Application and system profiling

Tools and methods

Tazio Ceri

Cynny

March 8, 2017

Presentation plan

1. Traditional application profiling tools
   - GProf
   - Valgrind cachegrind
2. System profiling tools
   - Linux observability tools
   - strace
3. Perf
   - Perf commands
   - Perf examples
   - Who can use Perf?
4. Methods
   - USE Method
   - TSA Method
5. Anti-method
   - Ionut Anti-method
   - Streetlight Anti-Method
   - Drunk man Anti-Method
6. Thanks!

GProf, the best technology from the 80s

How it works

- it requires a special compiling option (-pg);
- the options inserts the mcount and profil function call;
- program execution generates the gmon.out file;
- gprof is used to analyze it.
- It generates a call graph;
- it counts calls;
- it samples code execution.

GProf problems

Don't perturb what you measure!

The *-pg* option modifies the resulting code, repeated *mcount* calls modify timings, *profil* samples execution, by executing from a timer every 100μsec, perturbing further measuremens.

Right from the man page

The granularity of the sampling is shown, but remains statistical at best. We assume that the time for each execution of a function can be expressed by the total time for the function divided by the number of times the function is called. Thus the time propagated along the call graph arcs to the function's parents is directly proportional to the number of times that arc is traversed.

Reproducing the problem

Code

```
void work(int n) {
    volatile int i=0; //don't optimize away
    while(i++ < n);
}
void easy() { work(1000); }
void hard() { work(1000*1000*1000); }
int main() { easy(); hard(); }
```

Make it run!

Results

self seconds	calls	self s/call	total s/call	name
2.17	2	1.08	1.08	work
0.00	1	0.00	1.08	easy
0.00	1	0.00	1.08	hard

- *easy* and *hard* have a different total execution time;
- but gprof can not measure that because al work is done inside of *work*!
- Information about *work* does not record the caller, so its execution time is shared between all callers equally.
- Execution time is reliable only when a function has only one caller or when its execution time are "stable", in a sense that they are not much dependant on the arguments.

Conclusions

So long gprof, and thanks for all the samples!

- GProf is a *developer only* tool;
- perturbates the problem;
- it is also can not gather good results;
- it needs compiler support;
- only the callgraph is reliable;
- **do not use it!**

Linus Torvalds said...

Do not use gprof. You are much better off using the newish Linux perf tool.

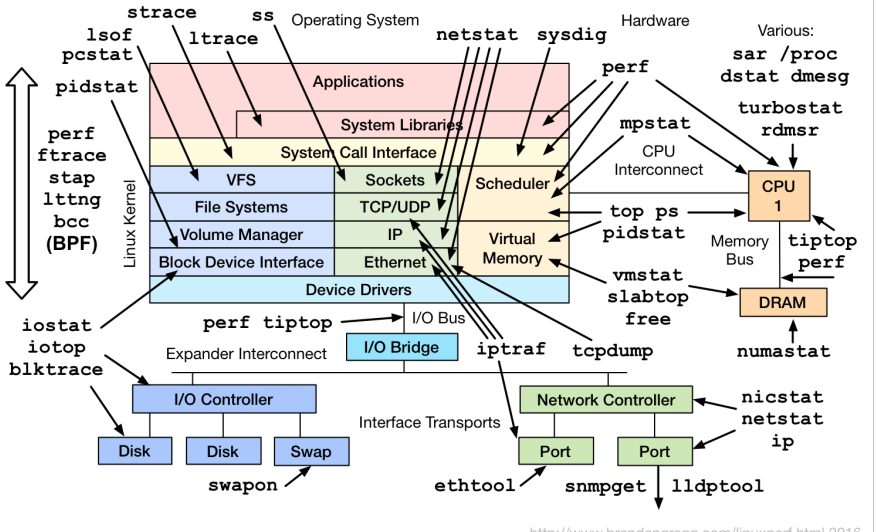
I can pretty much guarantee that once you start using it, you will never use gprof or oprofile again.

Cachegrind

What is it

- A processor simulator;
- its execution is terribly slow;
- harvest a lot more data than gprof;
- as a simulator, not always perfect, only useful for L3 cache for example;
- in the end, it's a developer only tool, for application that might get slow.
- *kcachegrind* can lie in its call *tree* in a similar way as *gprof*

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html 2016

Linux observability tools

A lot of tools

- Observing the whole living system,
- every layer of the stack;
- some tools are user space;
- the best tools need kernel support;
- now an overview!

strace

What is it?

