



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>

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

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

- 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

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

JSON

Taking a quick look at example outputs...

The image shows a terminal window with two panes. The left pane displays a table of system metrics, and the right pane displays a JSON representation of the same data.

Table Output (Left Pane):

Time	uptime	stime	utime	nprocs	nthreads	rchar	read_bytes	wchar	write_bytes	pss	rss	vmem	rx_packets	tx_bytes	tx_packets
1598408838	6	4	1	8	22	1279885	530096128	1176121	151552	2267	5996	0	44924	26581	19
1598408899	67	19	33	12	27	94476534	2386203648	5474233	561152	743794	847972	0	2566088	12831477	47629
1598408960	128	24	45	10	25	119012478	2570928128	5979079	1069056	664809	684520	0	1683300	21726023	78538
1598409021	189	25	53	10	25	126666748	2699255808	7261732	2154496	832887	840856	0	1931292	157449886	189653
1598409082	250	29	77	18	33	206714870	2882851840	18732569	14282752	2144848	7208780	0	3145780	286331951	10715342
1598409143	311	34	106	18	33	1211016747	3085361152	19072439	15220736	3445723	9498180	0	16328780	582953356	535835
1598409204	372	51	174	18	33	1783618193	3453157376	23023069	19341312	8091924	12856352	0	19392924	651933286	604135
1598409265	433	66	1051	18	33	2586074886	3688611840	64652210	61263872	9140696	13941012	0	20913900	659389117	629612
1598409326	494	77	1532	18	33	323547673	3787052832	96881278	93618176	9630981	14428176	0	21277700	763781460	723866
1598409387	555	87	2013	18	33	3780386731	3724632064	125745553	122769408	9784051	14499868	0	21478164	1118904444	907298
1598409448	616	97	2406	18	33	4332149261	3741631984	157137510	154370048	9743431	14539773	0	21612540	1120874733	1014708
1598409509	677	105	2979	18	33	4914517435	3763888480	192639534	190054400	9806457	14681142	0	21736200	1164704233	1068877
1598409570	738	114	3462	18	33	5554396738	3783462912	230439953	220881664	9883966	14679148	0	21796620	1211812645	1122462
1598409631	800	123	3941	18	33	6135929781	3801477120	25838680	253673472	10068286	14856868	0	21908436	1274179881	1185939
1598409692	860	133	4422	18	33	6683497659	3814576128	284067728	282062848	10143513	14931864	0	21939220	1293993315	1284467
1598409753	921	142	4898	18	33	7249646211	3832848384	316123431	314327840	9806032	14649524	0	21985300	1327200650	1328117
1598409814	983	152	5377	18	33	7784701749	3854438400	353635555	352071680	9766647	14555204	0	22001864	133421432	1356879
1598409875	1043	161	5852	18	33	8383666203	3871760384	385465913	384094208	9983090	14772048	0	22023332	1342309066	1384429
1598409936	1104	170	6330	18	33	8954314793	3963727872	411598693	410406912	10181563	14649252	0	22048044	143083614	1358278
1598409997	1166	180	6810	18	33	9459145240	3978416128	443093554	442808942	10017294	14493452	0	22176196	1441707852	1583064
1598410058	1226	188	7278	19	34	10026042483	3992735744	477478596	476688192	10081028	14557392	0	22181752	1447931563	1609394
1598410119	1287	197	7762	18	33	10610555550	4008973408	506248999	505659392	10242367	14718352	0	22247876	145578165	1658835
1598410181	1349	206	8192	18	33	11078999473	4047589376	538660166	538335364	9954185	14477064	0	22247876	1461369545	1670488
1598410241	1409	215	8668	18	33	11648811778	4085318864	570717801	578617866	9979067	14495834	0	22286464	1470881257	1711902
1598410302	1470	224	9144	18	33	12284611148	4083385664	6090933274	607865560	10170301	148789506	0	22329200	1476913283	1746070
1598410363	1532	233	9621	18	33	12834846616	4099944448	634528122	634867172	10280517	14811336	0	22335604	1484327210	1776379
1598410424	1592	242	10101	18	33	13372653085	4117991424	671384218	671862784	10118625	14649356	0	22335604	1488048350	1802102
1598410485	1653	251	10578	18	33	13968607601	4136992768	702248240	702988996	10126997	14655900	0	22335604	1494450117	1832548
1598410547	1715	259	11052	18	33	14516379848	415811072	733695549	734642176	10197333	14729292	0	22355852	1506523876	1885807
1598410607	1775	268	11524	18	33	15082283236	4167675904	766604007	767868928	10104187	14635080	0	22395228	1514078813	1924526
1598410668	1836	277	12001	18	33	15642616343	4185927680	800610578	801984512	10219839	14752784	0	22416284	1527791359	1964605
1598410729	1898	285	12479	18	33	16200009294	4203898112	836689106	838299648	10202559	14554760	0	22416284	1526771113	1990386
1598410790	1958	294	12959	18	33	16688789378	4219142144	867519288	869134560	10372322	14905180	0	22429132	1539275560	2045479
1598410851	2019	303	13438	18	33	17374941588	42310913472	902117419	904171520	10147959	14682460	0	22429132	1545487666	2067284
1598410912	2081	312	13920	18	33	17807991670	4261609472	937936799	940240896	10379146	14918008	0	22429132	1550736659	2111594
1598410973	2141	321	14484	18	33	18555743515	4281679872	972640810	975204352	10133882	14665800	0	22429132	1553770372	2136953
1598411034	2202	330	14887	18	33	19218836073	4299214848	1012326341	1015193600	10227528	14760516	0	22429132	1590846388	2261211
1598411095	2263	336	15372	18	33	19776637858	4315508160	1042184352	1045274624	10221502	14754180	0	22453372	2194345611	2619114
1598411156	2324	347	15854	18	33	20506636322	4330039568	1081428990	1084747776	10480780	15019660	0	22470900	2197503360	2641172
1598411217	2385	355	16339	18	33	21045513011	4340544784	1111618930	1115127808	10495371	14888096	0	22460844	2218009408	2669981
1598411278	2446	363	16823	18	33	21665350996	4360232960	1158556780	1154359296	10452426	15078544	0	22506316	2283434395	2690453
1598411339	2507	371	17386	18	33	22209479401	4375486644	1186649041	1190085584	10366233	14893944	0	22506316	2317446695	2835229
1598411400	2568	380	17791	18	33	22578764207	4391481344	121586188	1224047296	10278660	148221100	0	22518868	2292079299	28164423
1598411461	2629	389	18273	18	33	23149792894	4409581568	1261294959	1266458624	10511730	15045388	0	22529964	37610106903	26156870
1598411522	2690	399	18754	18	33	23971145611	44925898752	1287694427	1293099008	10361269	14846248	0	22529964	3761595640	26180118
1598411583	2752	409	19236	18	33	24736871163	45032497152	1332623278	1338294272	10606996	15040024	0	22529964	3767507915	26240067
1598411644	2812	416	19720	18	33	25266356628	45056057344	1364642257	1370529792	10218504	14678128	0	22529964	3768647955	26277142
1598411705	2873	425	20204	18	33	25847218074	45081169920	1408823380	140392592	14854812	14854812	0	22529674	37721417051	26324444

