GRANULATE

Low Overhead Python Application Profiling Using eBPF

Yonatan Goldschmidt
Principal Engineer and Research Team Lead @ Granulate

Connect on LinkedIn | GitHub: https://github.com/Jongy

Meet Yonatan Goldschmidt

- About me:
  - Six years of service as R&D specialist & team lead
  - Likes everything about computers and software
  - Today, a team lead on Granulate's Performance Research Team

- About Granulate:

  Enables companies to optimize their workloads, improve performance and leverage that to reduce compute costs

Me in Italy, during a sommelier course!

GRANULATE

Why profiling is amazing

It’s a win-win… win.

- Easy to apply and use
- Gain visibility on code hot spots, bottlenecks
- Identify impactful performance improvement opportunities

Deterministic Python profilers

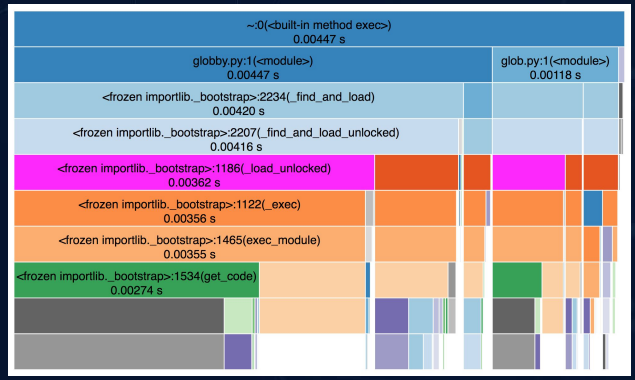
For versatile profiling needs

- **cProfile:** Deterministic profiling of Python programs, generate Python execution reports in the CLI via the pstats module, or use visualization tools like snakeviz
- Many other Python profilers: Pyinstrument, Yappi and more.

```
         409515 function calls (405541 primitive calls) in 3.421 seconds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
       15    0.397    0.026    0.399    0.027 search_index.py:76(not_operation)
        1    0.392    0.392    1.186    1.186 /System/Library/Frameworks/Python
        1    0.228    0.228    0.532    0.532 dictionary.py:53(from_json)
        1    0.193    0.193    0.193    0.193 /System/Library/Frameworks/Python
     1949    0.169    0.000    0.169    0.000 {method 'seek' of 'file' objects}
        1    0.129    0.129    0.129    0.129 /Library/Python/2.7/site-packages
        1    0.116    0.116    0.119    0.119 /System/Library/Frameworks/Python
        1    0.110    0.110    0.112    0.112 /System/Library/Frameworks/Python
        1    0.085    0.085    0.291    0.291 /System/Library/Frameworks/Python
    35735    0.074    0.000    0.074    0.000 dictionary.py:67(__init__)
        1    0.060    0.060    0.060    0.060 /System/Library/Frameworks/Python
        1    0.059    0.059    1.209    1.209 search_index.py:9(search)
        1    0.059    0.059    0.089    0.089 /Library/Python/2.7/site-packages
        1    0.048    0.048    0.050    0.050 /System/Library/Frameworks/Python
        1    0.045    0.045    0.079    0.079 /Library/Python/2.7/site-packages
        1    0.040    0.040    0.219    0.219 /System/Library/Frameworks/Python
        1    0.038    0.038    0.040    0.040 /Library/Python/2.7/site-packages
```



GRANULATE

Deterministic profiler limitations

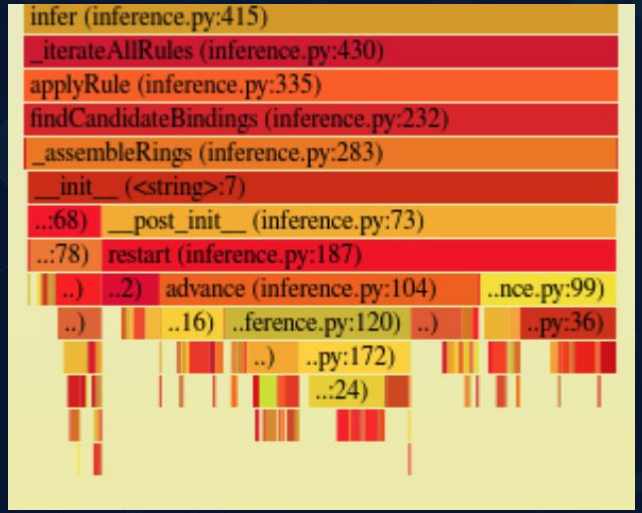
- Intrusive by design
- Require code changes in many cases (or deployment changes)
- Prone to high overhead (2x slower code)

- Not for production use!