- It traces right every syscall, complete with arguments and return code;
- this is mainly a sysadmin tool;
- sometimes useful to debug, by finding where the program is stuck;
- useful to understand how a program works;
- with -tT can print useful timing information.

strace -Tttt output

```
14:40:45.762935 poll([{fd=5, events=POLLIN|POLLPRI},
    {fd=7, events=POLLIN|POLLPRI},
    {fd=6, events=POLLIN|POLLPRI}], 3, 2009) = 1
    ([{fd=5, revents=POLLIN}]) <0.044169>
14:40:45.807167 recvfrom(5,
    "+\307Ry\310u#\352\357\1\0\0\0\24=dz\261\305\317\37\21", 1545, 0,
    {sa_family=AF_INET, sin_port=htons(10000),
    sin_addr=inet_addr("87.17.203.70")}, [16]) = 22 <0.000008>
14:40:45.807240 poll([{fd=5, events=POLLIN|POLLPRI|POLLOUT},
    {fd=7, events=0}, {fd=6, events=POLLIN|POLLPRI}], 3, 2009) = 1
    ([{fd=5, revents=POLLOUT}]) <0.000007>
14:40:45.807273 sendto(5,
    "#=dz\261\305\317\37\21\0\0\0\0\30\35\335,\20tX\235\7\273\215\0
    \267,\253\35\4{\346"..., 114, 0, {sa_family=AF_INET, sin_port=htons(10000),
    sin_addr=inet_addr("87.17.203.70")}, 16) = 114 <0.000032>
```

stracing strace

```
--- SIGCHLD {si_signo=SIGCHLD, si_code=CLD_TRAPPED, si_pid=12883, si_uid=479,
    si_status=SIGTRAP, si_utime=343, si_stime=169} ---
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
wait4(-1, [{WIFSTOPPED(s) && WSTOPSIG(s) == SIGTRAP | 0x80}], __WALL, NULL)
    = 12883
rt_sigprocmask(SIG_BLOCK, [HUP INT QUIT PIPE TERM], NULL, 8) = 0
ptrace(PTRACE_GETREGSET, 12883, NT_PRSTATUS, [{0x67f500, 216}]) = 0
process_vm_readv(12883, [{"...", 32}], 1, [{0x563dd1a3584f, 32}], 1, 0) = 32
write(2, "sendto(5, \"5\\342\\314v\\253\\270f_\\"..., 200) = 200
ptrace(PTRACE_SYSCALL, 12883, 0, SIG_0) = 0
```

PTRACE_SYSCALL

Restart the stopped tracee as for PTRACE_CONT, but arrange for the tracee to be stopped at the next entry to or exit from a system call, or after execution of a single instruction, respectively.

Performance impact on a stracee

dd if=/dev/zero of=/dev/null bs=2M count=2048

4294967296 bytes (4,3 GB, 4,0 GiB) copied, 0,265338 s, 16,2 GB/s

strace dd if=/dev/zero of=/dev/null bs=2M count=2048

4294967296 bytes (4,3 GB, 4,0 GiB) copied, 0,416209 s, 10,3 GB/s

- It perturbates too much to be a real profiling tool!

A tool for both worlds: commands (1)

The most commonly used perf commands:

- **annotate** Read perf.data (created by perf record) and display annotated code;
- **archive** Create archive with object files with build-ids found in perf.data file;
- **bench** General framework for benchmark suites;
- **buildid-cache** Manage build-id cache;
- **buildid-list** List the buildids in a perf.data file
- **data** Data file related processing
- **diff** Read perf.data files and display the differential profile
- **evlist** List the event names in a perf.data file
- **inject** Filter to augment the events stream with additional information
- **kmem** Tool to trace/measure kernel memory properties
- **kvm** Tool to trace/measure kvm guest os
- **list** List all symbolic event types

A tool for both worlds: commands (2)

- lock Analyze lock events
- **mem** Profile memory accesses
- **record** Run a command and record its profile into perf.data
- **report** Read perf.data (created by perf record) and display the profile
- **sched** Tool to trace/measure scheduler properties (latencies)
- **script** Read perf.data (created by perf record) and display trace output
- **stat** Run a command and gather performance counter statistics
- **test** Runs sanity tests.
- **timechart** Tool to visualize total system behavior during a workload