JSON Output (Right Pane):

```
{
  "Avg": {
    "nprocs": 17.810282776349616,
    "nthreads": 31.943444730077122,
    "pss": 9130922.994858611,
    "rchar": 0.0,
    "read_bytes": 0.0,
    "rss": 15609960.904884318,
    "rx_packets": 1557709.49157926,
    "tx_bytes": 10756.800280423261,
    "swap": 0.0,
    "tx_bytes": 2078446.5300103554,
    "tx_packets": 1841.3957240754758,
    "vmem": 2466610.344473008,
    "wchar": 0.0,
    "write_bytes": 0.0
  },
  "Hm": {
    "cpu": {
      "CPUs": 80,
      "CoresPerSocket": 20,
      "SocketName": "Intel(R) Xeon(R) Gold",
      "Models": 2,
      "ThreadsPerCore": 2
    }
  },
  "Mem": {
    "MemTotal": 394693820
  },
  "Max": {
    "nprocs": 21,
    "nthreads": 36,
    "pss": 12273983,
    "rchar": 0,
    "read_bytes": 0,
    "rss": 24319388,
    "rx_bytes": 368800402888,
    "rx_packets": 254665954,
    "stime": 4426,
    "swap": 0,
    "tx_bytes": 49206972374,
    "tx_packets": 44541818,
    "utime": 162056,
    "vmem": 37694800,
    "wchar": 0,
    "write_bytes": 0,
    "wtime": 23674
  }
}
```