Statistical Python profilers

Lower overhead, and not necessarily intrusive

- **py-spy:** Sampling profiler for Python. Not intrusive and incurs zero overhead on the application (but has **overhead** on the system). Production safe.
- Other options: vmprof, scalene (**intrusive**)



Statistical profiler limitations

- Can be made non-intrusive
  - No code changes!
- Low overhead on the system and negligible overhead on the application (if any)
- Suitable for production use
- When is deterministic profiling preferred?

eBPF changed the profiling game

About eBPF

- **1992**: Berkeley Packet Filter released, obscure kernel technology

- **2022**: Now known as eBPF and a household name, in-kernel execution environment:
  - User-defined programs
  - Limited, secure kernel access

```
# tcpdump -d host 127.0.0.1 and port 80
(000) ldh      [12]
(001) jeq      #0x800           jt 2   jf 18
(002) ld       [26]
(003) jeq      #0x7f000001      jt 6   jf 4
(004) ld       [30]
(005) jeq      #0x7f000001      jt 6   jf 18
(006) ldb      [23]
(007) jeq      #0x84            jt 10  jf 8
(008) jeq      #0x6             jt 10  jf 9
(009) jeq      #0x11            jt 10  jf 18
(010) ldh      [20]
(011) jset     #0x1fff          jt 18  jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
(014) jeq      #0x50            jt 17  jf 15
(015) ldh      [x + 16]
(016) jeq      #0x50            jt 17  jf 18
(017) ret      #262144
(018) ret      #0
```

https://www.usenix.org/system/files/lisa21_slides_gregg_bpf.pdf

eBPF – low overhead, system-wide tracing

```
PID    COMM               FD ERR PATH
490346 ThreadPoolForeg   118   0 /home/jong/.cache/pgadmin4/Default/Cache/df089dba81afe576_0
490396 python3            11   0 /home/jong/.pgadmin/pgadmin4.db
5581   Chrome_IOThread   175   0 /dev/shm/.com.google.Chrome.QUilLb
5581   Chrome_IOThread   186   0 /dev/shm/.com.google.Chrome.QUilLb
490396 python3            14   0 /home/jong/.pgadmin/sessions/6be80f34-9d9c-4268-8877-2277d86ecc05
490312 Chrome_IOThread   224   0 /dev/shm/.io.nwjs.GNJOVj
490312 Chrome_IOThread   226   0 /dev/shm/.io.nwjs.GNJOVj
490346 ThreadPoolForeg   118   0 /home/jong/.config/pgadmin4/Default/Cookies-journal
490346 ThreadPoolForeg   120   0 /home/jong/.config/pgadmin4/Default
490346 ThreadPoolForeg   118   0 /home/jong/.cache/pgadmin4/Default/Cache/df089dba81afe576_0
490396 python3            11   0 /home/jong/.pgadmin/pgadmin4.db
490346 ThreadPoolForeg   123   0 /home/jong/.cache/pgadmin4/Default/Cache/06a5699cbd40e70f_0
490396 python3            15   0 /home/jong/.pgadmin/pgadmin4.db
490363 exe               132   0 /home/jong/.local/share/pgadmin/pgadmin4.1642091395220.log
490396 python3            16   0 /home/jong/.pgadmin/sessions/6be80f34-9d9c-4268-8877-2277d86ecc05
490312 Chrome_IOThread   224   0 /dev/shm/.io.nwjs.3srpzk
490312 Chrome_IOThread   226   0 /dev/shm/.io.nwjs.3srpzk
490312 Chrome_IOThread   224   0 /dev/shm/.io.nwjs.L7utri
490312 Chrome_IOThread   226   0 /dev/shm/.io.nwjs.L7utri
```

```
18  BEGIN
19  {
20          printf("Tracing open syscalls... Hit Ctrl-C to end.\n");
21          printf("%-6s %-16s %4s %3s %s\n", "PID", "COMM", "FD", "ERR", "PATH");
22  }
23
24  tracepoint:syscalls:sys_enter_open,
25  tracepoint:syscalls:sys_enter_openat
26  {
27          @filename[tid] = args->filename;
28  }
29
30  tracepoint:syscalls:sys_exit_open,
31  tracepoint:syscalls:sys_exit_openat
32  /@filename[tid]/
33  {
34          $ret = args->ret;
35          $fd = $ret > 0 ? $ret : -1;
36          $errno = $ret > 0 ? 0 : - $ret;
37
38          printf("%-6d %-16s %4d %3d %s\n", pid, comm, $fd, $errno,
39              str(@filename[tid]));
40          delete(@filename[tid]);
41  }
42
43  END
44  {
45          clear(@filename);
46  }
```

