prmon: Process Monitor

Serhan Mete (Argonne) and Graeme A Stewart (CERN)

Grid Deployment Board Meeting
September 9, 2020

What is Process Monitor (*prmon*)?

- "... a small stand alone program that can monitor the resource consumption of a process and its children."
  - An open source HSF project, completely application agnostic and self-contained
    - The only external library dependency is nlohmann/json (and only for the build)
  - Tracks (process-level) **CPU/GPU**, **memory**, **disk I/O**, and (device-level) **network I/O** usage
    - Metrics are primarily collected from **ProcFS** (except for GPU which comes from *nvidia-smi*)
      - Adding support for additional hardware is in the future plans
  - It produces two main outputs:
    - Time-series text file that contains the measurements at each capture
    - JSON file that contains averages and maxima along with some hardware information
  - It includes python-based software to visualize the time-series data

https://github.com/HSF/prmon

Building/Distributing/Using *prmon*...

- *prmon* has been used in ATLAS distributed computing for many years
  - Predecessor was named *MemoryMonitor*, which was the starting point for *prmon*
- There are two main build/deployment options:
  - Integrating *prmon* as an external software and building it from scratch
    - Primarily requires C++11, Cmake 3.3+, and nlohmann/json
    - For GPU support, needs *nvidia-smi* installed
  - Using statically built *prmon* (published for each release, approx. 1 MB)
  - In either case, the binaries can be (are) distributed via CVMFS
  - More information can be found at https://github.com/HSF/prmon#build-and-deployment
- There are two main ways to execute:
  - Attach to an existing process w/ PID : `prmon --pid PID`
  - Start the program with *prmon* : `prmon [prmon options] -- program [program options]`
  - More information can be found at https://github.com/HSF/prmon#running

Latest Release: v2.1.0 on Sep 8

Available options, monitors, and output formats…

```
$ prmon --help

prmon is a process monitor program that records runtime data from a process
and its children, writing time stamped values for resource consumption into
a logfile and a JSON summary format when the process exits.

Options:
[--pid, -p PID]           Monitored process ID
[--filename, -f FILE]     Filename for detailed stats (default prmon.txt)
[--json-summary, -j FILE] Filename for JSON summary (default prmon.json)
[--interval, -i TIME]     Seconds between samples (default 30)
[--suppress-hw-info, -s]  Disable hardware information (default false)
[--units, -u]             Add units information to JSON file (default false)
[--netdev, -n dev]        Network device to monitor (can be given
                          multiple times; default ALL devices)
[--] prog [arg] ...       Instead of monitoring a PID prmon will
                          execute the given program + args and
                          monitor this (must come after other
                          arguments)

One of --pid or a child program must be given (but not both)
```

```
Monitors available:
 - countmon : Monitor number of processes and threads
 - cpumon : Monitor cpu time used
 - iomon : Monitor input and output activity
 - memmon : Monitor memory usage
 - netmon : Monitor network activity (device level)
 - nvidiamon : Monitor NVIDIA GPU activity
 - wallmon : Monitor wallclock time
```

