Resource Monitor

Release 2.3.1

Dec 19, 2021

Contents:

A simple cross-platform system resource monitor.

The **monitor** utility collects telemetry on system resources. Metrics are printed to *stdout* at regular intervals. Resources are organized under "cpu" or "gpu" *device* groups. All resources share some global options.

It is easily installable, has an intuitive interface, and is cross-platform. Python 3.7 or higher is required, however.

```
pip install resource-monitor
```

```
$ monitor
usage: monitor [-h] [-v] <device> <resource> [<args>...]
A simple cross-platform system resource monitor.
```

```
$ monitor cpu memory --actual --human-readable
2020-01-30 15:24:51.573 desktop.local monitor.cpu.memory 8.23G
2020-01-30 15:24:52.578 desktop.local monitor.cpu.memory 8.23G
...
```

CHAPTER 1

Getting Started

1.1 Installation

monitor is built on Python 3.7+ and can be installed using Pip

```
pip install resource-monitor
```

For use with GPUs you will need to have the associated command-line tools installed. Currently, Nvidia (using `nvidia-smi`) and AMD (using `rocm-smi`) are supported.

1.2 Basic Usage

The usage statement for each resource type is outlined below. The `--help` flag is provided at each level. `monitor --help` will show the device groups (i.e., cpu/gpu). `monitor <device> --help` will show available resources for that group.

For complete information including examples at the command-line, the manual page can be accessed with `man monitor`.

1.2.1 CPU Percent

```
usage: monitor cpu percent [-h] [--all-cores] [-s SECONDS] [--csv [--no-header]]
Monitor CPU percent utilization.

options:
-t, --total                   Show values for total cpu usage (default).
-a, --all-cores               Show values for individual cores.
-s, --sample-rate  SECONDS    Time between samples (default: 1).
    --plain                   Print messages in syslog format (default).
    --csv                     Print messages in CSV format.
```

(continues on next page)

(continued from previous page)

```
    --no-header                Suppress printing header in CSV mode.
-h, --help                     Show this message and exit.
```

1.2.2 CPU Memory

```
usage: monitor cpu memory [-h] [-s SECONDS] [--actual [--human-readable]] [--csv [--
↪no-header]]
Monitor CPU memory utilization.

options:
    --percent                   Report value as a percentage (default).
    --actual                    Report value as total bytes.
-s, --sample-rate     SECONDS   Time between samples (default: 1).
-H, --human-readable            Human readable values (e.g., "8.2G").
    --plain                     Print messages in syslog format (default).
    --csv                       Print messages in CSV format.
    --no-header                 Suppress printing header in CSV mode.
-h, --help                      Show this message and exit.
```

1.2.3 GPU Percent

```
usage: monitor gpu percent [-h] [-s SECONDS] [--csv [--no-header]]
Monitor GPU percent utilization.

options:
-s, --sample-rate  SECONDS     Time between samples (default: 1).
    --plain                    Print messages in syslog format (default).
    --csv                      Print messages in CSV format.
    --no-header                Suppress printing header in CSV mode.
-h, --help                     Show this message and exit.
```

1.2.4 GPU Memory

```
usage: monitor gpu memory [-h] [-s SECONDS] [--csv [--no-header]]
Monitor GPU memory utilization.

options:
-s, --sample-rate  SECONDS     Time between samples (default: 1).
    --plain                    Print messages in syslog format (default).
    --csv                      Print messages in CSV format.
    --no-header                Suppress printing header in CSV mode.
-h, --help                     Show this message and exit.
```

1.2.5 GPU Power

```
usage: monitor gpu power [-h] [-s SECONDS] [--csv [--no-header]]
Monitor GPU power consumption (in Watts).

```

(continues on next page)

(continued from previous page)

```
options:
-s, --sample-rate  SECONDS     Time between samples (default: 1).
    --plain                    Print messages in syslog format (default).
    --csv                      Print messages in CSV format.
    --no-header                Suppress printing header in CSV mode.
-h, --help                     Show this message and exit.
```

1.2.6 GPU Temperature

```
usage: monitor gpu temp [-h] [-s SECONDS] [--csv [--no-header]]
Monitor GPU temperature (Celsius).

options:
-s, --sample-rate  SECONDS     Time between samples (default: 1).
    --plain                    Print messages in syslog format (default).
    --csv                      Print messages in CSV format.
    --no-header                Suppress printing header in CSV mode.
-h, --help                     Show this message and exit.
```

CHAPTER 2

Examples

2.1 Simple

Monitor CPU on a per-core basis on a 10 second interval.

```
$ monitor cpu percent --all-cores --sample-rate 10

2020-01-30 12:55:55.521 some-hostname.local monitor.cpu.percent [0] 7.5
2020-01-30 12:55:55.522 some-hostname.local monitor.cpu.percent [1] 2.3
2020-01-30 12:55:55.522 some-hostname.local monitor.cpu.percent [2] 8.5
2020-01-30 12:55:55.522 some-hostname.local monitor.cpu.percent [3] 0.8
2020-01-30 12:56:05.525 some-hostname.local monitor.cpu.percent [0] 8.5
2020-01-30 12:56:05.525 some-hostname.local monitor.cpu.percent [1] 2.3
2020-01-30 12:56:05.525 some-hostname.local monitor.cpu.percent [2] 8.6
2020-01-30 12:56:05.526 some-hostname.local monitor.cpu.percent [3] 0.8
...
```

