



An overview of q-syscollect and q-view

David Mosberger



q-syscollect + q-view = gprof
— without the pain

Example program “p-fan”

- Loop calling 10 functions doing varying amount of work:

```
long f00(long x){return x+x;}           long f01(long x){return x*x;}
long f02(long x){return 1000 / x;}     long f03(long x){return x+x*x;}
long f04(long x){return x+x*(x+x);}    long f05(long x){return 1.0/x;}
long f06(long x){return sin(x);}       long f07(long x){return cos(x);}
long f08(long x){return sin(x)*cos(x);} long f09(long x){return tan(x);}

int main (int argc, char **argv) {
    unsigned long sum = 0, x = 0;
    while (1) {
        sum += f00(x++); sum += f01(x++); sum += f02(x++); sum += f03(x++);
        sum += f04(x++); sum += f05(x++); sum += f06(x++); sum += f07(x++);
        sum += f08(x++); sum += f09(x++);
        x &= 1023;
    }
    return sum;
}
```

Example: q-syscollect in action

- Build & start p-fan:

```
$ cc -O2 p-fan.c -o p-fan # Note: no special flags; just regular compile
$ p-fan & # runs forever
```

- Run q-syscollect for 20 seconds:

```
$ q-syscollect -t 20 -v
[12076] profiling on cpu 0 with clock-frequency 900000000 Hz
[12076] collected 25 BTB samples, 100 code samples in 0.10065 seconds
[12076] collected 472 BTB samples, 998 code samples in 1.00195 seconds
...snip...
[12076] collected 945 BTB samples, 998 code samples in 1.00098 seconds
[12076] termination-request received; shutting down
[12076] collected 828 BTB samples, 875 code samples in 0.87809 seconds
```

- Look at data files created by q-syscollect:

```
$ ls .q
p-fan-pid11920-cpu0.edge#0  xterm-pid11888-cpu0.edge#0
p-fan-pid11920-cpu0.hist#0  xterm-pid11888-cpu0.hist#0
p-fan-pid11920-cpu0.info#0  xterm-pid11888-cpu0.info#0
```

Example: q-view in action

- Look at profile for p-fan:

```

$ q-view .q/p-fan-pid11920-cpu0.info
Command: /home/davidm/src/q-tools/tests/p-fan
Flat profile of CPU_CYCLES in p-fan-pid11920-cpu0.hist#0:
Each histogram sample counts as 1.00057m seconds
% time  self  cumul  calls  self/call  tot/call  name
36.75  7.33   7.33   120M    61.2n     61.2n    cos
10.46  2.08   9.41   29.3M   71.2n     71.2n    tan
 8.91  1.78  11.19   -        -         -        main
 6.56  1.31  12.49  29.8M   43.9n     43.9n    __divdi3
 5.88  1.17  13.67  29.6M   39.7n     163n     f08
 5.23  1.04  14.71  30.2M   34.5n     34.5n    __divdf3
 3.33  0.66  15.37  29.2M   22.7n     95.3n    f09
 3.26  0.65  16.02  30.2M   21.5n     84.3n    f06
 3.18  0.63  16.66  30.2M   21.0n     55.5n    f05
 3.09  0.62  17.27  30.1M   20.5n     82.2n    f07
 3.02  0.60  17.88   -        -         -        sin
 2.67  0.53  18.41   -        -         -        _init
 2.00  0.40  18.81  29.6M   13.5n     13.5n    f01
 1.90  0.38  19.19  30.2M   12.6n     12.6n    f04
 1.75  0.35  19.54  29.8M   11.7n     11.7n    f03
 0.99  0.20  19.73  29.9M    6.59n     50.3n    f02
 0.56  0.11  19.84  1.37M   81.8n     81.8n    0x19750<libm-2.3.2.so>
 0.19  0.04  19.88   208k     183n     183n    matherr1
 0.18  0.04  19.92  29.2M    1.20n     1.20n    f00
  
```

Example: q-view in action (cont.)