https://github.com/iovisor/bpftrace/blob/master/tools/opensnoop.bt

GRANULATE

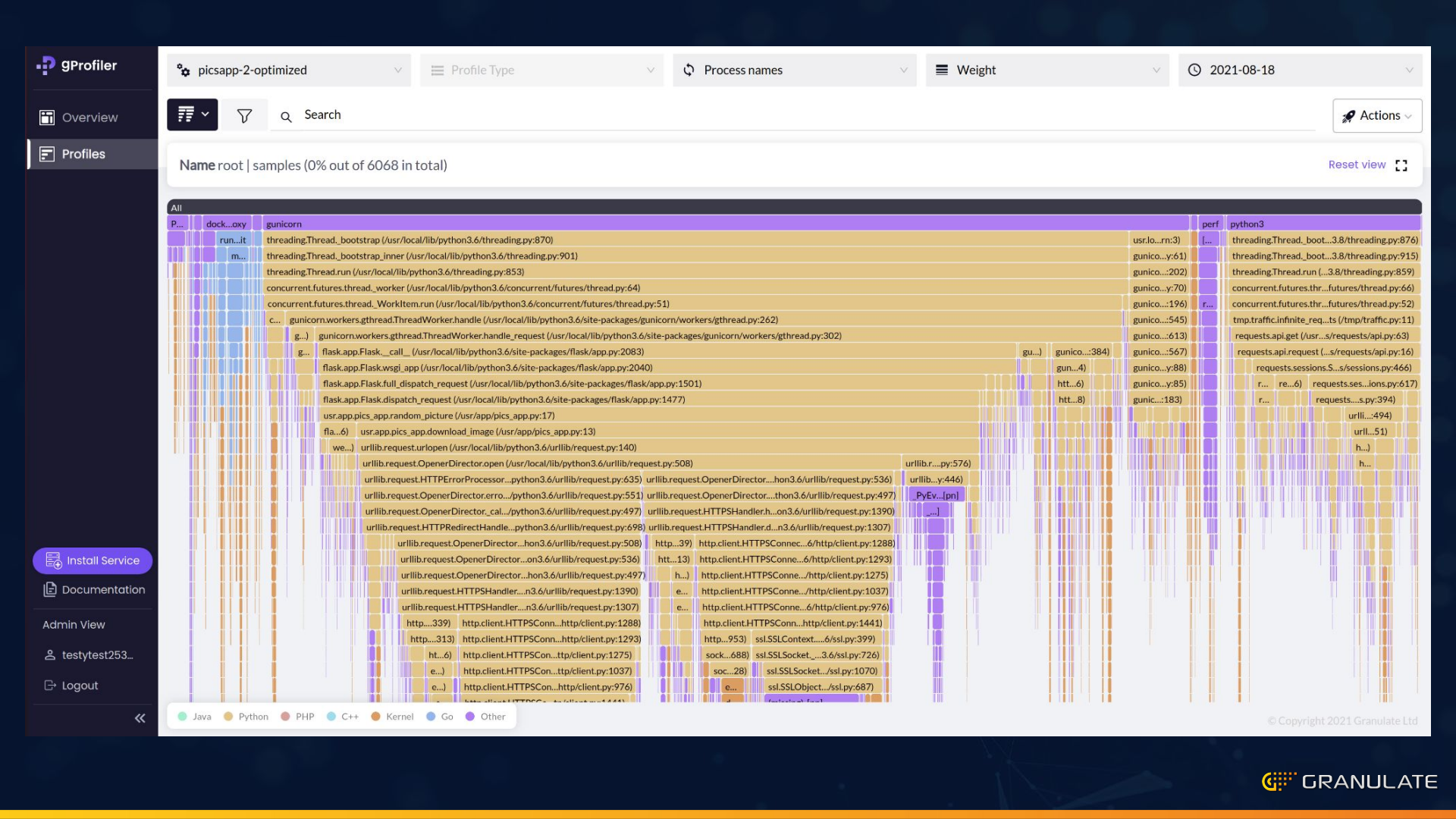
eBPF: A new type of software

	Execution model	User defined	Compil-ation	Security	Failure mode	Resource access
User	task	yes	any	user based	abort	syscall, fault
Kernel	task	no	static	none	panic	direct
BPF	event	yes	JIT, CO-RE	verified, JIT	error message	restricted helpers

https://www.usenix.org/system/files/lisa21_slides_gregg_bpf.pdf

Python profiling in eBPF

- Let's use eBPF to write a better profiler!
- The story of PyPerf - an open-source profiling tool for Python
  - py-spy is good, but we needed something more capable
  - Found PoC profilers PyPerf and rbperf
  - Took PyPerf miles ahead by adding many new features


gProfiler
Overview
Profiles
Install Service
Documentation
Admin View
testytest253...
Logout
picsapp-2-optimized
Profile Type
Process names
Weight
2021-08-18
Search
Actions
Name root | samples (0% out of 6068 in total)
Reset view
All
gunicorn
perf
python3
Java
Python
PHP
C++
Kernel
Go
Other
© Copyright 2021 Granulate Ltd

GRANULATE

Native code matters!

```
import zlib

def func(n):
    buf = ""
    for i in range(n):
        buf += chr(i % 128)
    zlib.compress(buf.encode())

while True: func(999999)
```

Native code matters!