A tool for both worlds: commands (3)

- **top** System profiling tool.
- **trace** strace inspired tool
- **probe** Define new dynamic tracepoints

perf top: Not processes...only functions!

```
Samples: 159K of event 'cycles:pp', Event count (approx.): 23336682401
Overhead  Shared Object                       Symbol
  2,69%  libglib-2.0.so.0.4800.2             [.] g_mutex_lock
  2,67%  [kernel]                            [k] dw_readl.isra.4
  2,43%  libglib-2.0.so.0.4800.2             [.] g_mutex_unlock
  2,40%  [kernel]                            [k] __fget
  1,35%  [kernel]                            [k] fput
  1,33%  libpthread-2.22.so                  [.] pthread_mutex_lock
  1,21%  libc-2.22.so                        [.] _int_malloc
  1,19%  libpthread-2.22.so                  [.] __pthread_mutex_unlock_usercnt
  1,14%  [vdso]                              [.] __vdso_clock_gettime
  1,12%  [kernel]                            [k] copy_user_enhanced_fast_string
  1,04%  [kernel]                            [k] do_sys_poll
  0,99%  perf                                [.] rb_next
  0,98%  [kernel]                            [k] entry_SYSCALL_64_after_swapgs
  0,98%  libglib-2.0.so.0.4800.2             [.] g_main_context_check
  0,93%  [kernel]                            [.] syscall_return_via_sysret
  0,90%  [kernel]                            [k] _raw_spin_lock
  0,75%  [kernel]                            [k] unix_poll
  0,73%  libc-2.22.so                        [.] _int_free
  0,72%  libc-2.22.so                        [.] malloc
```

perf mem: loads, stores, and misses

```
Samples: 20K of event 'cpu/mem-loads/pp', Event count (approx.): 234263
Overhead       Samples  Local Weight  Memory access         Symbol
  1,72%            574  7             L1 hit                [k] poll_idle
  0,87%            290  7             L1 hit                [k] poll_idle
  0,84%            282  7             L1 hit                [k] poll_idle
  0,72%            240  7             L1 hit                [k] poll_idle
  0,55%              1  1283          LFB hit               [.] QQmlNotifier::emitNotify
  0,46%              1  1072          L1 hit                [.] _int_free
  0,44%              1  1027          L3 hit                [k] 0x0000557d799f75d2
  0,43%              1  999           L1 hit                [.] 0x0000000002b1d6df
  0,40%              1  942           Local RAM hit         [.] 0x000000000297d13d
  0,38%              1  897           LFB hit               [.] 0x00000000000179b9
  0,37%              1  864           LFB hit               [.] _int_malloc
  0,36%              1  842           L1 hit                [.] 0x000000000015a7e0
  0,33%              1  765           L2 hit                [.] QWidgetPrivate::getOpaqueChildren
  0,32%              1  751           Local RAM hit         [.] 0x0000000000c8c3d6
  0,30%              1  697           Local RAM hit         [k] 0x00007f81773d1d96
  0,27%              1  634           L1 hit                [k] __fget
  0,26%              1  619           L1 hit                [k] _raw_spin_lock_irqsave
  0,25%              1  583           LFB hit               [.] 0x00007f8106f6fcae
  0,21%              1  486           Local RAM hit         [k] unix_poll
  0,20%              1  477           LFB hit               [k] unix_poll
  0,20%              1  462           L2 hit                [k] __switch_to
```

perf stat: the powerful cousin of time

```
Performance counter stats for 'ls':

          0,821735      task-clock (msec)         #    0,276 CPUs utilized
                 1      context-switches          #    0,001 M/sec
                 0      cpu-migrations            #    0,000 K/sec
               117      page-faults               #    0,142 M/sec
           2284049      cycles                    #    2,780 GHz
   <not supported>      stalled-cycles-frontend
   <not supported>      stalled-cycles-backend
           2239819      instructions              #    0,98  insns per cycle
            461991      branches                  #  562,214 M/sec
             16775      branch-misses             #    3,63% of all branches

       0,002978879 seconds time elapsed
```

A lot other hardware and software counters!

perf list: events and parties!

```
instructions                                       [Hardware event]
...
L1-icache-load-misses                              [Hardware cache event]
...
alignment-faults                                   [Software event]
...
branch-instructions OR cpu/branch-instructions/    [Kernel PMU event]
branch-misses OR cpu/branch-misses/                [Kernel PMU event]
...
ext4:ext4_allocate_inode                           [Tracepoint event]
```