Monitor CPU memory in actual bytes used and output in CSV format.

```
$ monitor cpu memory --actual --csv

timestamp,hostname,resource,memory_used
2020-01-30 12:58:21.476,some-hostname.local,cpu.memory,9707892736
2020-01-30 12:58:22.479,some-hostname.local,cpu.memory,9706946560
2020-01-30 12:58:23.480,some-hostname.local,cpu.memory,9724190720
2020-01-30 12:58:24.484,some-hostname.local,cpu.memory,9726636032
...
```

Monitor GPU utilization on a per-GPU basis on a 10 second interval and log to a file.

```
$ monitor gpu percent --sample-rate 10 >gpu.percent.log
$ head -8 gpu.percent.log

2020-01-30 13:04:22.938 node-001.cluster monitor.gpu.percent [0] 79.0
```

(continues on next page)

(continued from previous page)

```
2020-01-30 13:04:22.938 node-001.cluster monitor.gpu.percent [1] 0.0
2020-01-30 13:04:22.938 node-001.cluster monitor.gpu.percent [2] 0.0
2020-01-30 13:04:22.938 node-001.cluster monitor.gpu.percent [3] 87.0
2020-01-30 13:04:33.196 node-001.cluster monitor.gpu.percent [0] 72.0
2020-01-30 13:04:33.196 node-001.cluster monitor.gpu.percent [1] 0.0
2020-01-30 13:04:33.196 node-001.cluster monitor.gpu.percent [2] 0.0
2020-01-30 13:04:33.196 node-001.cluster monitor.gpu.percent [3] 90.0
```

2.2 Distributed

Monitor core utilization within a distributed computing context.

```
$ mpiexec -machinefile <(sort -u $NODEFILE) \
      monitor cpu percent --all-cores

2020-01-30 13:17:50.980 node-001.cluster monitor.cpu.percent [0] 100.0
2020-01-30 13:17:50.980 node-001.cluster monitor.cpu.percent [1] 1.0
...
2020-01-30 13:17:51.208 node-002.cluster monitor.cpu.percent [0] 100.0
2020-01-30 13:17:51.208 node-002.cluster monitor.cpu.percent [1] 100.0
...
2020-01-30 13:17:51.294 node-003.cluster monitor.cpu.percent [0] 100.0
2020-01-30 13:17:51.295 node-003.cluster monitor.cpu.percent [1] 100.0
...
2020-01-30 13:17:51.319 node-004.cluster monitor.cpu.percent [0] 100.0
2020-01-30 13:17:51.320 node-004.cluster monitor.cpu.percent [1] 100.0
...
```

Monitor percent main memory utilization within a distributed computing context, as a background task, and in CSV format. Basically, the produced headers will arrive from each node, suppress them with `--no-header`. Create a single header by just slicing it off the top of an initial invocation. Collect the process ID so the task can be interrupted at then end of your job.

```
$ monitor cpu memory --csv | head -1 >log/resource.mem.csv
$ mpiexec -machinefile <(sort -u $NODEFILE) \
      monitor cpu memory --csv --no-header >>log/resource.mem.csv &
$ MEM_PID=$!

...

$ kill -s INT $MEM_PID
```

CHAPTER 3

Recommendations

If collecting data for benchmarking/profiling/scaling purposes (regarding CPU/memory in particular), it may be appropriate to also collect data in the absence of your application as a null-scenario. This can approximate a “background noise” that can modeled and subtracted.

CHAPTER 4

Caveats

- **monitor** merely samples data made available by other libraries or command-line tools. In the case of CPU resources the **psutil** library in Python. In the case of GPU resources the output of the `nvidia-smi` tool. Metrics are reported with regard to the whole system, NOT JUST YOUR APPLICATION.
- For GPU resources, currently only `nvidia-smi` and `rocm-smi` are supported. Additional GPU providers could be supported in the future though.
- Sampling more frequently than 1 second is an error. The CPU percent utilization is a time averaged metric subject to how frequently it is sampled.

CHAPTER 5

For Developers

5.1 Roadmap

- Explore additional resources (e.g., disk/filesystem, threads).

5.2 Contributing

Development of **monitor** happens on Github. Contributions are welcome in the form of suggestions for additional features, *Pull Requests* with new features or bug fixes, etc. If you find bugs or have questions, open an *Issue*.

5.3 Guide

The **monitor** command-line interface is written in Python and uses the **psutil** library. Additional resources may be possible to collect but may not necessarily be easily made cross-platform.

The GPU functionality is simply a wrapper external tools, e.g., `nvidia-smi`. In the library, a fully generalized notion of an *ExternalMetric* interface is provided. In principle, anything that could conceivably be invoked on the command-line need only have a parser method implemented.

For example:

```
class OpenFiles(ExternalMetric):
    """Report the number of open files (psutil already provides this)."""

    _cmd = 'lsof -u `whoami`'

    @classmethod
    def parse_text(cls, block: str) -> Dict[str, int]:
        """Count lines in the output."""
        return {'count': len(block.strip().split('\n'))}
```