

Pikos: A Real-Time Python Profiling Framework

Simon Jagoe

Enthought Ltd

www.enthought.com

Why Was Pikos Created?

- Working with large data sets requires careful optimization and design for memory usage
- There can be errors in complex algorithms that are not immediately obvious
 - Memory Leak
 - Large temporary allocation
- How do you locate these problems in large and unfamiliar code bases?

Existing Tools

- `Meliae` **and** `Heapy`
 - Both provide tools for detailed memory analysis
 - Slows down with large memory use
 - Knowing where to look (i.e. starting with `Pikos`) can help with this analysis
- `memory_profiler` is a line-by-line profiler for memory usage
 - Only discovered after `Pikos` was started
 - `Pikos` can wrap it to make the functionality available

Beginnings of Pikos

- Find memory leaks and large allocation in unknown codebase
- Use deterministic monitoring (trace execution) as periodic monitoring will not find all cases
- Simply record useful values for later analysis
- Manual analysis of plotted data required to find problem areas

Example Memory Profile

```
from pikos.api import monitor
from pikos.recorders.text_stream_recorder import TextStreamRecorder
from pikos.monitors.function_memory_monitor import FunctionMemoryMonitor, \
    FunctionMemoryRecordFormater

memory = FunctionMemoryMonitor(
    TextStreamRecorder(sys.stdout, auto_flush=True,
                      formater=FunctionMemoryRecordFormater()))

@monitor(memory)
def legacy(size):
    b = np.mat(np.random.random(size).T)
    # very bad example that makes copies of numpy arrays when converting them
    # to matrix
    a = np.matrix(np.random.random(size))
    final = a * b
    return final.I

@monitor(memory)
def fixed(size):
    # more appropriate way using a numpy.mat
    b = np.mat(np.random.random(size).T)
    a = np.mat(np.random.random(size))
    final = a * b
    return final.I
```

