Python performance monitoring

Overview about different approaches

Holger Hees
Zeuthen, 22.05.2023

HELMHOLTZ

DESY.

About me

Holger Hees

- 25 years experiences in different it environments
- Fullstack, Mobile Developer & Unix Administrator
- 15 years freelancer & 10 years employee
  (Developer & lead architect - Visual Meta / Idealo)
- Since October 2021 in DESY Zeuthen

Agenda

01 Why is profiling important

02 Time messurement

02 Module: timeit

03 Module: cProfile

04 Other modules

05 Top 5 tips

Why is profiling important

1. Helps improving run time, performance and scalability
2. Better usage of system resources
3. Simplifying code and better readability
4. Understanding how your logic works internally
5. Helps to make right decision, which module to use
6. Long term learning how to code better

Time measurement

- Simplest way of measuring
- Not very accurate and meaningful
- Can only be a first step

Time measurement – Example 1

```
import time

start = time.time()

longstr = ""
for x in range(1000000):
  longstr += str(x)

end = time.time()

print("Runtime:{}".format( end - start))
```

Time measurement – Example 2

```
import time

start = time.time()

longstr = "".join([str(x) for x in range(1000000)])

end = time.time()

print("Runtime:{}".format( end - start))
```

Module: timeit

- More accurate measuring
- Eliminates irrelevant things like runtime startup, parsing, compiling/interpreting
- Faster comparison of different implementations

Module: timeit – Example 1 – String appending

```
python3 -m timeit 'longstr="" 'for x in range(1000000): longstr += str(x)'

python3 -m timeit ''.join([str(x) for x in range(1000000)])'
```

Module: timeit – Example 2- Square root

```
python3 -m timeit '[x**0.5 for x in range(1000)]'

python3 -m timeit -s 'from math import sqrt' '[sqrt(x) for x in range(1000)]'

python3 -m timeit -s 'from numpy import sqrt' '[sqrt(x) for x in range(1000)]'
```

"-s" option used to execute setup code, running only once

Module: timeit – Example 3 – Numpy optimized

```
python3 -m timeit -s 'import numpy as np; x=np.array(range(1000))' 'np.sqrt(x)'
```

Previous numpy was running slow, because it was not an array and numpy optimized for arrays

Module: timeit – Example 4 – Embedded

```
import timeit

measurements = timeit.repeat('[x**0.5 for x in range(1000)]', number=10000)

print(measurements)
```

Module: cProfile

- Profiling shows more details then benchmarking
- Insights about used function and how often
- Can give hints what to refactor

Module: cProfile – Example 1

```
def dict_lookup(map, key):
    for _key in map.keys():
        if key == _key:
            return True
    return False

print("fill dictionary")
map = {}
for i in range(1000000):
    map[i] = True

print("lookup dictionary")
for i in range(10000):
    dict_lookup(map, i)
```

python3 -m "cProfile" **-s** ncalls example_cprofile_1.py

Module: cProfile – Example 2

```
def dict_lookup(values, value):
    for _value in values:
        if value == _value:
            return True
    return False

print("fill dictionary")
map = {}
for i in range(1000000):
    map[i] = True

print("lookup dictionary")
values = map.keys()
for i in range(10000):
    dict_lookup(values, i)
```

python3 -m "cProfile" **-s** ncalls example_cprofile_2.py

Module: cProfile – Example 3

```
print("fill dictionary")
map = {}
for i in range(1000000):
    map[i] = True

print("lookup dictionary")
values = map.keys()
for i in range(10000):
    if i in map:
        pass
```

python3 -m "cProfile" **-s** ncalls example_cprofile_3.py

Module: cProfile – Example 4 – Embedded

```
import cProfile as profile
import pstats

def dict_lookup(map, key):
    for _key in map.keys():
        if key == _key:
            return True
    return False

print("fill dictionary")
map = {}
for i in range(1000000):
    map[i] = True


prof = profile.Profile()
prof.enable()

print("lookup dictionary")
for i in range(10000):
    dict_lookup(map, i)

prof.disable()
stats = pstats.Stats(prof).strip_dirs().sort_stats("ncalls")
stats.print_stats(10) # top 10 rows
```

Module: cProfile – Different sort options

Sort string	Meaning
calls	Call count
cumulative	Cumulative time
cumtime	Cumulative time
file	File name
filename	File name
module	File name
ncalls	Call count
pcalls	Primitive call count
line	Line number
name	Function name
nfl	Name/file/line
stdname	Standard name
time	Internal time
tottime	Internal time

Other modules

- timeit module: https://docs.python.org/3/library/timeit.html
- cProfile module and pstats module: https://docs.python.org/3/library/profile.html
- Python Call Graph: https://pycallgraph.readthedocs.io/en/master/
- Line Profiler: https://github.com/pyutils/line_profiler

Top 5 tips

1. Do not import root module
   python3 -m timeit -s 'import math' 'math.**sqrt**(100)'
   vs
   python3 -m timeit -s 'from math import sqrt' '**sqrt**(100)'

2. Avoid Using Dot / Dot Chaining
   python3 -m timeit -s 'my_list = [1, 2, 3]' 'my_list.**append**(4)' 'my_list.**remove**(4)'
   vs
   python3 -m timeit -s 'my_list = [1, 2, 3]; append = my_list.**append**; remove = my_list.**remove**'
   'append(4)' 'remove(4)'

3. Do not use + to join string arrays

4. Do Not Use Temp Variable for Value Exchange
   python3 -m timeit -s 'a = 1; b = 2' 'temp = a' 'a = b' 'b = temp'
   vs
   python3 -m timeit -s 'a = 1; b = 2' 'a, b = b, a'

Top 5 tips

5. Do Not Use While-Loop If We Can Use For-Loop
   python3 -m timeit 'i = 0' '**while** i < 10: i += 1'
   vs
   python3 -m timeit '**for** i in **range**(10): pass'

Thank you