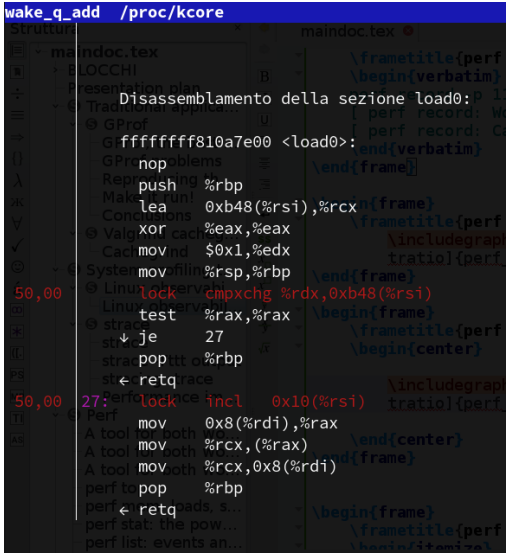
perf record on a running process

```
perf record -g -p 11597
[ perf record: Woken up 2 times to write data ]
    [ perf record: Captured and wrote 0.715 MB perf.data (450 samples) ]
```

perf report

```
Samples: 219  of event 'cycles:pp', Event count (approx.): 610626687
  Children      Self  Command          Shared Object                 Symbol
+    8,18%     8,18%  konsole          libc-2.22.so                  [.] _int_free
+    6,67%     0,00%  konsole          [unknown]                     [k] 0000000000000000
+    4,96%     4,96%  bash             libc-2.22.so                  [.] __mbrtowc
+    3,96%     0,00%  bash             [kernel.kallsyms]             [k] __do_page_fault
+    3,96%     0,00%  bash             [kernel.kallsyms]             [k] do_page_fault
+    3,96%     0,00%  bash             [kernel.kallsyms]             [k] page_fault
+    3,82%     0,00%  konsole          [kernel.kallsyms]             [k] entry_SYSCALL_64_fastpath
+    3,72%     0,00%  konsole          [unknown]                     [.] 0x000000270000006e
+    3,72%     0,00%  konsole          [unknown]                     [.] 0x00007fa8800000f4
+    3,63%     3,63%  konsole          libQt5Core.so.5.6.1           [.] 0x0000000000307083
+    3,63%     0,00%  konsole          libQt5Core.so.5.6.1           [.] 0xffff8057dd074083
+    3,41%     3,41%  konsole          libc-2.22.so                  [.] malloc
+    3,28%     0,00%  konsole          [unknown]                     [.] 0x0000000002111c28
+    3,28%     0,00%  konsole          [unknown]                     [.] 0x00007ffc603eeb70
+    3,28%     0,00%  konsole          [unknown]                     [.] 0x0000000002111c26
+    3,25%     0,00%  konsole          libfreetype.so.6.12.3         [.] 0xffff8057e576e4b0
-    3,15%     0,90%  konsole          libc-2.22.so                  [.] __GI___libc_read
   + 2,25% __GI___libc_read
   + 0,90% 0x8b48078948fb8948
+    2,81%     2,81%  konsole          libc-2.22.so                  [.] __memset_avx2
```

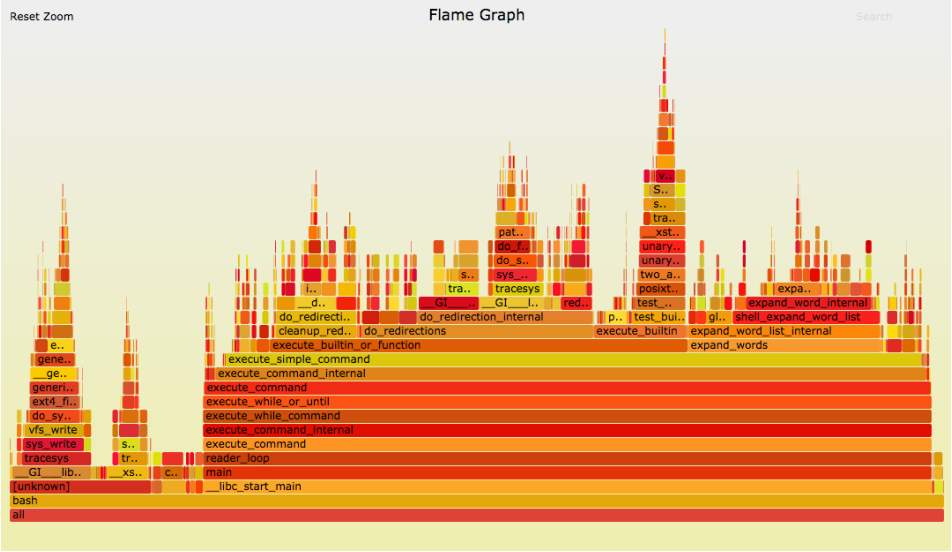
perf annotate (or annotating a function)