Example Profile Output

index	type	function	RSS	VMS	lineNo	filename

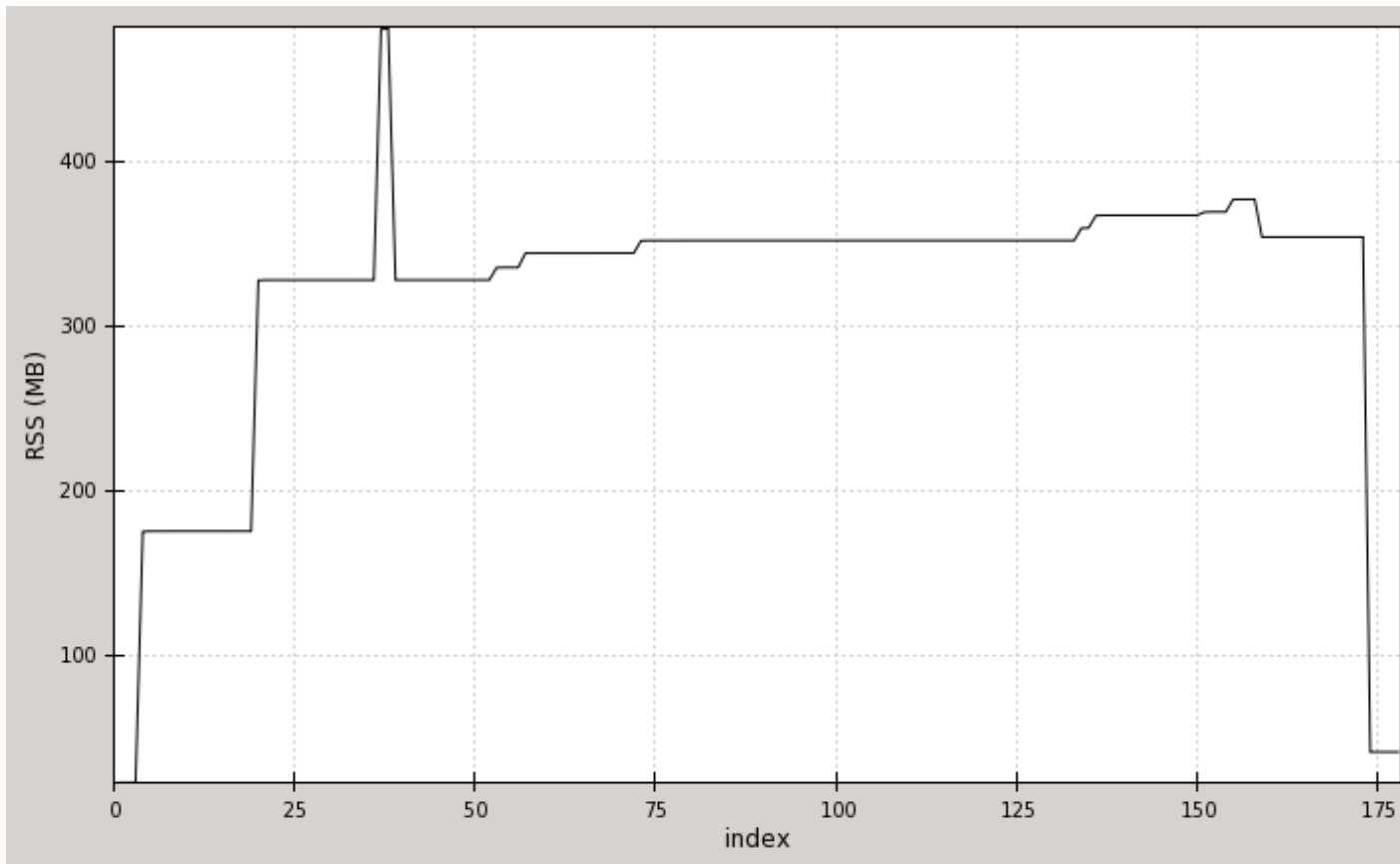
...						
18	return	asmatrix	181813248	307998720	85	<numpy>
19	c_call	random_sample	181813248	307998720	33	spike.py
20	c_return	random_sample	341577728	468000768	33	spike.py
21	call	__new__	341815296	468000768	233	<numpy>
...						
36	return	__array_finalize__	341827584	628002816	284	<numpy>
37	c_return	copy	501592064	628002816	250	<numpy>
38	return	__new__	501833728	628002816	250	<numpy>
39	call	__mul__	341831680	468000768	327	<numpy>
...						

We can see that line 33 in `spike.py` is the problem:

```
>>> a = np.matrix(np.random.random(size))
```

This copies the array to create a matrix, the temporary created by `np.random` creates a memory spike

Plotting The Profile



Generalizing Pikos

- Introduce general concepts to enable extensibility
 - Monitor: Top level machinery
 - Logger: Extensible concept to produce recordable data
 - Able to filter to produce limited set of records
 - Recorder: Extensible concept to format, serialize or process the data in some way
- These components can plug together in flexible configurations

Generalizing Pikos

- Logging functions
 - Trace program flow
 - Profile memory usage
 - Profile speed
- Recording functions
 - Write to Stdout
 - Write to disk
 - Send to ZeroMQ socket

Real-Time Profile Analysis

- If you don't know where to look for the problem, it can be useful to monitor the results while an application is running
 - Capture profile results “live”
- Analyse data while process under profiling is still running and producing more results
- Drill down into more detail about each data point

Future development

- Improve API / profiling user interface
- Improve support for existing tools (e.g. cProfile) to enable real-time analysis
- Add support for more tools (line_profiler, yappi, ...)
- Improve real-time analysis tools
 - Profile multiple processes from a single location
 - Display more information about each data point

Availability and Licence

- Available under the 3-clause BSD licence
- Now available on Github.com
 - <https://github.com/sjagoe/pikos>