TXT

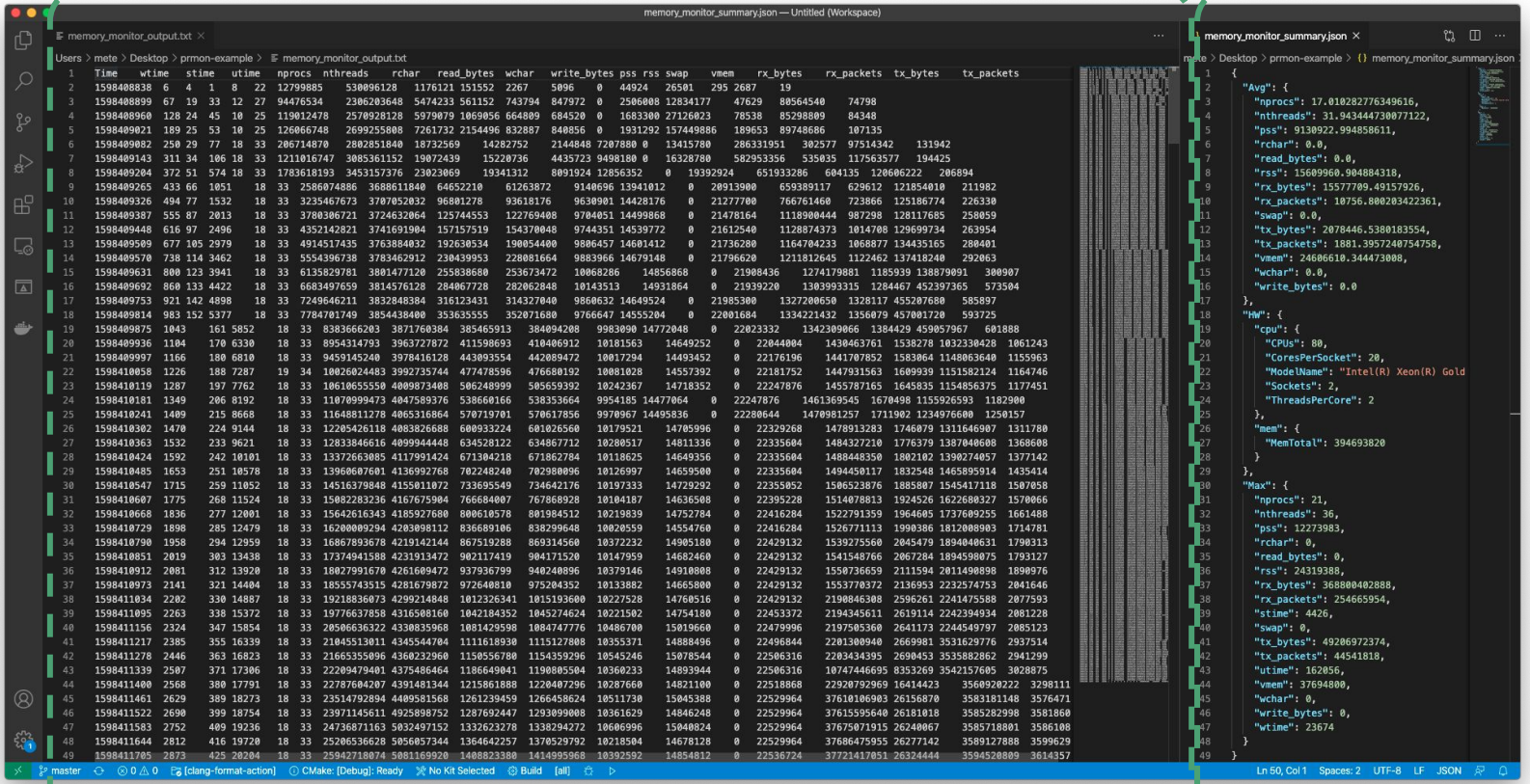
Time	metric-1	metric-2	...	metric-N
1599050513	...	...	...	...
1599050543	...	...	...	...
1599050573	...	...	...	...
...	...	...	...	...

JSON

```
{
  “Avg” : {
    “metric-1” : ...,
  },
  “HW” : {
    “cpu” : { ... },
    “mem” : { ... }
  },
  “Max” : {
    “metric-1” : ...,
  }
}
```

- The full set of metrics are provided in the back-up
- Currently all monitors are enabled by default
  - Some information is hardware dependent, e.g. GPU monitoring

Taking a quick look at example outputs...

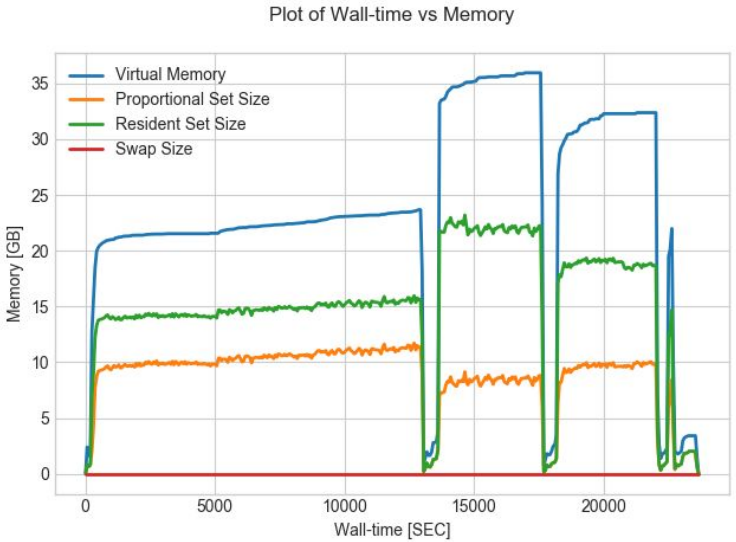


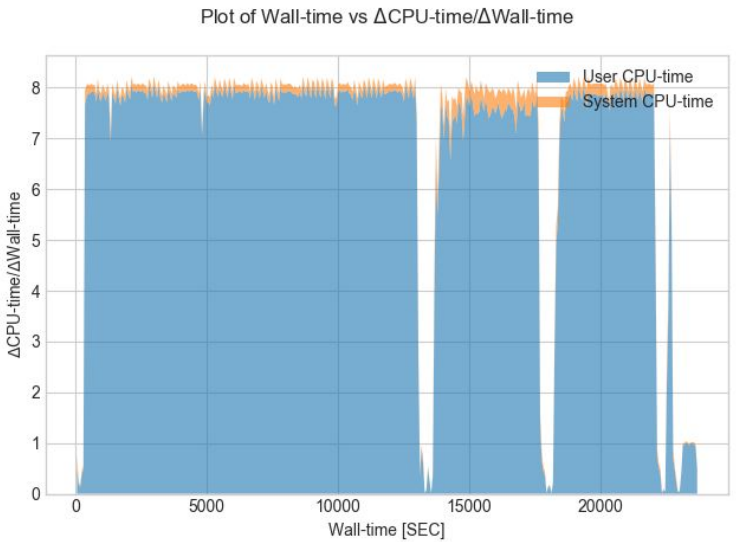
Time-series text file

Summary JSON file

Visualizing the results...