perf and flames!

- perf record -g -p 11597
- perf script | ./stackcollapse-perf.pl > out.perf-folded
- ./flamegraph.pl out.perf-folded > kernel.svg

flaming results



Passing an image: by reference or by value?

```
class large {
    char buffer[1024*1024*6];
    ....
};
void g( large& l);
void f( large l);

// Call one of them 4096 times!
```

Passing an image: by value

```
1089,608100      task-clock (msec)  #    1,000 CPUs utilized
2       context-switches            #    0,002 K/sec
0       cpu-migrations              #    0,000 K/sec
1634       page-faults              #    0,001 M/sec
3605460470       cycles             #    3,309 GHz
2827058349       instructions       #    0,78  insns per cycle
202960046       branches            #  186,269 M/sec
32447       branch-misses           #    0,02% of all branches

1,090117688 seconds time elapsed
```

Passing an image: by reference

```
2,611257      task-clock (msec)    #    0,845 CPUs utilized
0       context-switches           #    0,000 K/sec
0       cpu-migrations             #    0,000 K/sec
97       page-faults               #    0,037 M/sec
3380835       cycles               #    1,295 GHz
5512457       instructions         #    1,63  insns per cycle
1056591       branches             #  404,629 M/sec
14094       branch-misses          #    1,33% of all branches

0,003089825 seconds time elapsed
```

Passing an image by copy: let's see the slowness!

```
Samples: 19  of event 'cycles:pp', Event count (approx.): 15872257530057
  Children      Self  Command  Shared Object       Symbol
+  100,00%  100,00%  a.out    libc-2.22.so        [.] __memcpy_avx_unaligned
+  100,00%    0,00%  a.out    libc-2.22.so        [.] __libc_start_main
+    0,00%    0,00%  a.out    ld-2.22.so          [.] _dl_relocate_object
+    0,00%    0,00%  a.out    ld-2.22.so          [.] dl_main
+    0,00%    0,00%  a.out    ld-2.22.so          [.] _dl_sysdep_start
+    0,00%    0,00%  a.out    ld-2.22.so          [.] _dl_lookup_symbol_x
+    0,00%    0,00%  a.out    [kernel.kallsyms]   [k] mmap_region
+    0,00%    0,00%  a.out    [kernel.kallsyms]   [k] do_mmap
+    0,00%    0,00%  a.out    [kernel.kallsyms]   [k] vm_mmap_pgoff
+    0,00%    0,00%  a.out    [kernel.kallsyms]   [k] sys_mmap_pgoff
+    0,00%    0,00%  a.out    [kernel.kallsyms]   [k] entry_SYSCALL_64_fastpath
```

What is it?

Basically the whole execution time happens in copies!

Passing an image by reference: let's see the slowness!

```
Samples: 10  of event 'cycles:pp', Event count (approx.): 9895631470992
  Children      Self  Command  Shared Object       Symbol
+  100,00%     0,00%  a.out    ld-2.22.so          [.] dl_main
+  100,00%   100,00%  a.out    libc-2.22.so        [.] strpbrk
+  100,00%     0,00%  a.out    ld-2.22.so          [.] _dl_sysdep_start
+    0,00%     0,00%  a.out    ld-2.22.so          [.] read
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] copy_user_enhanced_fast_string
+    0,00%     0,00%  a.out    ld-2.22.so          [.] _dl_map_object
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] __vma_link_rb
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] vma_link
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] mmap_region
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] do_mmap
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] vm_mmap_pgoff
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] load_elf_binary
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] search_binary_handler
+    0,00%     0,00%  a.out    [kernel.kallsyms]   [k] do_execveat_common.isra.32
```