Observing the native side of Python helps improve performance in unforeseen ways

% operator

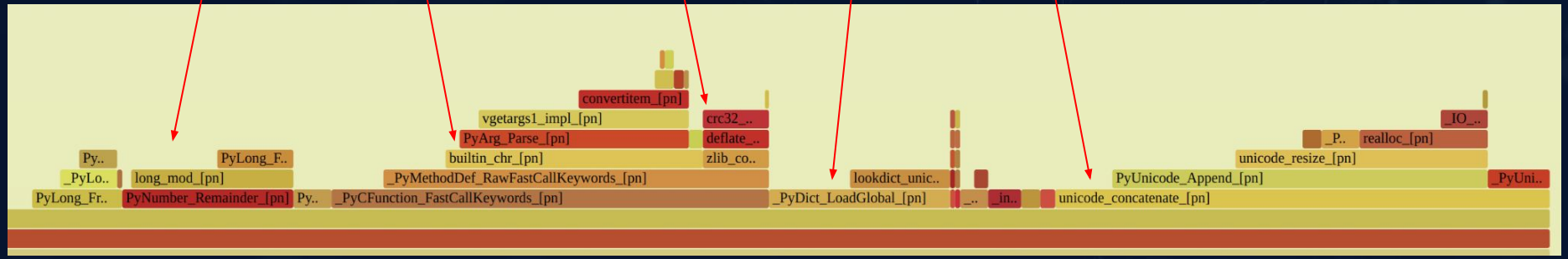
chr()

compression

Loading global variables, e.g “chr”

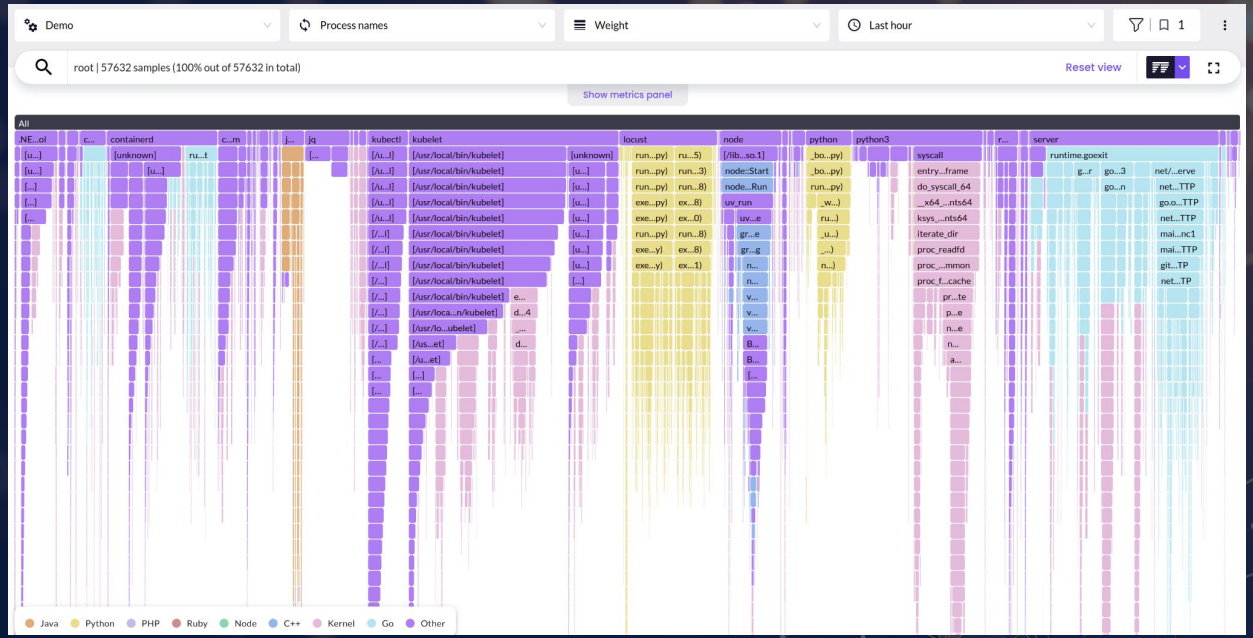
str +=

1. str +=
2. chr()
3. %
4. compress



GRANULATE

PyPerf is a part of gProfiler:



a free & open-source profiler

Thank You!

Feel free to DM here or connect on LinkedIn and GitHub (/jongy)

To start free profiling, visit: **gprofiler.io**

GRANULATE

Q&A