- Call-graph portion of profile for p-fan:

main called f09
22.9M times out
of 22.9M calls

call-graph table:

index	%time	self	children	called	name
[20]	99.8	1.78	17.0	-	<spontaneous> main
		0.663	2.12	29.2M/29.2M	f09 [21]
		1.17	3.65	29.6M/29.6M	f08 [25]
		0.616	1.86	30.1M/30.1M	f07 [26]
		0.649	1.90	30.2M/30.2M	f06 [27]
		0.634	1.04	30.2M/30.2M	f05 [31]
		0.379	0.00	30.2M/30.2M	f04 [33]
		0.349	0.00	29.8M/29.8M	f03 [34]
		0.197	1.31	29.9M/29.9M	f02 [35]
		0.399	0.00	29.6M/29.6M	f01 [37]
		35.0m	0.00	29.2M/29.2M	f00 [38]

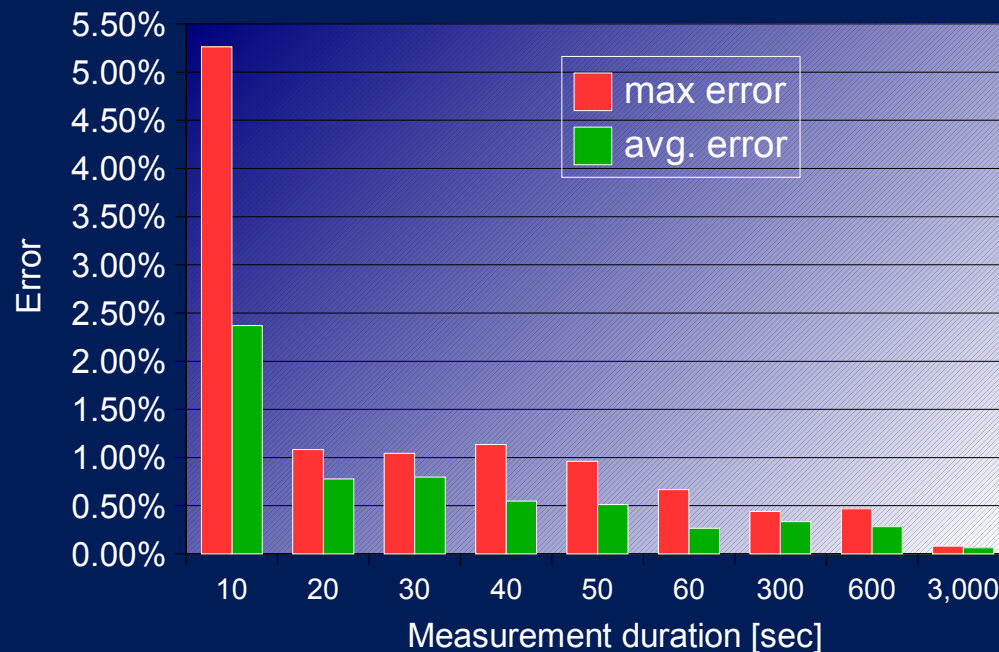
		3.61	0.00	59.1M	f08 [25]
		1.84	0.00	30.0M	f07 [26]
		1.88	0.00	30.7M	f06 [27]
[30]	39.0	7.33	0.00	120M	cos

...snip...					

percentage
of time in main
and children

How accurate are the call counts?