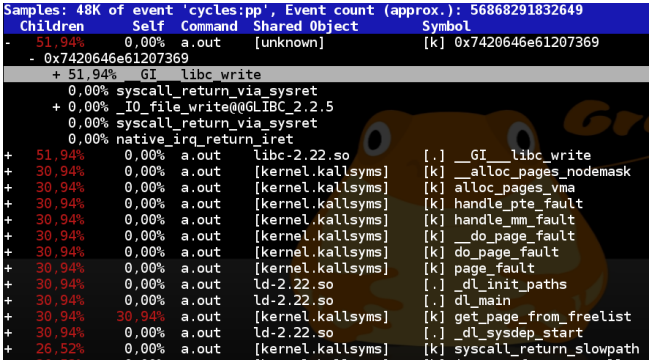
What is it?

strpbrg is a function that works on strings to find byte patterns. It has nothing to do with references.

How slow is iostream?

```
int calculate( int f1, int f2, int step )
{
   cout << "Debug this and that" << endl; // Turn it on
   if ( step == 0 ) return 0.0;
   if ( f1 < 1 ) f1 = 1;
   if ( f2 < 1 ) f2 = 1;
   return calculate( f2, f1 + f2, --step );
}
```

Creating havoc all over the system



What is it?

Memory, I/O... execution time jumps from milliseconds to ages.

Unaligned access leads to ...

```
struct __attribute__ ((packed)) madworld
{
   long double e;
   long double f;
};
```

What will happen?

How much we will gain removing the packed attribute?

Unaligned access leads to speed

```
1,524733    task-clock (msec)    #    0,843 CPUs utilized
109         page-faults          #    0,071 M/sec
5759003     instructions
806587      branches             #  529,002 M/sec
70805       cache-misses         #   24,891 % of all cache refs
284465      cache-references     #  186,567 M/sec

0,001809230 seconds time elapsed
```

Aligned access is not so sad anyway

```
1,656178      task-clock (msec)         #    0,863 CPUs utilized
622           page-faults               #    0,376 M/sec
6924223       instructions
1008788       branches                  #  609,106 M/sec
69772         cache-misses              #   33,095 % of all cache refs
210825        cache-references          #  127,296 M/sec

0,001918302 seconds time elapsed
```

So what?

There are cases where unaligned access is slightly faster. Here they are close dispite all the page faults! Continue to align, please.

The Java EE programmer

I am a senior Java Programmer, so I almost never profile or optimize my programs, but when I do, I could also use Perf. Brendan Gregg (not the guy in the pic) claims he 13 Million Computational Minutes per Day using Perf and flame graphs to solve performance problems in a Java EE application at Netflix.

The Node programmer

I am a happy Node.Js programmer, I have no problems with performance, garbage collector pauses, just nothing stops my software to scale. But if I would ever need it, I could also use Perf to generate flame graphs for Node Applications. Just using *–perf-basic-prof* option.

The Android programmer

I am an Android programmer. My performance problems are always very hard. I only need to recompile most of my system and use magic to find or build debug symbols for everything, so Perf can give me a useful output. No thanks, I'd prefer to eat this cake.

When something looks slow: USE method

Split your system into resources, and look for...

- Usage;
- Saturation;
- Error.

When something looks slow: TSA method

- It records Thread states - that points at a direction to look for slowdowns.
- or just use valgrind drd!

Blame-Someone-Else aka Ionut Antimethod

- Find a component that you are not responsible for;
- the issue is clearly with that component;
- redirect the issue to the othe team.

Streetlight Anti-Method

- Find random tools on the internet;
- or use only tools that you know;
- no hypothesis;
- no real interpretation;
- try to spot something.

Drunk man Anti-Method

Base

- Random modification until the problem goes away;
- or you get fired.

Drunk man Anti-Method advanced

Advanced

- Measure something;
- modificate some random component in a direction;
- repeat measurement;
- modificate some random component in the opposite direction;
- repeat measurement;
- take the better solution;
- repeat.

Thanks...

- to Fabio Collini as *The Senior Java Programmer*;
- to Federico Bertolucci as *The Happy NodeJs Programmer*;
- to Francesco De Felice as *The Greedy Android Programmer*.

References

- Systems Performance: Enterprise and the Cloud;
- Saving 13 million of computational minutes per day;
- Brendan Gregg's site;
- Perf's Wiki.