- *prmon_plot.py* can be used to visualize the time-series results, e.g.:
  - `prmon_plot.py --input prmon.txt --xvar wtime --yvar vmem,pss,rss,swap --yunit GB`
  - `prmon_plot.py --input prmon.txt --xvar wtime --yvar utime,stime --yunit SEC --diff --stacked`





  - More information can be found at https://github.com/HSF/prmon#visualisation

Publishing *prmon* results in the production system...

ATLAS MC Production Job

Executing time: 6h30min

Sampling time: 60sec

Output sizes:

- 57 KB for the text file
- 1 KB for the JSON file

BigPanDA Plots (per job):

- **CPU** (**GPU** if it exists)
- **Memory**
- **Disk I/O**
- Semi-interactive
- Published in BigPanda

Represented data metrics were collected using prmon program.

What is *prmon* used for exactly?

- Summary JSON file allows to:
  - Understand the application's overall resource usage **metrics** via **averages** and **maxima**
    - E.g. "*What is the maximum RSS/PSS usage of my application?*"
  - Have a summary of the hardware information for the particular node that runs the application
    - "*Is my application running on a specific processor?*"
- Detailed Text file allows to:
  - Have an in-depth time-series understanding of the application's resource usage
    - E.g. "*Does the memory usage increase significantly over time (i.e. memory leak)?*"
  - Cross-correlate possible application problems (errors/warnings) with resource usage
    - E.g. "Did my application performed poorly due to memory swapping? When did it start?"
- More importantly do all these in a general/generic/application-agnostic way

Experiences from the ATLAS distributed computing

- *prmon* is one of the essential tools for PanDA job brokering:
  - For each task, 10 scout jobs are released and their resource usages are analyzed
  - Remaining tasks are released *if and only if* scout jobs' resource usages fit the allowed envelope
  - *prmon* is **the main tool** for the memory measurements in this context
- *prmon* summary numbers are recorded in PanDA DB/Chicago analytics cluster
  - Also used for general job studies afterwards
- Pilot also runs an instance of *prmon* on the worker node to collect job data:
  - Sampling is configured to 60 seconds, i.e. `prmon --interval 60 [...]`
  - The outputs are stored in the job log tarball and stored on the SCRATCHDISK
    - Typically cleaned-up after 2-4 weeks
    - Overhead negligible compared to everything else
- Plots (a la slide 6) are produced by the BigPanda monitoring and published
  - Accessible to the users via the BigPanDA web-interface
  - The infrastructure is being extended to also handle GPU related metrics

Using *prmon* for detailed workflow analysis

- Riccardo Maganza started to work on analyzing large-scale *prmon* data
  - Fellow in CERN IT-SC-RD (w/ a background in data analytics) working with Markus Schulz
- The overarching goal of the work is to *understand the efficiency of the workflows and find potential improvements*
- The planned steps in the project are to:
  - Store the time-series results in a very compact format,
  - Classify efficiency differences of workloads at different sites and under changing conditions,
  - Detect anomalies in the time-series results linked to job failures etc.
- The effort began by focusing on ATLAS workflows but the long term goal is to extend this to the other experiments

Conclusions

- *prmon* is a light-weight, self-contained resource monitoring program
  - Developed primarily for the HEP workflows but completely application agnostic
- It has been used in the ATLAS distributed computing for more than a year now:
  - Used in the context of job brokering as well as general resource monitoring
  - Other experiments (CMS and LHCb) are also planning to adopt it
- We, as the current developers, would be more than happy to get feedback:
  - Bug reports, feature requests etc. are most welcome!
  - See our Contribution Guide
- Acknowledgements:
  - Last but not least, thanks to Johannes Elmsheuser, Andrea Sciabà and many others for their contributions, inputs, feedback and more...

Back-up

Full set of *prmon* metrics...

procs/threads	CPU	Memory	Disk I/O	Network I/O	Nvidia GPU
nprocs	utime	vmem	rchar	rx_bytes	ngpus
nthreads	stime	rss	read_bytes	rx_packets	gpusmpct
	wtime	pss	wchar	tx_bytes	gpumempct
		swap	write_bytes	tx_packets	gpufbmem

- All metrics are process-level except for the Network I/O, which are device-level
  - For more information please refer to https://man7.org/linux/man-pages//man5/procfs.5.html
- GPU metrics are collected via nvidia-smi
  - More information regarding the metrics can be found at https://github.com/HSF/prmon/pull/125