [number] after it. You can search on that number throughout the file to see who calls that function and whom that function calls. Emacs incremental search is really nice for this. Here you can see that DString is a big time gobbler:

```
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 17.7       3.72     3.72 13786208     0.00     0.00  Ns_DStringNAppend [8]
  6.1       5.00     1.28   107276     0.01     0.03  MakePath [10]
  2.9       5.60     0.60  1555972     0.00     0.00  Ns_DStringFree [35]
  2.7       6.18     0.58  1555965     0.00     0.00  Ns_DStringInit [36]
  2.3       6.67     0.49  1507858     0.00     0.00  ns_realloc [40]
```

Out of 21.05 seconds of total clock time, Ns_DStringNAppend consumed about 4 seconds (about 18% of the time) in and of itself. It was called 13 million times.

MakePath consumed one and a half seconds itself, and its children consumed three and a half seconds. At least one individual call to this consumed 0.01, and at least one individual call took a total of 0.03 seconds in MakePath and its children.

Handy tip - the function numbers in brackets are approximately sorted by time consumption, so a function with a [low number] will generally be more interesting than one with a [high number].

Now that we know that Ns_DStringNAppend is called a bunch of times, and could be a useful target for optimization, I'd look at its entry in the call graph section.

Before doing that, just for illustration, take a look at AllocateCa [33] since it has all of the interesting pieces of the call graph in a more compact size:

```
                0.04        0.18   53622/160866      Ns_CacheNewEntry [62]
                0.04        0.18   53622/160866      Ns_CacheDoStat [58]
                0.04        0.18   53622/160866      Ns_CacheLockURL [64]
[33]     3.0    0.11        0.53  160866         AllocateCa [33]
                0.16        0.17  160866/321890      Ns_DStringVarAppend [30]
                0.06        0.00  160866/1555972     Ns_DStringFree [35]
                0.06        0.00  160866/1555965     Ns_DStringInit [36]
                0.04        0.00  160866/1341534     Ns_LockMutex [43]
                0.03        0.00  160866/1341534     Ns_UnlockMutex [53]
```

The entries above AllocateCa [33] are the functions that call AllocateCa. The entries below that are the functions that AllocateCa calls. For the numbers separated by a slash, the first number is the number of calls that the function has made, and the second number is the total number of invocations of that function.

In other words, for 160866/321890 Ns_DStringVarAppend [30], this means that AllocateCa called Ns_DStringVarAppend 160866 times. Across all of AOLServer, Ns_DStringVarAppend was called 321890 times.

Similarly, for 53622/160866 Ns_CacheNewEntry [62], this means that Ns_CacheNewEntry called AllocateCa 53622 times, and AllocateCa was called 160866 times total.

So, just by looking at this snippet, you know that The three Ns_Cache functions each call AllocateCa about once per serving of index.html, and that AllocateCa makes a single call to Ns_DStringVarAppend, Ns_DStringFree, etc... each time. What's also interesting to note is that someone is calling Ns_DStringFree more than Ns_DStringInit. This may be (or may not) be a bug in AOLServer. You can go see Ns_DStringInit and Ns_DStringFree yourself and track down who the culprit is.

The floating "3.0" is the percent of total time that function consumed. The two columns of numbers are the amount of time (in seconds) that the function consumed itself (AllocateCa took 0.11 seconds of time total to run its own code) and the amount of time in the function's children (0.53 seconds were spent in its children)

Getting back to real analysis of DStringNAppend, we can see that MakePath made 50% of the Ns_DStringNAppend calls. Since we know that there were 53623 fetches of index.html, that means that for each page, MakePath was called twice, and for each call to MakePath, Ns_DStringNAppend was called 64 times.

If one call to MakePath could be elided (since it's getting called twice), or if fewer than 64 Ns_DStringNAppends could be done per call, we could see a performance boost.

Just browsing the gprof output can be an illuminating exercise. If you have a gut feeling that a particular function is a hot spot (say, Ns_LockMutex [43]), you can see the call graph for that function, see if it's consuming lots of time, or if it's being called a whole bunch (hmm, was called 1,341,534 times, or about 25 times per page serve). Sometimes a suspected culprit isn't there, or you find a surprising time waster.

Note that because this sample gprof output was done on an Alpha, which has some suckage involved, such as no explicit time recorded for system calls, so we don't know if, for example, select() blocked for a long time on each call.

[back to Badgertronics]
markd@badgertronics.com