- Empirically, accuracy good even for relatively short runs (e.g., ~20–30 sec) and complex call-graphs
- **Caveat:** infrequent calls may be missed completely, which can disconnect portions of the call-graph
- Error analysis for f00()-f09() in “p-fan”:



Motivation & technology behind q-syscollect and q-view

Motivation

- Confluence of three streams...
 - **Need**
 - lack of quality open-source profiling tools for Linux
 - bad enough for x86 — worse for non-x86
 - **Interest**
 - HP Labs Linux group has research interest in performance tools
 - **Opportunity**
 - offered by Itanium 2 perf. monitoring unit (PMU)
 - powerful perfmon kernel subsystem in ia64 linux
- ...resulted in a desire for:
 - simple yet flexible infrastructure for building performance tools
 - a tool that gives gprof-like results — without the pain

The q-file-format

- Considered building on top of oprofile, but its file-format appeared rather ad-hoc and limited
- Goals for q-file format:
 - self-describing & extensible
 - text-based (ASCII) with support for separate data files
- Approach:
 - nested name/value pairs in the form of Lisp-style s-expressions
 - XML possible, but would be much more verbose
- Example:
 - Expression specifying a filename:
`(q:file . "p-fan-pid11920-cpu0.hist#0")`



name/tag



value

How q-syscollect uses the q-file format

- Generates one q-formatted file per monitored task
 - these files have “.info” extension
- All files generated in a sub-directory called “.q”
 - use environment variable `Q_DIR` to specify any other directory
- Uniqueness ensured with version number (think VMS...):
 - `p-fan.info#0`
 - `p-fan.info#1`
- Dependent files are checksummed and (virtually) copied into .q directory:
 - e.g., `/usr/libc-2.3.2.so` → `.libc-2.3.2.so.crc32.a23bac21`
 - collect all files required for data-analysis with single command:
`$ tar chvf files.tar .q`
- Histogram & edge data stored in separate text files

Example of q-syscollect-produced files

- “.info” file:

```
(q:info '((q:syscollect-cmdline . ("q-syscollect" "-t" "20" "-v"))
          (q:cmdline . ("p-fan"))))
(q:histogram '((q:file . "p-fan-pid11920-cpu0.hist#0")
              (q:event-name . "CPU_CYCLES") (q:x-unit-label . "address")
              (q:y-unit-label . "seconds") (q:y-unit-conversion-factor . 1.1e-09)
              (q:y-granularity . 0.00100057)))
(q:call-counts '((q:file . "p-fan-pid11920-cpu0.edge#0")))
(q:kallsyms ".kallsyms.crc32.c23c2a57")
(q:object '((q:name . "p-fan") (q:file . ".p-fan.crc32.eb5a8b94")
           (q:maps .
            ((q:addr . #x4000000000000000) (q:size . 16384) (q:offset . #x0))
            ((q:addr . #x6000000000000000) (q:size . 16384) (q:offset . #x0))))))
```

- “.hist” file:

	<i>address</i>	<i>count</i>
:		
0xa000000100097760		1801105
0xa0000001000a1170		2700565
0x40000000000000500		129674717
0x40000000000000510		66638842
0x40000000000000520		114367398
:		

Why separate data files?

- Extremely easy to post-process with other tools, e.g.:
 - Use “sort” command to see instruction-level histogram
 - Use “ls -l *.hist*” to find longest-running program (max. size)
 - Use “gnuplot” to plot the same
 - Use “cat” to merge several histogram- or edge-files
 - etc.
- Wouldn't a binary format be more efficient?
 - yes, but note that explicit address/value pairs are used in the histogram, so the file-size is proportional to the size of the code executed, not the static code size!
 - if space is really an issue, we could easily add support for compressed data files
 - preserves the advantages of text-files with space-efficiency of binary files

Summary

- q-syscollect & q-view are quite powerful already and extremely easy to use
 - installation remains a bit tricky for the time being:
 - requires 2.6-based kernel, libpfm 3.x, GUILE (with SLIB), ...
 - most of these issues will go away as distributions get updated
- The existing tools have already proven effective in a number of situations and applications
 - resulted in significant improvements in the Linux **kernel**, GNU **linker**, network-intensive server, graphics toolkit, etc.
 - several of these improvements were enabled by call-graph info
- The tools & infrastructure are open source and beg for experimentation & exploration
 - if you have a great idea for a new performance-tool, go wild!