Time-series text file

Summary JSON file

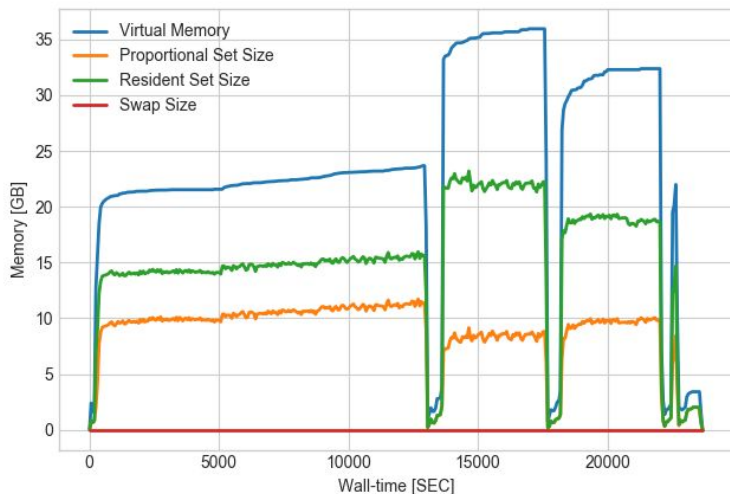
Visualizing the results...

- `prmon_plot.py` can be used to visualize the time-series results, e.g.:

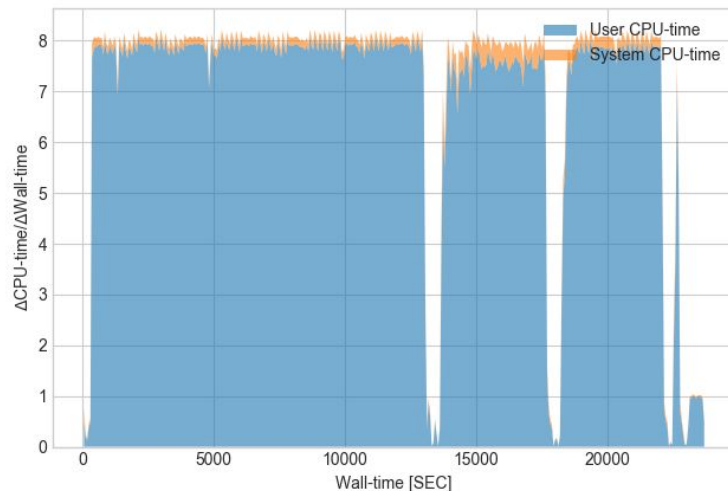
- `prmon_plot.py --input prmon.txt --xvar wtime --yvar vmem,pss,rss,swap --yunit GB`

- `prmon_plot.py --input prmon.txt --xvar wtime --yvar utime,stime --yunit SEC --diff --stacked`

Plot of Wall-time vs Memory



Plot of Wall-time vs Δ CPU-time/ Δ Wall-time



- 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>